Ядра и потоки современных микропроцессоров

Леонид Черняк

Ресурс экстенсивного роста производительности за счет увеличения сложности и тактовой частоты процессоров себя исчерпал.

Ресурс экстенсивного роста производительности за счет увеличения сложности и тактовой частоты процессоров себя исчерпал. Для того чтобы по-прежнему шагать в ногу с законом Мура, требуются новые архитектурные решения, основанные на росте числа процессорных ядер на кристалле и количества обрабатываемых ими потоков.

Для информационной индустрии начало XXI века совпало со сдвигами, которые можно охарактеризовать как «тектонические». К признакам новой эпохи стоит отнести использование сервис-ориентированных архитектур (service-oriented architecture, SOA), кластерных конфигураций и многого-многого другого, в том числе многоядерных процессоров. Но, разумеется, фундаментальная причина происходящего — развитие физики полупроводников, следствием которого стало увеличение числа логических элементов на единицу площади, подчиняющееся закону Гордона Мура. Количество транзисторов на кристалле уже исчисляется сотнями миллионов и скоро преодолеет миллиардный рубеж, в результате чего неизбежно проявляется действие известного закона диалектики, постулирующего взаимосвязь количественных и качественных изменений. В изменившихся условиях на первый план выходит новая категория — **сложность**, причем системы становятся сложными и на микроуровне (процессоры) и на макроуровне (корпоративные информационные системы).

В какой-то мере происходящее в современном компьютерном мире можно уподобить эволюционному переходу, произошедшему миллионы лет назад, когда появились многоклеточные организмы. К тому моменту сложность одной клетки достигла определенного предела, и последующая эволюция пошла по пути развития инфраструктурной сложности. То же самое происходит и с компьютерными системами: сложность одного ядра процессора, равно как и монолитной архитектуры корпоративных информационных систем достигла определенного максимума. Теперь на макроуровне осуществляется переход от монолитных систем к компонентным (или составленным из сервисов), а внимание разработчиков фокусируется на инфраструктурном программном обеспечении промежуточного слоя, а на микроуровне появляются новые архитектуры процессоров.

Буквально в самое последнее время представление о сложности начало терять общеупотребительный смысл, превратившись в самостоятельный фактор. В этом отношении сложность еще не вполне осмыслена, а отношение к ней не вполне определено, хотя, как ни странно, уже почти 50 лет существует отдельная научная дисциплина, которая так и называется — «теория сложных систем». (Напомним, что в теории «сложной» называют систему, отдельные компоненты которой объединены нелинейным способом; такая система не является просто суммой компонентов, как бывает в линейных системах.) Можно лишь удивляться тому, что теория систем пока не воспринята теми специалистами и компаниями, деятельность которых приводит их к созданию этих сложных систем средствами информационных технологий.

«Бутылочное горло» архитектуры фон Неймана

На микроуровне аналогом перехода от одноклеточных организмов к многоклеточным может оказаться переход от одноядерных процессоров к многоядерным (Chip MultiProcessors, CMP). CMP дает одну из возможностей преодоления врожденной слабости современных процессоров — «бутылочного горла» архитектуры фон Неймана.

Вот что говорил Джон Бэкус на церемонии вручения ему Тьюринговской премии в 1977 году: «Что такое компьютер по фон Нейману? Когда 30 лет назад Джон фон Нейман и другие предложили свою оригинальную архитектуру, идея показалась элегантной, практичной и позволяющей упростить решение целого ряда инженерных и программистских задач. И хотя за прошедшее время условия, существовавшие на момент ее публикации, радикально изменились, мы отождествляем наши представления о компьютерах с этой старой концепций. В простейшем изложении фон-неймановский компьютер состоит из трех частей: это центральный процессор (CPU или ЦПУ), память и соединяющий их канал, который служит для обмена данными между CPU и памятью, причем маленькими порциями (лишь по одному слову). Я предлагаю назвать этот канал «бутылочным горлом фон Неймана». Наверняка должно быть менее примитивное решение, чем перекачивание огромного количества данных через «узкое бутылочное горло». Такой канал не только создает проблему для трафика, но еще и является «интеллектуальным бутылочным горлом», которое навязывает программистам «пословное» мышление, не позволяя рассуждать в более высоких концептуальных категориях».

Наибольшую известность Бэкусу принесло создание в середине 50-х годов языка Fortran, который в течение несколько последующих десятилетий был самым популярным средством создания расчетных программ. Но позже, видимо, Бэкус глубоко осознал его слабости и понял, что разработал «самый фон-неймановский язык» изо всех языков высокого уровня. Поэтому основной пафос его критики был обращен прежде всего к несовершенным методам программирования.

