МОДЕЛИРОВАНИЕ И ИССЛЕДОВАНИЕ АРХИТЕКТУРЫ МНОГОПРОЦЕССОРНОЙ СИСТЕМЫ С ИСПОЛЬЗОВАНИЕМ РАСПРЕДЕЛЁННОЙ МОДЕЛИРУЮЩЕЙ СРЕДЫ GRAPHITE

Г.С. Речистов, В.М. Пентковский, А.А. Иванов, П.Л. Шишпор

Многопроцессорные вычислительные комплексы находят широкое применение практически во всех областях современного научного исследования. Без использования достаточных вычислительный мощностей некоторые расчёты могут занять неприемлемо большое время. Однако один лишь факт использования ЭВМ с большим числом вычислительных узлов не способен дать желаемого результата, и при неправильном выборе конфигурации вычислительного комплекса дорогостоящее оборудование лишь будет напрасно тратить электроэнергию и время вместо того, чтобы эффективно решать поставленные задачи.

Очевидно, что не существует универсального рецепта построения многопроцессорной системы. Для каждого класса вычислительных задач решение должно приниматься с учётом используемых алгоритмов, объёма обрабатываемых данных и т.п. Поэтому является актуальной задача предсказания эффективности предложенной архитектуры многопроцессорного вычислительного комплекса на выбранном классе вычислительных задач. Перспективным представляется подход с компьютерной симуляцией новой ЭВМ ещё до её постройки. Подобная практика хорошо зарекомендовала себя в области т.н. pre-Silicon development [1, 2].

Дополнительное преимущество симуляционного подхода — возможность оценить производительность таких ЭВМ, реализация которых «в железе» является затруднительной по ряду причин (напр. такая машина будет слишком дорога, или ещё не существует компонент, из которых её можно было бы построить).

Примером подобных исследований могут служить работы по построению моделей систем с миграцией исполняемого кода на узел с данными, подлежащими обработке (архитектура EM2 [3]), а также исследования компьютеров с транзакционной памятью [4, 5].

В качестве класса задач, для которых будет производиться исследование архитектуры, были выбраны приложения моделирования молекулярной динамики, используемые в биологии для изучения взаимодействия молекул лекарств с клеточными мембранами. В подобных задачах численно моделируется взаимодействие большого числа атомов и молекул, происходящее на протяжении большого числа шагов. Существует несколько математических пакетов для проведения подобных исследований, например GROMACS [6], которые являются сложными программами, и поэтому было решено в первую очередь провести исследование на приложении достаточно простом и в то же время хорошо масштабируемом на многопроцессорных системах.

В качестве такого приложения был взят тестовый пакет High Performance LINPACK [7], который широко используется для оценки производительности вычислительных комплесков, в частности, в составлении списка Top 500 суперькомпьютеров. Нас будет интересовать его модификация, адаптированная для гибридных систем (т.е. одновременно с общей памятью некоторых процессоров и с соединением узлов) компанией Intel [8].

Данная работа описывает построение модели многопроцессорной ЭВМ на основе Graphite [9] — параллельной распределённой системы симуляции. Данная система предназначена для моделирования многоядерных процессоров, имеющих вплоть до нескольких тысяч ядер. Graphite позволяет запускать многопоточные приложения, использующие для распараллеливания библиотеку POSIX threads под управлением ОС Linux, при этом требуется минимальное изменение их исходного кода (добавление двух вызовов функций в начале и в конце функции main() для обозначения интересующего нас региона программы) и процесса сборки (необходимо добавить библиотеки поддержки Graphite). Работа распределяется среди многих машин, соединённых по сети, таким образом увеличивается скорость симуляции. Вместе с тем для исследуемого приложения сохраняется иллюзия того, что оно работает на одном компьютере как один процесс с единым адресным пространством.

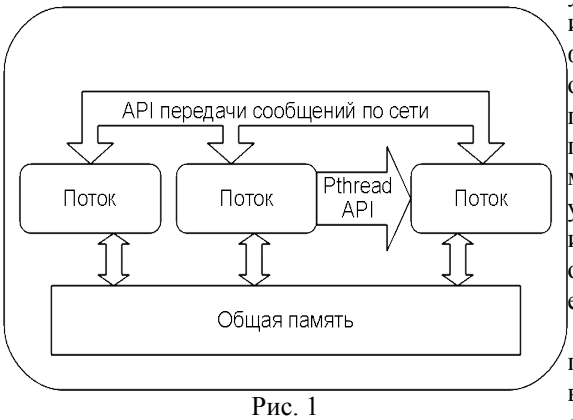

Рис. 1

Разработанный для моделирования многоядерных процессоров, симулятор Graphite предоставляет возможность моделировать разные части исследуемой архитектуры: вычислительный конвейер, подсистему

памяти (кэши и динамическую память), сеть, соединяющую вычислительные ядра. По завершении работы приложения симулятор создаёт файл статистики, в котором содержится информация от этих моделей: число выполненных инструкций, эффективность работы кэшей, количество пакетов, прошедших по сетям, соединяющей вычислительные ядра, и т.п.

