важную роль. Учёт влияния микроструктуры на поведение макрообъекта влечёт за собой необходимость использования численных методов и средств, реализующихся на вычислительных кластерах. В качестве примера в данной работе рассматривается модель билиарной системы (рис. 2), включающей в себя такие элементы, как печень, желчный пузырь, желчные протоки и поджелудочная железа. Рассматривается актуальность изучения её элементов, начиная со сложной микроструктуры с переходом в дальнейшем на более высокие уровни. Постулируется класс задач [связанные задачи о взаимодействии жидкости и твердого тела (fluid–solid interaction), контактные задачи и т.д.] и методы, которые необходимо использовать при моделировании поведения данной системы в норме и при патологиях.

А.А. Лабутина

Нижегородский государственный университет им. Н.И. Лобачевского

ОПТИМИЗАЦИЯ ТЕСТОВ HPC CHALLENGE BENCHMARK SUITE ДЛЯ ГЕТЕРОГЕННЫХ КЛАСТЕРОВ

В этом году студенческая команда Нижегородского государственного университета им. Н.И. Лобачевского примет участие в соревновании Student Cluster Competition, которое будет проходить в рамках международной конференции Supercomputing–2010 в Новом Орлеане, США. Команда ННГУ выступает при поддержке компаний Microsoft, IBM и NVidia.

При подготовке к соревнованию каждая команда в сотрудничестве с консультантами и спонсорами разрабатывает архитектуру своего передового высокопроизводительного вычислительного кластера. Кластер команды ННГУ предоставлен компанией IBM, крупнейшим в мире производителем серверов. Кластер состоит из шести серверов IBM iDataPlex dx360 M3 с центральными 6-ядерными процессорами Intel Xeon серии 5600, укомплектованных двумя вычислительными модулями Nvidia Tesla M2050. Модуль общего назначения Tesla M2050, включающий 448 процес-

сорных ядер и оснащенный 3 Гб выделенной памяти GDDR5, предлагает скорость работы до 515 GFlops для вычислений с двойной точностью (1030 GFlops для вычислений с одинарной точностью). По оценке IBM, по сравнению с серверами предыдущих поколений серверы iDataPlex dx360 M3 обеспечивают на 40 % улучшенное потребление энергии и до 5 раз увеличенную производительность.

Основная задача команды на соревновании – за отведенное время правильно выполнить максимальное число запусков заранее определенных параллельных приложений на тех входных данных, которые предоставляются организаторами. Для вычислительной системы действуют ограничения на максимальную потребляемую мощность.

В качестве предварительного этапа соревнования командам предстоит выполнить запуск набора тестов HPC Challenge Benchmark Suite (HPCC) и получить максимально возможный результат. Команде, которой удастся добиться наибольшей производительности, начисляются дополнительные очки.

- HPCC представляет собой средство оценки и исследования производительности высокопроизводительных систем, реализующее более сложные модели доступа к оперативной памяти, нежели тест High Performance Linpack. В состав HPCC входят тесты *HPL*, *DGEMM*, *STREAM*, *PTRANS*, *Random Access*, *FFT* и *Communication bandwidth and latency*.
- *HPL* оценивает производительность при решении системы линейных уравнений.
- *DGEMM* оценивает производительность при выполнении матричного умножения.
- *STREAM* оценивает пропускную способность оперативной памяти на основе четырёх простейших векторных операций с векторами большой длины (COPY, SCALE, ADD, TRIAD).
- *PTRANS* оценивает пропускную способность сети и памяти на задаче параллельного транспонирования матриц.
- *Random Access* оценивает скорость случайного доступа к оперативной памяти.

• *FFT* оценивает производительность при выполнении одномерного быстрого преобразования Фурье.

• *Communication bandwidth and latency* оценивает пропускную способность сети и величину латентности в трёх моделях коммуникации между процессами – ping-pong, ring, random ring.

Многие из этих тестов представляют собой уже давно зарекомендовавшие себя вычислительные ядра (computation kernels), распространённые в среде специалистов по высокопроизводительным вычислениям. Однако с целью более точного и тонкого тестирования авторами были внесены коррективы, проводящие верификацию данных и вывод извлечённых характеристик в наглядной форме.

