УДК 681.31.323

ЭФФЕКТИВНОСТЬ МОДЕЛИ ПРОГРАММИРОВАНИЯ НА БАЗЕ ЯВНОГО ЗАДАНИЯ ЛЕГКИХ ТРЕДОВ

В.В. Корнеев, А.В. Будник

Представлены результаты исследования эффективности выполнения параллельных программ, созданных на основе разных моделей программирования. Программы исполняются на *SMP*, образуемом на материнской плате несколькими многоядерными кристаллами. Одна модель основана на неявном для пользователя распределении порождаемых параллельных тредов по предоставляемым аппаратурой SMP ресурсам и традиционных подходах к синхронизации тредов. Эта модель представлена *Open MP*. Вторая модель использует явное задание распределения по ресурсам SMP легких тредов с использованием разработанной в *Sandia National Laboratory* библиотеки *Qthreads*.

Ключевые слова: масштабируемые параллельные программы, эффективность параллельных вычислений, легкие треды.

Не вызывает сомнения, что суперкомпьютеры экзафлопсного и более высокого уровня производительности будут строиться на базе существенно многоядерных кристаллов и содержать 10-ки и более миллионов процессорных ядер, иначе откуда взяться такой производительности? Однако для того, чтобы достичь этой производительности на реальных задачах, необходима разделяемая всеми процессорными ядрами память с поддержкой множественного доступа для получения высокой производительности на программах с интенсивным обращением к разделяемой памяти по адресам, вычисляемым в ходе исполнения программ [1].

Прежде, чем переходить к изложению модели программирования, подходящей для суперкомпьютеров экзафлопсного уровня производительности, представим архитектуру, необходимую для этой модели программирования.

Для реализации разделяемой памяти подходящим узлом суперкомпьютера представляется конструктивный элемент, содержащий локальный блок памяти и существенно многоядерный кристалл со встроенными контроллерами памяти и высокоскоростными портами для подсоединения к коммуникационной среде, объединяющей узлы суперкомпьютера. Конечно, возможны и другие структуры узла, а именно: отдельный коммуникационный кристалл для одного или нескольких кристаллов с процессорными ядрами, но принципиально суперкомпьютер будет иметь распределенные по узлам локальные блоки разделяемой всеми узлами памяти с введенным в каждое слово памяти *FE* битом [2]. Встроенные в кристалл контроллеры памяти должны поддерживать высокую степень расслоения локального блока памяти, а процессорные ядра допускать большое число незавершенных обращений к памяти. На уровне ОС суперкомпьютера должно программно формироваться глобальное адресное пространство разделяемой памяти, состоящей из блоков локальной памяти узлов, имеющих достаточно большой объем.

Процессорные ядра должны быть мультитредовыми и иметь команды *writeef*, *readfe*, *readff* и *writeff*, а также команды атомарных операций, например *fetch_ and_ add*, *swap* и другие, для поддержки синхронизации и коммуникации между легкими тредами [2], *POSIX* тредами и процессами исполняемых программ. Эти команды, обращающиеся к ячейке памяти, могут выполняться только при определенном в них в первом компоненте суффикса значении бита *FE* ячейки памяти 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.

Для достижения высокой производительности и энергоэффективности аппаратура ядер и связей между ними может включать специализированные функциональные устройства, ориентированные на программы с жесткими «связками» тредов как, например, в *NVIDIA Fermi*. Загрузка в функциональное устройство «связки» тредов, протекание которых организуется синхронно, уменьшает объем управляющей логики.

Синхронизация на базе *FE* бита не требует специальных разделяемых переменных, таких как «замки», семафоры и другие, 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.

Ясно, что из-за разрыва в быстродействии памяти и процессорных ядер необходимо, во-первых, минимизировать количество обращений к памяти и, во-вторых, максимизировать количество одновременных доступов к памяти [3, 4]. Первое возможно, например, при выборе алгоритма решения задачи, допускающего создание потоковой программы. В этом случае прикладная программа подготавливается пользователем исходно как потоковая или преобразуется в таковую компилятором. При этом результаты, полученные в ядре, вместо того, чтобы быть записанными в память, передаются в одно или несколько процессорных ядер, как того требует потоковая программа.

Максимизация количества одновременных доступов к памяти достигается, например, при мультитредовой архитектуре процессорных ядер и возможности задавать протекание большого числа асинхронных и синхронных легких тредов, локализуемых в областях разделяемой памяти процессорных ядер.

Технологически не вызывает сомнений возможность создания мультиядерных кристаллов со сколь угодно разнообразной архитектурой процессорных ядер, включая архитектуру с асинхронными и синхронными легкими тредами кратко рассмотренную выше. Однако подходы к программированию и эффективной загрузке миллионов ядер, объединенных накристальными и межкристальными коммуникационными средами, служат предметом интенсивных исследований. Проблема имеет фундаментальный характер, определяющий дальнейшее развитие вычислительной техники.

