Модель программирования: смена парадигмы

Виктор Корнеев

Эффективное использование суперкомпьютеров с миллионами процессоров невозможно без средств программирования, учитывающих мельчайшие гранулы параллелизма и взаимодействия между ними. Освоение такой парадигмы параллельных вычислений позволит существенно сократить затраты на подготовку пользователей суперкомпьютеров.

После ошеломляющего рывка в росте производительности, вызванного построением «параллельных» суперкомпьютеров из коммерчески доступных компонентов, и выявления проблем, препятствующих дальнейшему повышению быстродействия, пришло осознание того, что параллельная обработка требует особых архитектурных решений. Без учета особенностей преодоления энергетических ограничений и разрыва между быстродействием логических элементов и элементов памяти дальнейшее увеличение производительности невозможно. Наступил момент смены парадигмы суперкомпьютерных вычислений вообще и парадигмы программирования в частности.

Существующие модели программирования

Первая модель программирования базировалась на архитектуре с последовательным исполнением команд, и возникла она в то время, когда повышение производительности компьютеров осуществлялось преимущественно за счет увеличения тактовой частоты. Тогда расцвели алгоритмические языки программирования, и повышение эффективности программирования виделось в создании типизированных, объектно-ориентированных, функциональных и других языков высокого уровня. Однако для получения высокой производительности требовалось привлечение низкоуровневых средств – команд процессора.

Для преодоления все возрастающего разрыва в быстродействии логических схем и элементов памяти пришлось вводить дополнительные механизмы: виртуальную память, многоуровневую кэш-память, таблицы предсказаний направлений переходов и т.п. Все эти механизмы потребовали дополнительного оборудования, что само по себе снизило показатель максимально достижимой производительности на единицу оборудования. Кроме того, единообразно автоматически функционирующие механизмы замены страниц виртуальной памяти и строк кэш-памятей создают при исполнении ряда программ неприемлемо большие потери производительности, поэтому в систему команд процессоров были добавлены команды управления содержимым кэш-памяти. Это позволяло перемещать данные между уровнями памяти в соответствии с потребностями алгоритма.

Повышение производительности достигалось также за счет использования параллелизма на уровне функциональных устройств и реализации конвейерной обработки в векторных процессорах. Однако для эффективной загрузки функциональных устройств потребовалось дополнительное оборудование для реализации суперскалярной архитектуры, а неудовлетворенность достигаемым на этом пути результатом породила архитектуру процессоров с длинным командным словом (Very Large Instruction Word, VLIW), позволяющих программисту самому указывать параллельно выполняемые преобразования. В течение нескольких лет разработчикам компиляторов с языков высокого уровня удалось справиться с проблемой эффективной загрузки векторных процессоров и функциональных устройств в суперскалярных и VLIW-процессорах, однако акцент на неявном для программиста использовании параллелизма, скрытого компиляторами языков высокого уровня, вызвал существенное замедление в 80-90-х годах темпов роста производительности вычислителей –

лучшие показатели быстродействия демонстрировали системы с небольшим числом векторных процессоров.

Вторая модель программирования ассоциируется с построением массово-параллельных систем из коммерчески доступных микропроцессоров с явным для программиста использованием параллелизма, например, средствами библиотеки пересылки сообщений MPI (Message Passing Interface). Здесь удалось достигнуть значений производительности, прямо пропорциональных произведению тактовой частоты на число используемых процессоров с коэффициентом пропорциональности, равным количеству результатов, получаемых процессором за такт. Однако сам подход к программированию с явным указанием межпроцессорных обменов неудобен и малопродуктивен, что выразилось в эмоциональной оценке таких систем – «преждевременный терафлоп» [1].

