ИССЛЕДОВАНИЕ ВЫСОКОПРОИЗВОДИТЕЛЬНОГО РЕШЕНИЯ ЗАДАЧИ *N* ТЕЛ НА БАЗЕ ПЛАТФОРМЫ OPENCL

М.М. Захаров1, Д.К. Боголепов2, О.Д. Блохин2, Д.П. Сопин2

1*Нижегородский государственный технический университет им. Алексеева*

2*Нижегородский государственный университет им. Лобачевского*

# Введение

OpenCL является *первым* открытым межплатформенным стандартом для параллельных вычислений на современных процессорах, включая многоядерные центральные процессоры и графические ускорители. По мнению разработчиков стандарта, обладая низкоуровневым (“close to the metal”), высокопроизводительным и переносимым интерфейсом программирования, OpenCL должен сформировать фундаментальный слой в экосистеме параллельных вычислений. В настоящий момент стандарт реализован в продуктах таких компаний, как ATI/AMD, Apple, NVIDIA, IBM и др.

# Winter School 2010-04-06 14-04-04-82.bmp Winter School 2010-04-06 14-04-40-15.bmp Winter School 2010-04-06 14-03-30-00.bmp

**Рис. 1**. Пример моделирования системы из *N* = 214 тел

Целью данной работы являлось исследование возможности программирования аппаратуры с помощью OpenCL и оценка производительности в сравнении с другими решениями: шейдерные языки, технология CUDA от компании NVIDIA для графических процессоров и OpenMP – для ЦПУ. В качестве тестовой была выбрана задача моделирования динамики *N* точечных зарядов, помещенных в магнитное поле, при этом рассчитывались взаимодействия между *всеми* *парами* зарядов. Основными физическими моделями для задачи является закон Кулона [1] и сила Лоренца [2]. Для интегрирования полученной системы дифференциальных уравнений использовался метод Эйлера, который является стандратным для тестовых программ такого рода.

# Теоретическая оценка производительности

Перед анализом фактической производительности полезно дать некоторые теоретические оценки. Для этого следует выделить основной вычислительный код и оценить число *тактов* и *элементарных операций*, которые затрачиваются на его выполнение. Данные величины в общем случае *различны*: число тактов зависит от архитектуры, в то время как число элементарных операций является характеристикой алгоритма. В рассматриваемой задаче основной объем вычислений связан с расчетом *ускорения* частицы под воздействием остальных частиц системы. Далее приводится псевдокод соответствующей функции:

|  |  |  |  |
| --- | --- | --- | --- |
| *Функция**расчета ускорения для одной частицы*  *p и q – положение и заряд текущей частицы* | *Элементарных операций* | *Циклов ГПУ* | *Циклов ЦПУ* |
| *1* | *for ( int i = 0; i < N; ++i )* | *-* | *-* | *-* |
| *2* | *{* | *-* | *-* | *-* |
| *3* | *float3 r = p – p [i];* | *3* | *3* | *1* |
| *4* | *float dist = dot ( r, r );* | *5* | *3* | *2* |
| *5* | *float invDist = inversesqrt ( dist + SOFTENING );* | *2* | *2* | *4* |
| *6* | *float invDistCube = invDist \* invDist \* invDist;* | *2* | *2* | *2* |
| *7* | *float s = q \* q [i] \* invDistCube;* | *2* | *2* | *2* |
| *8* | *acc += s \* r;* | *6* | *3* | *2* |
| *9* | *}* |  |  |  |
| *Итого* | *20* | *15* | *13* |

В данной работе функция вычисления обратного квадратного корня (строка *5*) оценивается в *одну* элементарную операцию и *один* такт для ГПУ. В большинстве работ принимаются другие оценки, учитывающие особенности конкретной архитектуры. Мы игнорируем эти различия и назначаем данной функции *минимальную* оценку, что позволит оценить производительность в GFLOPS *снизу* и число итераций в секунду *сверху*. Необходимо отметить, что современные ГПУ поддерживают инструкцию *mad*, выполняя сложение и умножение за *один* такт (строки *4* и *8*).

Поскольку тело цикла в функции расчета ускорения выполняется *N* раз, на обработку одной частицы затрачивается 20∙*N* элементарных операций, в то время как расчет одной итерации требует 20∙*N*2 элементарных операций.

В качестве примера, построим временную оценку одной итерации для графической карты NVIDIA Quadro FX 5600 1.5 Гб:



Данная оценка предполагает, что система из *N* = 214 = 16384 тел интегрируется методом Эйлера.

# Результаты экспериментов

Для проведения экспериментов на базе технологии CUDA и шейдерного языка OpenGL Shading Language (GLSL) использовались реализации [3] и [4] соответственно. Результаты замера производительности для *метода Эйлера* представлены в следующих таблицах.