Анализ ретроспективы развития микроэлектроники показывает, что как только появляется возможность, в кристалле реализуется вся функциональность материнских плат и блоков, состоящих из совокупности плат, за исключением, пожалуй, памяти большого объема. Поэтому на базе передовых суперкомпьютеров, построенных из многоядерных кристаллов *IBM*, *AMD* и *Intel* с большой уверенностью представляется возможным прогнозировать развитие суперкомпьютеров в целом, следуя эволюционному пути совершенствования микропроцессорных кристаллов и архитектур суперкомпьютеров.

Поэтому представляется возможным вести исследования по моделям параллельного программирования на существующих суперкомпьютерах, построенных на базе многоядерных кристаллов со сравнительно небольшим числом 4 – 8 ядер на кристалле.

В настоящей работе представлены результаты исследования эффективности выполнения параллельных программ, созданных на основе разных моделей программирования, на

SMP, образуемом на материнской плате многоядерными кристаллами. Одна модель основана на неявном для пользователя распределении порождаемых параллельных тредов по предоставляемым аппаратурой *SMP* ресурсам и традиционных подходах к синхронизации тредов. Эта модель представлена *Open MP*. Вторая модель использует явное задание распределения по ресурсам *SMP* легких тредов, синхронизация которых реализуется *FE* битами. Для порождения и завершения асинхронных тредов в программах на традиционных языках программирования, например *C*, могут быть использованы 3 дополнительные команды: *Spawn (n), Joint* и *Prefix-Sum (b, e)* [5], позволяющие соответственно породить заданное параметром n количество тредов, завершить тред, исполняющий команду *Join*, а также выполнить как неделимую операцию присвоение переменной, например номеру треда, значения параметра b и увеличить значение b, прибавив к b значение e. В совокупности с общей памятью, слова которой снабжены *FE* битами, эти команды позволяют задать синхронизацию и межтредовые коммуникации.

Для выполнения на традиционных процессорах программ, созданных на базе модели легких тредов, синхронизация которых реализуется *FE* битами, в *Sandia National Laboraroties* разработана библиотека *Qthreads* [6]. В рамках этой библиотеки *FE*-биты эмулируются хэш-таблицей. Для поддержки локальности легких тредов в распределенной разделяемой памяти вводятся «наставники–*shepherds*», под управлением которых порождаются и протекают *qthreads*-треды в одном и том же с наставником, реализованным как *pthread*, сегменте памяти, в частности, в памяти одного *Unix*-процесса.

Были подготовлены по две версии параллельных программ, реализующих умножение матрицы на вектор и решение двумерной краевой задачи для уравнения теплопроводности методом Якоби при одном и том же подходе к распараллеливанию с распределением равных блоков строк матрицы по параллельным ветвям. Для каждой задачи были подготовлены версии программы с использованием компилятора *gcc–4, 1, 2* и библиотек *OpenMP* и *qthread–1, 5, 1* [6]. Тестирование проводилось на двух серверах с характеристиками: сервер 1: 2 ядра: *Intel(R) Xeon(R) CPU 5110 @ 1,60GHz, RAM 2GB, OC Linux RHEL 5,5 x86_64;* сервер 2: 12 ядер: *Intel(R) Xeon(R) CPU X5670 @ 2,93GHz RAM 56 GB, OC Linux RHEL 5,5 x86_64*.

Каждая версия программы умножения матрицы на вектор использовала одинаковые файлы со значениями матрицы и вектора размером 1831 *MB* и 16 *MB* для 2-х ядерного сервера и 3749 *MB* и 32,76 *MB* для 12-ядерного сервера соответственно.

Время выполнения умножения матрицы на вектор с применением библиотек *OpenMP* и *qthreads* на 2-х ядерном сервере составляет соответственно 5,5 с и 3,0 с. Для 12-ти ядерного сервера эти времена равны соответственно 0,22 с и 0,195 с.

Время получения решения двумерной краевой задачи для уравнения теплопроводности методом Якоби с применением библиотек и *OpenMP* и *qthreads* на 2-х ядерном сервере при размере задачи 1024x960 и числе итераций 1000 с библиотекой *OpenMP* равно 31 с, а с библиотекой qthreads 23,3 с. Соответственно, при том же размере задачи, но числе итераций 10000 на 12-ти ядерном сервере время выполнения задачи с библиотекой *OpenMP* 20,3 с, а с qthreads 17 с.

Следует отметить, что из многих опробованных вариантов порождения различного количества наставников и легких тредов наиболее производительным оказался запуск на каждом ядре одного «наставника» и одного легкого треда (*qthread*), что объясняется работой только с памятью при отсутствии обменов между ядрами через коммуникационную среду.