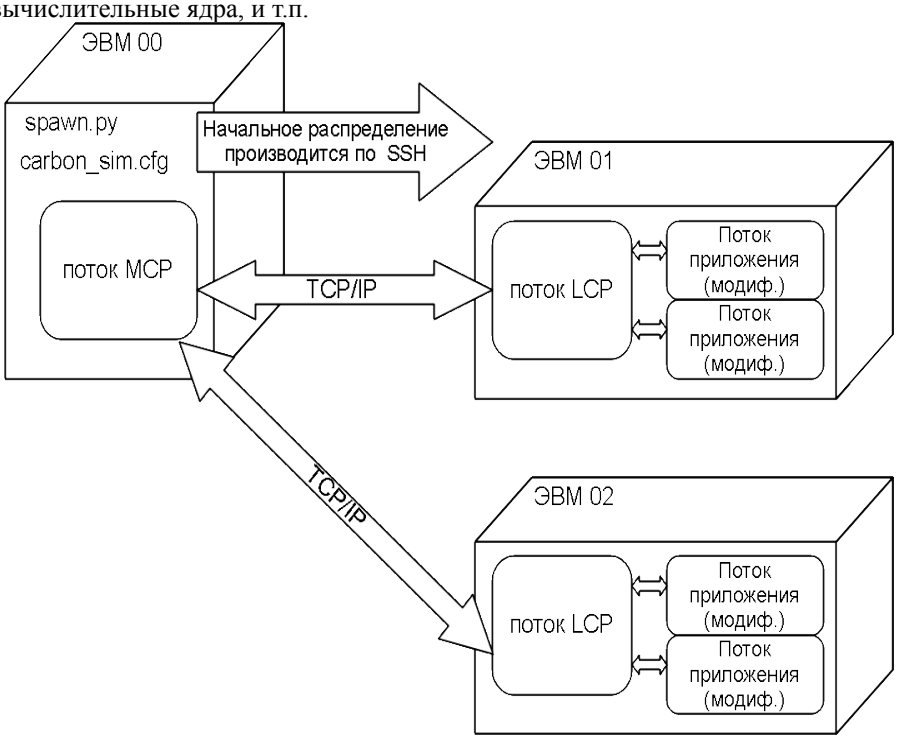

Рис. 2

На первом рисунке схематически изображено многопоточное приложение; его части могут взаимодействовать через общую память и/или через механизм сообщений по сети, а также через вызовы библиотеки Pthread (создание/уничтожение потоков и т.п.). На втором рисунке показано, как потоки исходного приложения распределяются симулятором по нескольким машинам, соединённым по сети; здесь MCP (master control program) и LCP (local control program) — специальные потоки, созданные и выделенные под нужды самого симулятора, **carbon_sim.cfg** — файл конфигурации, описывающий работу моделей и содержащий список компьютеров, на которые будет распределено выполнение, **spawn.py** — скрипт, призванный автоматизировать процесс запуска программ на удалённых машинах по протоколу SSH. В процессе дальнейшей работы все коммуникации осуществляются уже по незащищённым каналам с адресами портов, начинающимися с 2000.

Ещё раз отметим, что Graphite не предоставляет виртуализации на уровне операционной системы, лишь на уровне одного приложения. Для научных задач этого, как правило, достаточно, так как в большинстве случаев они не требуют знаний о системном уровне ЭВМ, на которой они выполняются.

При проектировании модели ЭВМ, которая будет моделироваться под управлением Graphite, имеется возможность варьировать различные характеристики модели, касающиеся как системы межпроцессорных соединений, так и функций отдельного вычислительного ядра.

Для системы в целом мыслимы следующие параметризации:

1. Варьировать полное число ядер в системе.
2. Перераспределять ядра между уровнями блоков с общей памятью и группами ядер с независимой памятью, соединённых в сеть.
3. Изменять топологию системы, т.е. определять, коммуникация между какими узлами возможна и какова её стоимость в терминах производительности: пропускная способность (throughput), задержка (latency).

Следующие возможности усматриваются в конфигурации одиночного ядра:

1. Изменять количество и параметры частных кэшей ядра.
2. Изменять архитектуру ядра. Здесь возможности Graphite существенно ограничены, так как он не предоставляет средств к моделированию инструкций, не присутствующих в наборе команд используемого процессора. Фактически они сводятся к изменению ширины набора векторных инструкций SIMD, использованных при сборке приложения, и таким образом зависят от компилятора. Следует отметить, что этот вопрос требует отдельного тщательного рассмотрения.