1. Производительность в задаче *N* тел (ATI/AMD Radeon HD 4890 1 Гб, драйвер 10.3)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Число частиц* | *OpenCL* | | *GLSL* | |
| *FPS* | *GFLOPS* | *FPS* | *GFLOPS* |
| *2048* | *316* | *26.5* | *3109.0* | *260.71* |
| *4096* | *152.9* | *51.3* | *1617.0* | *542.57* |
| *8192* | *67.3* | *90.3* | *392.2* | *526.40* |
| *16384* | *16.9* | *90.7* | *106.9* | *573.91* |

1. Производительность в задаче *N* тел (NVIDIA Quadro FX 5600 1.5 Гб, драйвер 196.21)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Число частиц* | *OpenCL* | | *CUDA* | | *GLSL* | |
| *FPS* | *GFLOPS* | *FPS* | *GFLOPS* | *FPS* | *GFLOPS* |
| *2048* | *1147.7* | *96.27* | *1398* | *117.27* | *907.6* | *76.13* |
| *4096* | *583.8* | *195.89* | *672.19* | *225.55* | *319.9* | *107.34* |
| *8192* | *148.3* | *199.04* | *172.44* | *231.45* | *106.9* | *143.47* |
| *16384* | *37.3* | *200.25* | *43.39* | *232.98* | *27.8* | *149.25* |

Эксперименты на центральном процессоре были проведены для алгоритма, реализованного на C++ с использованием OpenMP. Приложение скомпилировано двумя компиляторами: MS VC++ 2008 и Intel® Parallel Composer из Intel® Parallel Studio 9.

1. Производительность в задаче *N* тел (Intel Core2Duo E8200)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Число частиц* | *OpenCL* | | *MS VC++ 2008* | | *Intel® Parallel Composer* | |
| *FPS* | *GFLOPS* | *FPS* | *GFLOPS* | *FPS* | *GFLOPS* |
| *2048* | 15.6 | *1.30* | *15.5* | *1.30* | *110.2* | *9.24* |
| *4096* | 3.8 | *1.27* | *4* | *1.34* | *27.6* | *9.26* |
| *8192* | 1.0 | *1.34* | *1.0* | *1.34* | *7.0* | *9.39* |
| *16384* | 0.25 | *1.34* | *0.25* | *1.34* | *1.75* | *9.39* |

# Заключение

Шейдеры обеспечивают минимальные возможности программирования в задачах общего назначения (в частности, не поддерживается *локальная* или *разделяемая* память), а платформы OpenCL и CUDA идентичны по своим возможностям. Поэтому от шейдерной реализации следовало бы ожидать наименьшей производительности, в то время как CUDA и OpenCL должны демонстрировать схожие результаты.

Данные соображения в целом реализуются для карты NVIDIA Quadro FX 5600 1.5 Гб. Незначительное отставание OpenCL можно объяснить недостаточной проработкой текущей реализации в сравнении с платформой CUDA. Интересно, что при использовании технологии CUDA достигается пиковая оценка производительности в *данной* задаче; при использовании OpenCL достигается 87% от пика.

Для карты ATI/AMD Radeon 4890 1 Гб полученная производительность находится на крайне низком уровне: в качестве базового ориентира можно рассматривать производительность шейдеров, которая находится на уровне 570 GFLOPS. Очевидно, что текущая реализация OpenCL от ATI/AMD не обеспечивает должного уровня производительности. Столь скромный результат объясняется еще и тем, что семейство ускорителей ATI/AMD Radeon 4800 *не поддерживает* локальную память на физическом уровне – она отображается на область глобальной памяти.

Важно заметить, что производительность решения на OpenCL приблизительно совпадает с производительностью OpenMP в связке с компилятором от MS, это может сыграть решающую роль, когда одно и то же приложение необходимо запустить на разных устройствах, отказавшись от привязки к операционной системе и архитектуре.

Платформа OpenCL продолжает активно развиваться и вполне может претендовать на роль стандарта для параллельных вычислений на гетерогенных системах. Выигрыш в производительности от использования графической карты вместо центрального процессора для решения подобных задач выглядит колоссальным. Мы столкнулись с рядом проблем текущих реализаций (например, реализация NVIDIA не поддерживает исполнение на центральном процессоре, а платформа ATI/AMD не поддерживает текстуры), однако рекомендовать OpenCL к использованию и получать приемлемые результаты можно уже сейчас.

# Литература

1. Закон Кулона (Материал из Википедии — свободной энциклопедии).

[*http://ru.wikipedia.org/wiki/Закон\_Кулона*](http://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%9A%D1%83%D0%BB%D0%BE%D0%BD%D0%B0)

1. Сила Лоренца (Материал из Википедии — свободной энциклопедии).

[*http://ru.wikipedia.org/wiki/Сила\_Лоренца*](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BB%D0%B0_%D0%9B%D0%BE%D1%80%D0%B5%D0%BD%D1%86%D0%B0)

1. Erich Elsen V., Vishal Mike Houston и др. N-Body Simulations on GPUs.

[*http://arxiv.org/pdf/0706.3060*](http://arxiv.org/pdf/0706.3060)

1. Боголепов Д.К., Турлапов В.Е. Вычисления общего назначения на графических процессорах с использованием шейдерных языков // Труды международной научной конференции “Параллельные вычислительные технологии”, Нижний Новгород, 30 марта – 3 апреля 2009 г., с. 339-410.

[*http://omega.sp.susu.ac.ru/books/conference/PaVT2009/papers/short\_papers/012.pdf*](http://omega.sp.susu.ac.ru/books/conference/PaVT2009/papers/short_papers/012.pdf)