Попытки создать специальные языки для параллельного программирования не привели к повышению продуктивности труда программистов: проблема оказалась не в средствах представления, а в модели параллельных вычислений, которая формируется интеллектом программиста. Поэтому стала популярной идея «инкрементального распараллеливания», при котором вместо создания новой параллельной программы в текст последовательной добавляются специальные директивы, облегчающие компилятору принятие решения о последовательном или параллельном исполнении секций программного кода. Но в целом модель параллельного программирования так и не была создана, возможно потому, что обработка данных и их доставка рассматривались отдельно.

Новая модель программирования

Ограничение скорости распространения сигналов на кристалле, высокое энергопотребление и тепловыделение привели к тому, что для дальнейшего повышения производительности стали размещать на кристалле сеть «простых» процессорных ядер, коммуникации между которыми как внутри кристаллов, так и между ядрами разных кристаллов были основаны на принципе близкодействия. Это позволяет преодолеть ограничения на рост тактовой частоты за счет использования только коротких проводников [2].

При реализации на кристалле фрагмента прямоугольной решетки процессорных ядер, ориентированной на исполнение параллельных программ с большой совокупностью асинхронно взаимодействующих нитей, или тредов (thread), и процессов, сигналы, включая тактовый, должны синхронно распространяться только внутри области кристалла, занимаемой одним простым процессорным ядром (без внеочередного исполнения команд и других аппаратно затратных приемов повышения производительности). Все процессорные ядра функционируют на одной тактовой частоте, но между ними возможен произвольный сдвиг фаз тактовых сигналов. Линии связи между процессорными ядрами обеспечивают асинхронную передачу данных, что компенсирует возможный сдвиг фаз тактовых сигналов между ядрами.

Возможны и другие, более специализированные реализации как ядер, так и связей между ними, ориентированные на программы с жесткими «связками» тредов, протекание которых организуется синхронно, что уменьшает объем управляющей логики. В качестве примера можно назвать кристалл nVidia Fermi. Кроме того, сегодня имеются реализации Intel Teraflops Research Chip и его развитие – 48-ядерный кристалл Single-Chip Cloud Computer, кристалл Tile 64 [3], содержащий сотню процессорных ядер, и можно утверждать, что использование многоядерных кристаллов позволяет преодолеть ограничения роста тактовой частоты и создать в ближайшие годы суперкомпьютеры, имеющие 10 млн и более процессорных ядер. Однако современные параллельные суперкомпьютеры даже с 1 тыс. процессорных ядер показывают низкую реальную производительность при вычислениях с интенсивным доступом к памяти по адресам, определяемым в ходе вычислений.

Для преодоления разрыва в быстродействии процессоров и блоков памяти необходимо работать с памятью в поточном режиме, организуя поток запросов к памяти и получая

ответы в том же темпе – это создает режим, при котором время выполнения запроса к памяти определяется темпом их выдачи, а не временем выполнения одного запроса. Для реализации этого режима необходимо построить многоблочную расслоенную память и ввести в процессоры механизм отложенных обращений к памяти, допускающий исполнение команд, следующих за командой обращения к памяти, не дожидаясь ее завершения.

Запуск новых команд на выполнение прекращается при достижении предельных количеств незавершенных обращений к памяти по чтению или по записи, а также в том случае, если выполнение очередной команды невозможно без завершения предшествующего обращения к памяти. Однако интенсивности потока обращений к памяти, создаваемой аппаратурой и компиляторами суперскалярных и VLIW-процессоров, недостаточно для скрытия задержки обращения к памяти. Для эффективной работы в поточном режиме пользователи должны готовить программы в виде совокупности асинхронных и синхронных тредов.

Возможны разные реализации тредовой модели: от чисто аппаратной, в которой процессор каждый такт переключается на новый набор регистров, сохраняющих контекст треда (Tera MTA, MTA-2 и Eldorado Cray XMT), до программной, в которой вводятся стандартные треды POSIX (pthread). Кроме того, аппаратные средства процессора могут выполнять формируемые пользователем «связки» тредов, синхронно загружая в процессоры совокупность тредов, образующих связку (nVidia Fermi и TRIPS [4]). Чтобы отличать отдельные треды от их связок, можно говорить об асинхронных и синхронных тредах.