С момента произнесения Бэкусом его речи в программировании произошли заметные сдвиги, появились функциональные и объектно-ориентированные технологии, и с их помощью удалось преодолеть то, что Бэкус назвал «интеллектуальным фон-неймановским бутылочным горлом». Однако архитектурная первопричина данного явления, врожденная болезнь канала между памятью и процессором — его ограниченная пропускная способность — не исчезла, несмотря на прогресс в области технологии за прошедшие с тех пор 30 лет. С годами эта проблема постоянно усугубляется, поскольку скорость работы памяти растет гораздо медленнее, чем производительность процессоров, и разрыв между ними становится все больше [1].

Фон-неймановская архитектура компьютера не является единственно возможной. С точки зрения организации обмена командами между процессором и памятью все компьютеры можно разделить на четыре класса:

- *SISD (Single Instruction Single Data)* — «один поток команд, один поток данных»»;
- *SIMD (Single Instruction Multiply Data)* — один поток команд, много потоков данных;
- *MISD (Multiple Instruction Single Data)* — много потоков команд, один поток данных;
- *MIMD (Multiple Instruction Multiple Data)* — много потоков команд, много потоков данных.

Из этой классификации видно, что фон-неймановская машина является частным случаем, попадающим в категорию SISD. Возможные усовершенствования в рамках архитектуры SISD ограничиваются включением в нее конвейеров и других дополнительных функциональных узлов, а также использованием разных методов кэширования. Две другие категории архитектур (SIMD, в которую входят векторные процессоры, и конвейерные архитектуры MISD) были реализованы в нескольких проектах, но не стали массовыми. Если оставаться в рамках этой классификации, то единственной возможностью преодоления ограничений «бутылочного горла» остается развитие архитектур класса MIMD. В их рамках обнаруживается множество подходов: это могут быть и различные параллельные и кластерные архитектуры, и многопотоковые процессоры.

Еще несколько лет назад в силу технологических ограничений все многопотоковые процессоры строились на базе одного ядра, и такая многопотоковость была названа

«одновременной» — *Simultaneous MultiThreading (SMT)*. А с появлением многоядерных процессоров появился альтернативный тип многопотоковости — *Chip MultiProcessors (CMP)*.

Особенности многопотоковых процессоров

Переход от простых однопотоковых процессоров к логически более сложным многопотоковым сопряжен с преодолением специфических, не встречавшихся прежде сложностей. Функционирование устройства, где процесс исполнения разбивается на агенты или потоки (threads) отличает две особенности:

- *принцип недетерминированности (indetermination principle)*. В многопотоковом приложении процесс разбивается на взаимодействующие между собой потоки-агенты без заранее оговоренной определенности;
- *принцип неизвестности (uncertainty principle)*. То, как именно ресурсы будут распределяться между потоками-агентами, также неизвестно заранее.

Из-за этих особенностей работа многопотоковых процессоров принципиально отличается от детерминированных вычислений по фон-неймановской схеме. В данном случае текущее состояние процесса нельзя определить как линейную функцию предшествующего состояния и поступивших на вход данных, хотя каждый из процессов можно рассматривать как фон-неймановскую микромашину. (В приложении к поведению потоков можно даже употребить термин «странность», используемый в квантовой физике.) Наличие этих особенностей приближает многопотоковый процессор к представлениям о сложной системе, но с чисто практической точки зрения понятно, что на уровне выполнения процессов ни о какой недетерминированности или неопределенности, а тем более о странности и речи быть не может. Корректно выполняемая программа не может быть странной.

В самом общем виде многопотоковый процессор состоит из двух типов примитивов. Первый тип — это ресурс, поддерживающий исполнение потока, который называют mutex (от Mutual Exclusion — «взаимное исключение»), а второй — события. То, как физически реализован тот или иной mutex, зависит от выбранной схемы — SMT или CMP. В любом случае выполнение процесса сводится к тому, что очередной поток захватываает mutex на время своего исполнения, а затем освобождает его. Если mutex занят одним потоком, то второй поток не может его заполучить. Конкретная процедура передачи полномочий на обладание mutex от одного потока другому может иметь случайный характер; она зависит от реализации управления, например, в определенной операционной системе. В любом случае управление должно быть построено так, чтобы ресурсы, состоящие из mutex, распределялись корректно и подавлялся эффект неопределенности.