Полученные результаты позволяют сделать вывод о перспективности использования библиотеки *qthreads* для программирования современных суперкомпьютеров и создания параллельных программ, способных эффективно выполняться на будущих суперкомпьютерах с миллионами процессорных ядер. Исследование эффективности использования библиотеки

qthreads для модельных и реальных задач будет продолжено.

Хотя введение в модель суперкомпьютерного программирования легких тредов с контролируемой пользователем привязкой тредов к сегментам разделяемой памяти требует от программиста дополнительных усилий, но легкие треды необходимы для эффективной работы памяти, и пока нельзя рассчитывать на их автоматическое формирование компилятором.

Статья рекомендована к публикации программным комитетом международной суперкомпьютерной конференции «Научный сервис в сети Интернет: экзафлопсное будущее».

Литература

1. Kogge, P. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems / Peter Kogge et al. – University of Notre Dame. September 28, 2008. – URL: http://www.sc.doe.gov/ascr/Research/CS/DARPA%20exascale%20–%20hardware%20(2008).pdf (дата обращения: 04.04.2012).
2. Wheeler, K. Qthreads: An API for Programming with Millions of Lightweight Threads / K. Wheeler et al // Workshop on Multithreaded Architectures and Applications at IEEE IPDPS, April, 2008. – P. 1 – 8.
3. Корнеев, В.В. Следующее поколение суперкомпьютеров / В.В. Корнеев // Открытые системы. – 2008. – №8. – С. 14 – 19.
4. Корнеев, В.В. Подход к программированию суперкомпьютеров на базе многоядерных мультитредовых кристаллов / В.В. Корнеев // Вычислительные методы и программирование. – 2009. – Т. 10.– С. 123 – 128.
5. Wen, X. FPGA–Based Prototype of a PRAM–On–Chip Processor. / X. Wen, U. Vishkin. – CF'08, May 5–7, 2008, Ischia, Italy. – P. 55 – 66.
6. Qthreads (2012). – URL: www.cs.sandia.gov/qthreads (дата обращения: 2.03.2012).

Виктор Владимирович Корнеев, доктор техничеких наук, профессор, зам. директора ФГУП НИИ «Квант» по научной работе (г. Москва, Российская Федерация), korv@rdi-kvant.ru.

Александр Владимирович Будник, инженер ФГУП НИИ «Квант» (г. Москва, Российская Федерация), budnikjob@mail.ru.

MSC 68N19

Efficiency of Programming Model Based on Explicit Creating Light Threads

V.V. Korneev, Research and Development Institute «Kvant» (Moscow, Russian Federation)**,** ***A.V. Budnik,*** Research and Development Institute «Kvant» (Moscow, Russian Federation)

The results of research of efficiency of execution of parallel programs created on the basis of the different programming models are presented. Programs run on an SMP formed a motherboard with few multi-core crystals. One programming model is based on the implicit to the user allocation of generated parallel threads on the provided SMP hardware resources and traditional approaches to the synchronization of threads. This model is represented by Open MP. The second model uses the explicit allocation of lightweights threads on resources SMP with the use of developed at Sandia National Laboratory library Qthreads.

Keywords: scalable parallel programs, the effectiveness of parallel computing, lightweight threads.

References

1. Kogge P. *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems.* University of Notre Dame. September 28, 2008. Available at: http://www.sc.doe.gov/ascr/Research/CS/DARPA%20exascale%20–%20hardware%20(2008).pdf (accessed 4 April 2012).
2. Wheeler K. Qthreads: An API for Programming with Millions of Lightweight Threads. *Workshop on Multithreaded Architectures and Applications at IEEE IPDPS*, April, 2008, pp. 1 – 8.
3. Korneev V.V. Next Supercomputers Generation [Sleduyushchee pokolenie superkomputerov]. *Otkrytye sistemy* [Open systems], 2008, no. 8, pp. 14 – 19.
4. Korneev V.V. An Approach to the Programming of Supercomputers on the Basis of Multicore Multithreaded Chips [Podkhod k programmirovaniu superkomputerov na base mnogoyadernykh multitredovykh kristallov]. *Vychislitel'nye metody i programmirovanie* [Numerical Methods and Programming], 2009, vol. 10, pp. 123 – 128.
5. Wen X., Vishkin U. FPGA–Based Prototype of a PRAM–On–Chip Processor. *CF'08, May 5–7*, 2008, Ischia, Italy, pp. 55 – 66.
6. Qthreads (2012). Available at http://www.cs.sandia.gov/qthreads (accessed 2 March 2012).

Поступила в редакцию 6 декабря 2011 г.