Таким образом, чтобы увеличить свои шансы на победу, команде ННГУ необходимо произвести перенос части вычислений, которые выполняются в тестах HPCC, на GPU.

В данной статье мы представим подход, который используется для увеличения производительности теста Linpack при помощи технологии CUDA. В состав пакета для разработчика CUDA Toolkit входят профилировщик, отладчик и широко используемые библиотеки для высокопроизводительных вычислений:

• библиотека CUBLAS – реализация библиотеки операций линейной алгебры BLAS;

• библиотека CUFFT – библиотека функций для выполнения быстрого преобразования Фурье.

Подход, представленный в данной статье, предполагает использование библиотеки CUBLAS и не требует программирования дополнительных CUDA-функций.

Тест Linpack решает случайно заданную систему линейных уравнений методом LU-разложения с выбором ведущего элемента столбца, используя 64-битную арифметику с плавающей запятой, и позволяет судить о производительности системы при решении такой задачи. Наибольшее время при выполнении теста Linpack затрачивается на выполнение операции матричного умножения (DGEMM), причем с ростом размерности матрицы время, необходимое на выполнение данной операции, также

растет. Таким образом, оптимизация матричного умножения – основной шаг к увеличению производительности теста Linpack.

Матричное умножение – одна из основных операций при реализации многих алгоритмов. Интерфейс BLAS предусматривает наличие нескольких функций для выполнения этой операции: для случая, если умножение выполняется над матрицами с элементами типа double, предусмотрена функция DGEMM.

Если *A[M, K]*, *B[K, N]* и *C[M, N]* – входные данные, то вызов функции DGEMM выполнит операцию $C = \alpha AB + \beta C$. Интерфейс BLAS предусматривает возможность задания размеров ведущих измерений (*leading dimension*) матриц *A*, *B* и *C* (*lda*, *ldb* и *ldc*) отличными от *M*, *K* и *N*. Для выполнения представленной задачи умножения матриц необходимо вызвать функцию DGEMM со следующими аргументами:

DGEMM('N', 'N', M, N, K, alpha, A, lda, B, ldb, beta, C, ldc).

Выполнение матричного умножения – одна из немногих задач, где процессор может продемонстрировать производительность, близкую к пиковой. Для функции DGEMM существует несколько высокопроизводительных реализаций, например, реализация в рамках библиотеки GotoBLAS, разработанной в техасском университете, а также реализации от производителей процессоров, такие как Intel MKL и AMD ACML.

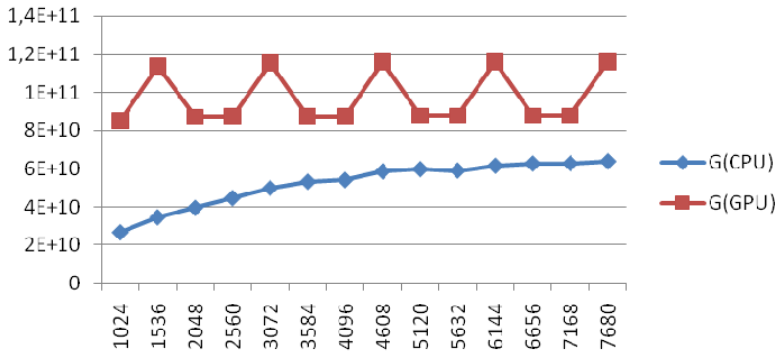

Рис. 1. Выполнение операции DGEMM на CPU и GPU

На тестовом сервере с двумя процессорами Intel Xeon серии 5600 и двумя графическими модулями nVidia Tesla M2050 были проведены эксперименты для определения производительности CPU и GPU при выполнении функции DGEMM. Для выполнения умножения матриц на центральном процессоре использовалась функция cbas_Dgemm, реализованная в библиотеке IntelMKL. Для выполнения умножения матриц на графическом процессоре использовалась реализация из библиотеки CUBLAS. Результаты экспериментов представлены на рис. 1.