Существующие средства программирования асинхронных тредов, например pthread, используют для взаимодействия тредов примитивы ОС, что вызывает большие задержки при исполнении программ с плохой локальностью данных. Кроме того, эти примитивы способны обеспечить взаимодействие не более сотни тредов, но для достижения петафлопного уровня производительности их требуются миллионы. Поэтому необходимо составлять программы из «легких» тредов [5], образованных из небольшого числа команд и требующих компактного стека для сохранения своего состояния, а также использующих иной механизм межтредовой синхронизации и коммуникации – добавление к каждому слову памяти full/empty (FE) бита и изменение семантики команд обращения к памяти. Значение FE-бита устанавливает, что слово памяти имеет (или не имеет) содержимое.

Команды writeef, readfe, readff и writeff, обращающиеся к ячейке памяти, могут выполняться только в том случае, если в первом компоненте их суффикса определено значение бита FE, и оставляют после выполнения значение этого бита, заданное программистом во втором компоненте суффикса команды. Выполнение команды задерживается, если FE-бит не имеет требуемого значения. Например, команда writeff требует, чтобы перед ее выполнением значение FE-бита слова памяти, в которое будет запись, было full, и оставляет после выполнения это же значение.

Синхронизация на базе FE-бита не требует специальных разделяемых переменных, таких как «замки», семафоры, и других механизмов, а также механизмов синхронизации, весьма затратных по времени их определения и исполнения при миллионах тредов. Кроме того, применение разделяемых переменных и неделимых (атомарных) последовательностей команд, определяющих значение условия ветвления и выполняющих переход в соответствии с полученным значением, существенно сложнее и более длительно, чем выполнение команд доступа к памяти при синхронизации динамически порождаемых легких тредов. Поэтому синхронизация на базе FE-бита позволит выдавать максимально возможное в исполняемой программе количество обращений к памяти, генерируемых легкими тредами, и параллельно выполнять эти доступы в расслоенной памяти.

Для выполнения на традиционных процессорах программ, созданных на базе модели легких тредов, синхронизация которых реализуется FE-битами, в Национальной лаборатории в Сандии Министерства энергетики США разработана библиотека Qthreads [5]. В рамках этой библиотеки FE-биты эмулируются хеш-таблицей. Для поддержки локальности легких тредов в распределенной разделяемой памяти вводятся «наставники», под управлением которых

порождаются и протекают qthread-треды в одном и том же с наставником, реализованном как pthread сегменте памяти, в частности в памяти одного Unix-процесса.

Введение в модель суперкомпьютерного программирования асинхронных легких тредов с контролируемой пользователем привязкой тредов к сегментам разделяемой памяти, а также синхронных тредов требует от программиста дополнительных усилий, однако асинхронные легкие треды и синхронные треды необходимы для эффективной работы памяти в поточном режиме, и пока нельзя рассчитывать на их автоматическое формирование компилятором. Кроме сокращения времени выполнения доступа к памяти, в модели суперкомпьютерного программирования должен использоваться еще один прием повышения производительности – уменьшение количества обращений к памяти за счет непосредственной передачи операндов между процессорными ядрами, минуя промежуточное хранение операндов в памяти.

В рамках модели суперкомпьютерного программирования программы, исполняемые ядрами, являются мультитредовыми с множеством легких тредов и синхронных тредов, протекающих в памяти ядра, а межъядерные потоки программируются исходя из параметрического описания графов связей подсистем, на которых эти программы способны выполняться [2, 6]. Подготовка программ, способных масштабироваться от нескольких до миллионов и более ядер, возможна при программировании межъядерных обменов, исходя из требуемого программой типа графа межъядерных связей и алгоритма нумерации ядер, приписывающего каждому ядру уникальный номер. Пользователь должен запрограммировать путевую процедуру, управляющую передачами данных. Для выполнения таких программ операционная система суперкомпьютера должна формировать связную подсистему процессорных ядер с требуемым программой количеством ядер в подсистеме и типом графа межъядерных связей.