События — это объекты (event), сигнализирующие о об изменении во внешней среде. Они могут переводить себя в режим ожидания до наступления иного события или сообщать о своем состоянии другому событию. Таким способом события могут взаимодействовать между собой, и при этом должна обеспечиваться преемственность данных между событиями. Ожидающий исполнения агент необходимо информировать о готовности данных для него. И как в распределении mutex должен подавляться эффект неопределенности, так при работе с событиями должен подавляться эффект неизвестности. Впервые схема SMT была реализована в процессорах Compaq Alpha 21464, а также в Intel Xeon MP и Itanium [2].

Логически CMP проще: здесь параллелизм обеспечивается тем, что каждый из потоков обрабатывается собственным ядром. Но если приложение не может быть разделено на потоки, то оно (при отсутствии специальных мер) обрабатывается одним ядром, и в таком случае общая производительность процессора ограничивается быстродействием одного ядра. На первый взгляд, процессор, построенный по схеме SMT, более гибок, а следовательно, эта схема предпочтительна. Но такое утверждение справедливо лишь при невысокой плотности размещения транзисторов. Если частота измеряется мегагерцами и число транзисторов в кристалле приближается к миллиарду, а задержки при передаче

сигналов становятся большими, чем время переключения, то преимущества получает микроархитектура CMP, в которой связанные вычислительные элементы локализованы.

Однако физическое распараллеливание приводит к тому, что CMP не слишком эффективна при последовательных вычислениях. Для преодоления этого недостатка используется подход, получивший название «спекулятивная многопотоковость» (Speculative Multithreading). В русском языке слово «спекулятивный» имеет отрицательный смысловой оттенок, поэтому мы назовем такую многопотоковость «условной». Данный подход предполагает аппаратную или программную поддержку деления последовательного приложения на условные потоки, согласование их выполнения и интеграцию результатов в памяти [3].

Эволюция CMP

Первые массовые CMP-процессоры предназначались для серверного рынка. Вне зависимости от вендора они, в сущности, представляли собой два независимых суперскалярных процессора на одной подложке. Основная мотивация создания подобных конструкций состоит в уменьшении объема, с тем чтобы в одном конструктиве можно было «упаковывать» больше процессоров, повышая удельную мощность на единицу объема (что критически важно для современных центров обработки данных). Тогда на общем системном уровне достигается некоторая дополнительная экономия, поскольку процессоры, находящиеся на одном кристалле, используют общие системные ресурсы, такие как высокоскоростные коммуникации. Обычно между соседствующими процессорами имеется лишь общий системный интерфейс (*рис. 1*, б).

Апологеты использования CMP-процессоров обосновывают дальнейшее (свыше двух) увеличение числа ядер особенностями серверной нагрузки, отличающей этот тип компьютеров от встроенных или предназначенных для массивных вычислений систем. От сервера требуется большая общая производительность, но задержка отдельного обращения к системе является не столь критичной. Тривиальный пример: пользователь может просто не заметить миллисекундную задержку появления обновленной Web-страницы, но весьма болезненно реагирует на перегрузку сервера, которая может стать причиной перебоев в обслуживании.

Специфика нагрузки дает CMP-процессорам еще одно заметное преимуществ. Скажем, заменяя одноядерный процессор двухъядерным, можно при той же производительности вдвое уменьшить тактовую частоту. При этом теоретически время обработки отдельного запроса может возрасти вдвое, но поскольку физическое разделение потоков уменьшает ограничение «бутылочного горла» фон-неймановской архитектуры, суммарная задержка окажется значительно меньшей, чем двукратная. При меньших частоте и сложности одного ядра существенно сокращается потребление энергии, а при увеличении числа ядер перечисленные аргументы в пользу CMP только усиливаются. Поэтому следующий логически оправданный шаг состоит в том, чтобы собрать несколько ядер и объединить их общей кэш-памятью, например как в проекте Hydra (рис 1, в). А далее можно усложнить ядра и сделать их многопотоковыми, что и было реализовано в проекте Niagara (рис 1, г).

Сложность процессоров имеет еще одно важное проявление. Проектирование изделия, насчитывающего миллиарды компонентов, становится все более трудоемкой задачей — несмотря на использование средств автоматизации. Показательно, что мы являемся свидетелями более чем десятилетнего «доведения до ума» архитектуры IA-64. Проектирование CMP-процессора существенно проще: если есть проработанное ядро, то оно может тиражироваться в нужных количествах, а проектирование ограничивается созданием внутренней инфраструктуры кристалла. К тому же однотипность ядер упрощает проектирование системных плат, которое сводится к масштабированию, а в конечном счете, меняются показатели подсистем ввода/вывода.