Идея, которая использована в данной работе для ускорения теста Linpack, очень проста: для того чтобы ускорить выполнение операции матричного умножения, исходные матрицы C и B представляются как объединение двух подматриц $C=C_1UC_2$ и $B=B_1UB_2$. В этом случае операция DGEMM

$$C = \alpha AB + \beta C$$

может быть представлена следующим образом:

$$C = \alpha(AB_1+AB_2)+\beta(C_1+C_2),$$

а вызов функции разделяется на два последовательных вызова (рис. 2):

DGEMM('N', 'N', M, N_1, K, alpha, A, lda, B_1, ldb, beta, C_1, ldc)
DGEMM('N', 'N', M, N_2, K, alpha, A, lda, B_2, ldb, beta, C_2, ldc).

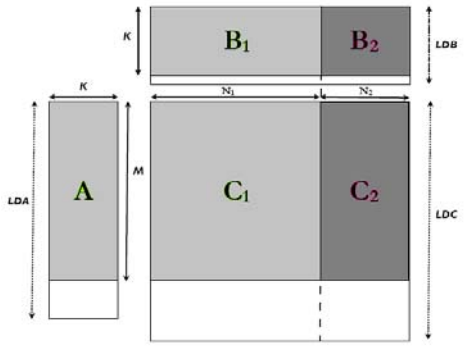

Рис. 2. Разделение вычислительной нагрузки между CPU и GPU при выполнении операции умножения матриц

Поскольку эти две операции независимы, мы можем выполнить первую на GPU а вторую – на CPU. Как только необходимые данные переданы в память графического процессора, мы можем параллельно выполнять операции DGEMM, как это показано на рис. 3. Кроме того, следует отметить, что при выполнении разделения вычислительной нагрузки можно подобрать размеры частей матрицы таким образом, чтобы производительность графического процессора достигала наибольшего возможного значения.

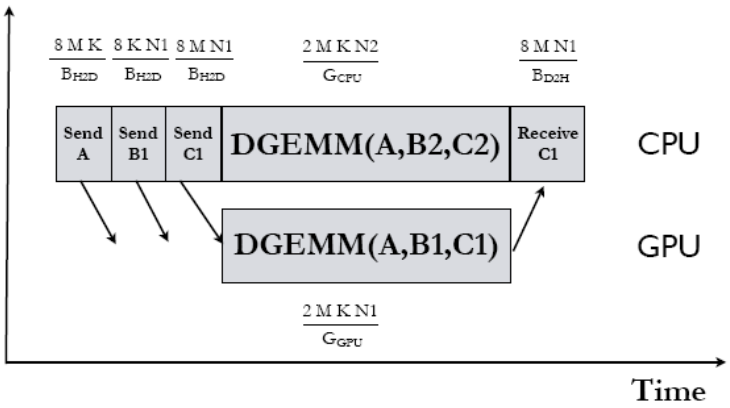

Рис. 3. Схема параллельного выполнения операции матричного умножения на CPU и GPU

Используя специализированные функции библиотеки CUBLAS для перемещения данных между оперативной памятью и памятью графического процессора, реализовать подобный подход не представляет труда:

```
// Copy A from CPU memory to GPU memory devA
status = cublasSetMatrix (m, k , sizeof(A[0]), A,
lda, devA, m_gpu);
// Copy B1 from CPU memory to GPU memory devB
status = cublasSetMatrix (k ,n_gpu, sizeof(B[0]),
B, ldb, devB, k_gpu);
// Copy C1 from CPU memory to GPU memory devC
```

```
status = cublasSetMatrix (m, n_gpu, sizeof(C[0]),
C, ldc, devC, m_gpu);
// Perform DGEMM(devA,devB,devC) on GPU
// Control immediately return to CPU
cublasDgemm('n', 'n', m, n_gpu, k, alpha, devA,
m,devB, k, beta, devC, m);
// Perform DGEMM(A,B2,C2) on CPU
dgemm_cpu('n','n',m,n_cpu,k,        alpha,        A,
lda,B+ldb*n_gpu, ldb,
  beta,C+ldc*n_gpu, ldc);
// Copy devC from GPU memory to CPU memory C1
status = cublasGetMatrix (m, n, sizeof(C[0]), devC,
m, C, *ldc);
```

Для того чтобы максимизировать производительность, мы должны определить наилучшее разделение объема вычислений между CPU и GPU. Необходимо учитывать, что дополнительное время будет затрачено на перемещение данных между памятью центрального процессора и графического процессора. Требуется измерить значения следующих величин:

- B_{H2D} – величина пропускной способности канала при передаче данных из оперативной памяти в память графического процессора (GB/s);
- G_{GPU} – производительность операции DGEMM на графическом процессоре (GFlops);
- G_{CPU} – производительность операции DGEMM на центральном процессоре (GFlops);
- B_{D2H} – величина пропускной способности канала при передаче данных из памяти графического процессора в оперативную память (GB/s);

Вызов функции DGEMM на CPU выполняет 2KMN операций, и, следовательно, если центральный процессор может выполнять эти операции со скоростью GCPU, то время, необходимое для вычислений

$$T_{CPU}(M,K,N) = 2\frac{MKN}{G_{CPU}}.$$

Общее время, необходимое на выполнение части операции матричного умножения на GPU, состоит из времени ввo-

да/вывода, которое тратится на передачу данных в память графического процессора и обратно, и времени вычислений:

$$T_{GPU}(M,K,N) = 8\frac{(MK + KN + MN)}{B_{H2D}} + 2\frac{MKN}{G_{GPU}} + 8\frac{MN}{B_{D2H}}.$$

Оптимальное разделение вычислительной нагрузки между CPU и GPU достигается в случае

$$T_{CPU}(M,K,N) = T_{GPU}(M,K,N_1),$$

где

$$N = N_1 + N_2.$$

Для упрощения полученных оценок величины оптимального разделения $\mu = N_1/N$ мы можем принебречь временем передачи данных $O(N^2)$ в сравнении со временем вычислений $O(N^3)$:

$$\mu = \frac{G_{GPU}}{G_{GPU} + G_{CPU}}.$$

Для тестовой системы, где центральный шестиядерный процессор Intel Xeon имеет производительность на операции DGEMM 60 GFlops, а графический процессор – 115 GFlops, данная формула позволяет определить оптимальное разделение $\mu = 0{,}65$.

Подобный механизм разделения вычислений необходимо реализовать также для выполнения операции DTRSM.

Одной из важных характеристик представленного подхода является то, что для его реализации не нужно вносить каких-либо изменений в исходный код тестов HPCC. Достаточно реализовать библиотеку-обертку, которая перехватывает вызов функции DGEMM и DTRSM и выполняет распределение вычислений между CPU и GPU.

Выполнение некоторых других тестов из набора HPCC также может быть оптимизировано с использованием технологии CUDA. Планируется разработать оптимизированную версию тестов DGEMM и STREAM.

Список литературы

1. http://www.top500.org

2. HPL – a portable implementation of the high performance Linpack benchmark for distributed memory computers, version 2.0 / Petitet A. [et al.]. – URL: http://www.netlib.org/benchmark/hpl.

3. NVIDIA CUDA Compute Unified Device Architecture Programming Guide.

4. The HPC Challenge (HPCC) Benchmark Suite / P. Luszczek [et al.]. – URL: http://icl.cs.utk.edu/projectsfiles/hpcc/pubs/sc06_hpcc.pdf.

5. Fatica M. Accelerating Linpack with CUDA on heterogenous clusters // ACM International Conference Proceeding Series. Vol. 383.

6. Dongarra J.J. Performance of various computers using standard linear equations software. Technical report, 2008. – URL: http://www.netlib.org/benchmark/performance.pdf.

А.А. Лабутина, А.В. Линев

Нижегородский государственный университет им. Н.И. Лобачевского

ОПТИМИЗАЦИЯ ПРИКЛАДНЫХ ПАКЕТОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КЛАСТЕРА КОМПОЗИТНОЙ АРХИТЕКТУРЫ В РАМКАХ ПОДГОТОВКИ К СТУДЕНЧЕСКОМУ СОРЕВНОВАНИЮ STUDENT CLUSTER COMPETITION–2010

В ноябре 2010 года будет проходить SC10 (SuperComputing–2010) – международная конференция, посвященная высокопроизводительным вычислениям, сетям, системам хранения и исследованиям [1]. Одним из пунктов программы конференции является Student Cluster Competition (SSC) – соревнование, в котором участвуют команды, состоящие из шести студентов, проходящее в режиме реального времени на территории выставки [2]. В ходе состязания команды должны продемонстрировать использование для решения предоставленного организаторами набора задач ряда распространенных прикладных пакетов с открытым кодом, установ-