Подход к программированию в рамках новой модели напоминает MPI при работе с группой процессов с заданной топологией коммуникатора. Принципиальное отличие состоит в том, что в библиотеке MPI топология группы процессов определяет виртуальные связи между процессами группы, в то время как в новой модели атрибуты окружения в каждом процессоре получают реальные значения в подсистеме, сформированной для исполнения программы.

Таким образом, модель суперкомпьютерного программирования требует от пользователя, во-первых, представления алгоритма вычислений как потоковой схемы или клеточного автомата с локальными передачами данных и команд между процессорными ядрами по принципу близкодействия по каналам «точка-точка», и, во-вторых, представления вычислений в ядрах как протекания систем асинхронных легких тредов и, если аппаратура позволяет, синхронных тредов. Однако вопрос о том, следует ли в рамках этой модели создавать специальный язык программирования для подготовки программ или же ограничиться библиотеками для работы с легкими тредами и окружениями, а также синхронными тредами, пока открыт.

Исполнение программ, созданных на основе модели суперкомпьютерного программирования, возможно и на процессорных ядрах традиционной архитектуры, но для повышения эффективности их выполнения требуется введение в процессорные ядра многоядерных кристаллов механизмов управления глобальным адресным пространством, порождения и уничтожения легких тредов и их синхронизации на базе расширения слов памяти FE-битами.

* * *

Новая модель суперкомпьютерного программирования повысит эффективность использования существующих суперкомпьютеров, так как эти программы смогут полнее загружать процессорные и коммуникационные ресурсы, а прикладные программы будут адаптируемы для будущего поколения экзафлопных суперкомпьютеров.

Литература

1. Gordon Bell, Ultracomputers: A Teraflop Before Its Time. Science, Vol. 256. 3 April 1992
2. В.В. Корнеев. Архитектура вычислительных систем с программируемой структурой. Новосибирск: Наука, 1985. 168 с. http://andrei.klimov.net/reading/1985.Korneev-arkhitektura.vychislitel'nykh.sistem.s.programmiruemoi.strukturoi.zip.
3. D. Wentzlaff et al, On-Chip Interconnection Architecture of the Tile Processor. IEEE Micro, September-October 2007.
4. Doug Burger et al, Scaling to the End of Silicon with EDGE Architectures. Computer, July 2004,Vol. 37, No. 7.
5. K. Wheeler et al, Qthreads: An API for Programming with Millions of Lightweight Threads. Workshop on Multithreaded Architectures and Applications at IEEE IPDPS, April 2008.
6. В. Корнеев. Подход к программированию суперкомпьютеров на базе многоядерных мультитредовых кристаллов. Москва, НИВЦ МГУ им. М.В. Ломоносова, Научный журнал «Вычислительные методы и программирование», 2009, т 10.

Виктор Корнеев *(korv@rdi-kvant.ru) – сотрудник НИИ «Квант» (Москва).*

Программная настраиваемость аппаратной структуры

Кроме наращивания тактовой частоты и повышения параллелизма есть еще третий путь увеличения производительности – программная настраиваемость структуры для аппаратной реализации вычислений. http://www.osp.ru/os/2007/10/4705462

Оценка быстродействия нерегулярного доступа к памяти

Появление приложений, интенсивно взаимодействующих с памятью, стимулировали создание вычислительных систем с новой архитектурой. Однако для оценки таких систем традиционные тесты уже не подходят. http://www.osp.ru/os/2008/01/4836914

27.04.2010г.

Постоянный URL статьи: http://www.osp.ru/os/2010/03/13001875/

© 1992-2011 Все права защищены. Издательство "Открытые системы"