Несмотря на приведенные аргументы, пока нет достаточных оснований для однозначного утверждения о преимуществах CMP по сравнению с SMT. Опыт создания процессоров,

реализующих SMT, является гораздо большим: начиная с середины 80-х годов созданы несколько десятков экспериментальных изделий и несколько серийных процессоров. История развития CPM пока короткая: если не учитывать семейство специализированных сигнальных процессоров Texas Instruments TMS 320C8x, то первым успешным проектом стал Hydra, выполненный в Стэндфордском университете. Среди университетских исследовательских проектов, нацеленных на построение CMP-процессоров, известны еще три — Wisconsin Multiscalar, Carnegie-Mellon Stampede и MIT M-machine.

Микропроцессор Hydra

Микропроцессор Hydra разработан авторским коллективом под руководством профессора Кунле Олукотуна.

Кристалл Hydra состоит из четырех процессорных ядер на основе известной RISC-архитектуры MIPS . Каждое ядро имеет кэш-память команд и кэш-память данных, а все ядра объединены в общую кэш-память второго уровня. Процессоры выполняют обычный набор команд MIPS плюс команды условного хранения (Store Conditional или SC), предназначенные для реализации синхронизационных примитивов. Процессоры и кэш-память второго уровня объединяются шинами чтения/записи, а кроме того, есть вспомогательные адресные и управляющие шины. Все эти шины являются виртуальными, то есть логически представляются проводными шинами, а физически разделены на множество сегментов, использующих повторители, и буферов, что позволяет повысить скорость работы ядер.

Шина чтения/записи играет роль системной. Благодаря ее расположению внутри кристалла она имеет достаточную пропускную способность для обеспечения обмена с кэш-памятью за один цикл. Достичь таких показателей производительности обмена даже в самых дорогих традиционных мультипроцессорных архитектурах сложно из-за физических ограничений на число внешних контактов процессоров. Эффективные шины обмена с кэш-памятью предотвращают проблему возникновения «бутылочного горла» между ядрами и памятью.

Тестирование Hydra при нагрузках с явно выраженным параллелизмом на типичных Web- и серверных приложениях показало, что производительность четырех ядер по сравнению с одним ядром возрастает в 3-3,8 раз, то есть практически линейно. Это дает основания полагать, что процессоры такого типа вполне удачно «впишутся» в те приложения, в которых используются серверы с SMP-архитектурой. Но понятно, что процессор должен достаточно эффективно работать и с последовательными приложениями, поэтому одна из важнейших задач заключается в реализации условной многопотоковости. В Hydra она реализована на аппаратном уровне, и выбор этого подхода обоснован тем, что он не требует дополнительных затрат на программирование параллельных приложений.

Условная многопотоковость базируется на разбиении последовательности команд программы на потоки, которые могут выполняться параллельно. Естественно, между такими потоками может быть логическая взаимозависимость, и для их согласования в процессор встраивается специальный механизм синхронизации. Суть его работы сводится к тому, что если какому-то потоку требуются данные из параллельного потока, а они еще не готовы, то выполнение такого потока приостанавливается. На деле здесь проявляются элементы недетерминированности, о которых шла речь выше. Процесс синхронизации довольно сложен, поскольку необходимо определить все возможные зависимости между потоками и условия синхронизации. Условная синхронизация позволяет распараллеливать программы без предварительного знания их свойств. Важно, что механизм синхронизации является динамическим, он работает без вмешательства программиста или компилятора, который способен только на статическое деление приложений на потоки. Испытания модели на основе разных тестов показали, что средства условной многопотоковости позволяют увеличить производительность процессора в несколько раз, и чем более явным параллелизмом характеризуется тест, тем меньше такое значение.

В 2000 году в обстановке строгой секретности была создана компания Afara. Ее

основателями стали профессор Кунле Олукотун из Стэнфордского университета и известный разработчик процессоров Лес Кон, имевший опыт работы в Intel и Sun Microsystems. Кон был одним из авторов RISC-процессоров i860 и i960 в первой из этих корпораций и UltraSPARC-I — во второй. Под его руководством осуществлена переработка Hydra под процессорные ядра на базе процессора SPARC. В 2002 году Afara была куплена Sun Microsystems, и на этом закончилась история проекта Hydra и началась история Niagara.

Niagara — «сплав» MAJC и Hydra

У процессора UltraSPARC T1, более известного как Niagara, два основных предшественника — Hydra и MAJC.