Предлагается следующая методика для учёта всех видов взаимодействия вычислительного ядра с

иерархией памяти и сетевой подсистемой. Каждый доступ по адресу в память регистрируется в модели системы кэшей/общей памяти. С функциональной точки зрения отличие оперативной памяти от кэш-памяти состоит лишь в том, что чтение/запись в неё видны в соседних ядрах; с точки зрения потактовой модели разница состоит в увеличении времени доступа для более удалённых от ядра уровней памяти. При этом доступ достигает одного или нескольких уровней этой системы, в зависимости от характеристик модели. При этом определяется суммарное время доступа. Похожим образом, при вызове API модели сети соответствующая модель ответственна за маршрутизацию доступа, чтение/запись данных и регистрацию полной задержки, вызванной этим доступом. Иерархичность сетевой системы не показана, но она также может присутствовать.

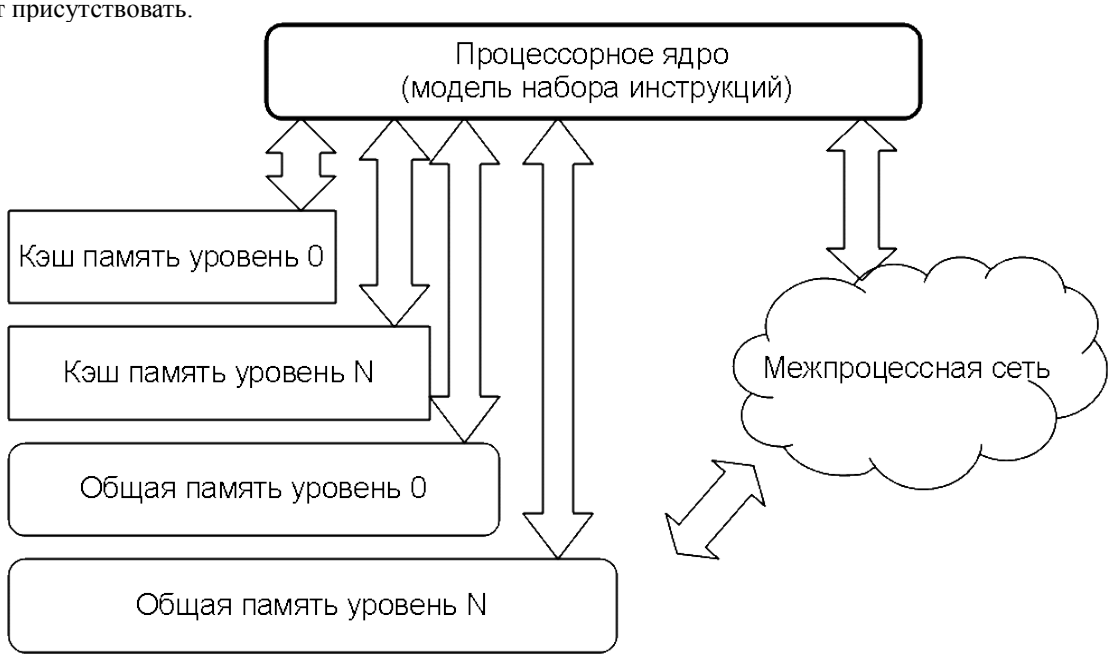

Рис. 3

В настоящее время в симуляторе уже присутствуют модели кэшей и сетей. Потребуется некоторая их модификация для того, чтобы получить более сложные модели, соответствующие реальному оборудованию; в первую очередь это касается топологий межпроцессорных соединений.

Суммируя всё вышеописанное, можно сказать, что наша задача состоит в измерении характеристик производительности для некоторого приложения (в нашем случае — LINPACK) на различных конфигурациях модели и нахождение оптимальной конфигурации, при которой есть основания считать, что время исполнения на реальной аппаратуре будет наиболее быстрым. Для её решения необходимо ввести метрики, позволяющие сравнивать конфигурации между собой. Фактически, единственная ключевая характеристика построенной системы — это симулированная длительность исполнения программы. Не менее важной является профильная информация, собранная для отдельных подсистем, например, максимальная и средняя загруженность вычислительных ядер, пропускная способность сетевых соединений. Знания этих чисел позволяют определять «узкие места» и направленно оптимизировать систему.

В процессе работы по адаптации LINPACK нами было обнаружено, что в настоящее время у Graphite существует несколько ограничений и нереализованных функций. Они описаны ниже вместе с предполагаемыми путями их решения.

1. Не все системные вызовы Linux поддерживаются, реализованы лишь те, что были необходимы авторам Graphite для выполнения определённого набора программ. Например, не поддерживается вызов socket(), необходимый для инициализации процесса коммуникации приложений через UNIX-сокеты.
2. Важный частный случай описанной выше проблемы составляют неподдерживаемые вызовы fork() и exec(). Поэтому запуск приложений, использующих во время инициализации инфраструктуру MPI (например, программу mpiexec), затруднён, так как требует одновременной работы многих процессов. К таким приложениям относится оригинальный тест HPL [7], именно поэтому мы основываем нашу работу на варианте Intel.
3. Двоичный транслятор не всегда надёжно обрабатывает машинные инструкции. Так, было обнаружено, что выполнение теста аварийно прерывается на ранней стадии при сборке программы с помощью компилятора Intel на инструкции pushfq. При этом сборка с помощью компилятора GCC не подвержена этой проблеме.
4. Для проверки степени «прозрачности» симулятора для исследуемого приложения был написан простой тест, суть которого заключалась в создании нескольких потоков, каждый из

которых затем распечатывал имя ЭВМ (доступное через функцию gethostname()), на которой он выполняется. В результате в выводе программы были получены несколько имён машин, по которым в действительности было распределено исполнение, тогда как при полностью прозрачной работе ожидается, что все потоки будут получать одно и то же значение имени компьютера, как это происходит при работе на единственной физической системе.

Нами предлагается несколько подходов к решению описанных выше проблем.

1. Неподдерживаемую в настоящее время работу с UNIX-сокетами необходимо будет реализовать и при этом сопрячь с моделью сети, чтобы правильным образом моделировать трафик, через неё идущий.
2. Затруднения, вызванные необходимостью адаптации MPI, можно решить, используя одну из доступных реализаций этой библиотеки, основанных на разделяемой памяти [10, 11, 12], а не на сетевом взаимодействии. Тем самым мы избегаем выхода симуляции из рамок одного процесса.
3. Анализ исходных кодов LINPACK показал, что программа использует сравнительно небольшое количество функций библиотеки MPI — всего 19. Представляется возможным написать собственные варианты этих подпрограмм, при этом учитывающих специфику изучаемой нами проблемы.

В результате симулятор Graphite является достаточно перспективным инструментом для задач исследований архитектуры многопроцессорных систем. В настоящее время, как активно развивающийся проект, он не лишён ряда недостатков, которые представляются нам исправимыми. Рассмотренный в данной работе подход позволяет достигнуть нескольких важных результатов при проектировании таких машин.

1. Оценить производительность ЭВМ для различных её конфигураций с целью поиска наилучшей.
2. Изучить поведение многопроцессорной системы на интересующем классе реальных приложений через симуляцию их выполнения.
3. Ускорить процесс моделирования с помощью распределения симулятора на несколько компьютеров, прозрачного для изучаемого приложения.

Работа выполнена в рамках гранта, выделенного в соответствии с постановлением Правительства России №220 от 9.04.2010 г.

ЛИТЕРАТУРА:

1. SoftSDV: A presilicon software development environment for the IA-64 architecture/ R. Uhlig, R. Fishtein, O. Gershon et al. // Intel Technology Journal.1999. Pp. 112–126.
2. Simics: A full system simulation platform / P. S. Magnusson, M. Christensson, J. Eskilson et al. // Computer.— 2002.—February.— Vol. 35.— Pp. 50–58. http://portal.acm.org/citation.cfm?id=619072.621909.
3. Khan, O. EM2: A scalable shared-memory multicore architecture: Tech. rep. /O. Khan, M. Liz, S. Devadas: Massachusetts Institute of Technology, 2010.—June.
4. Rajwar, R. Speculation-based techniques for transactional lock-free execution of lock-based programs: Ph.D. thesis / University of Wisconsin – Madison.— 2002.
5. LogTM-SE: Decoupling hardware transactional memory from caches / L. Yen, J. Bobba, M. R. Marty et al. // The 13th Annual International Symposium on High Performance Computer Architecture (HPCA-13).— 2007.—February.
6. GROMACS: Fast, flexible, and free / D. Van Der Spoel, E. Lindahl, B. Hess et al. // Journal of Computational Chemistry.— 2005.— Vol. 26, no. 16.— Pp. 1701–1718.
7. High Performance LINPACK benchmark. http://www.netlib.org/linpack/.
8. Intel LINPACK benchmarks. http://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download/.
9. Graphite: a distributed parallel simulator for multicores / J. E. Miller, H. Kasture, G. Kurian et al. // The 16th IEEE International Symposium on High-Performance Computer Architecture (HPCA).– 2010.–January.
10. Diaz Martin, J. AzequiaMPI – a thread-based implementation of the message passing interface (MPI-1) standard.
11. Demaine, E. A threads-only MPI implementation for the development of parallel programs / E. Demaine.– 1997. – July. – Pp. 153–163. http://erikdemaine.org/software/TOMPI/.
12. Tang, H. Optimizing threaded MPI execution on SMP clusters / H. Tang, T. Yang.–2001.– Pp. 381–392. http://www.cs.ucsb.edu/projects/tmpi/.