В середине 90-х годов, на волне увлечения специализированными Java-процессорами, в Sun Microsystems была предпринята попытка создания процессора «с очень длинным словом» — Very Long Instruction Word (VLIW). Эта инициатива получила название MAJC (Microprocessor Architecture for Java Computing). Как и в других проектах, стартовавших в то время (Intel IA-64 Itanium), в данном случае ставилась задача переноса некоторых из самых сложных операций в ведение компилятора. Освободившуюся транзисторную логику можно использовать для создания более производительных функциональных узлов (functional units), с тем чтобы обеспечить продуктивный обмен командами и данными между CPU, кэш-памятью и основной памятью. Таким образом, преодолевалось фон-неймановское «бутылочное горло».

MAJC отличался от большинства процессоров отсутствием специализированных сопроцессоров (subprocessors), которые обычно называют функциональными устройствами, предназначенными для выполнения операций с целыми числами, числами с плавающей точкой и мультимедийными данными. В нем все функциональные устройства были одинаковыми, способными к выполнению любых операций, что, с одной стороны, снижало эффективность выполнения отдельных операций, но с другой повышало коэффициент использования всего процессора.

Niagara воплощает в себе лучшее из двух альтернативных подходов к реализации многопотоковости — SMT и CMP. На первый взгляд, он очень похож на Hydra, но скорее Hydra можно назвать «макетом» Niagara. Помимо того что в последнем — вдвое больше ядер, каждое из них может обрабатывать четыре потока.

Процессор Niagara обеспечивает аппаратную поддержку выполнения 32 потоков, которые разделены на восемь групп (по четыре потока в каждой). Для выполнения каждой группы служит свой обрабатывающий канал SPARC pipeline (*рис.2*). Он представляет собой процессорное ядро, построенное в соответствии с архитектурой SPARC V9. Каждый SPARC pipeline содержит кэш-память первого уровня для команд и данных. Совместно 32 потока используют кэш-память второго уровня емкостью 3 Мбайт, разделенную на четыре банка. Коммутатор соединяет восемь ядер, банки кэш-памяти второго уровня и другие распределяемые ресурсы CPU, причем поддерживает скорость обмена 200 Гбайт/с. Кроме того, в коммутаторе находится порт для систем ввода/вывода и каналы к памяти типа DDR2 DRAM, обеспечивающие скорость обмена 20 Гбайт/с; максимальная емкость памяти составляет до 128 Гбайт.

Проект Niagara ориентирован на операционную систему Solaris, поэтому все приложения, работающие под управлением Solaris, могут выполняться на новом процессоре без каких-либо изменений. Прикладное программное обеспечение воспринимает Niagara как 32 дискретных процессора.

Проект Cell

Собственный подход к созданию многоядерных процессоров предложила корпорация IBM, чей проект Cell назван «гетерогенным мультипроцессорным чипом» (heterogeneous chip multiprocessor). Архитектуру Cell именуют еще и Cell Broadband Engine Architecture (CBEA).

Мультипроцессор Cell состоит из ядра IBM 64-bit Power Architecture и восьми специализированных сопроцессоров, реализующих схему «одна команда много данных». В компании IBM эту архитектуру называют Synergistic Processor Unit (SPU). Она может с успехом использоваться при необходимости обрабатывать большие потоки данных, например в криптографии, в разных мультимедийных и научных приложениях, таких как быстрое преобразование Фурье или матричные операции. Архитектура Cell создана группой исследователей IBM Research совместно с коллегами из IBM Systems Technology Group, Sony и Toshiba, а ее первым приложением должны стать мультимедийные устройства, требующие больших объемов вычислений.

Основа Synergistic Processor Unit — набор команд Instruction Set Architecture (ISA). Команды имеют длину 32 бит и адресуются трем операндам, размещаемым в регистровом пуле, который состоит из 128 регистров по 128 бит в каждом.

В перспективе применение Cell не будет ограничено игровыми системами. На очереди — телевидение высокой четкости, домашние серверы и даже суперкомпьютеры.

Литература

1. Леонид Черняк. Ревизия первооснов — конец застоя? // Открытые системы. — 2003, №5.
2. Михаил Кузьминский. Многонитевая архитектура микропроцессоров // Открытые системы. — 2002, №1.
3. Rajat A Dua, Bhushan Lokhande. A Comparative study of SMT and CMP multiprocessors. — *wwwece.umn.edu/users/rajat/SmtCmp.pdf*.

12.12.2005г.

Постоянный URL статьи: http://www.osp.ru/os/2005/12/380625/