Журнал обчислювальної та прикладної математики **2013, №2(112)** Journal of Computational & Applied Mathematics

УДК 004

МЕТОД ОПТИМИЗАЦИИ ВЫЧИСЛЕНИЙ С ИСПОЛЬЗОВАНИЕМ GPU В ГЕТЕРОГЕННЫХ КЛАСТЕРАХ И В ГРИД-СЕТИ

А. Н. Лавренюк, Н. С. Лавренюк

Резюме. Рассмотрен вопрос оптимального выбора GPU вычислителей для выполнения ядер-программ на OpenCL при их запуске на GPU в кластерах гетерогенного типа и в грид-сети, грид-узлами которой являются кластеры на GPU. Предложен метод оптимального выбора GPU вычислителей для минимизации времени выполнения программ с использованием GPU. В предложенном подходе учитываются определенные отличия работы GPU разных производителей и различной архитектуры. Предложенный метод будет эффективным также при использовании его на персональных компьютерах или ноутбуках с несколькими GPU.

Введение

Продолжается широкое использование ускорителей, особенно графических (GPU), не только при построении больших и малых кластеров, но уже и персональных компьютеров. Много таких кластеров доступны в грид-сети как грид-узлы [1], [2]. В последнее время кластеры с GPU стали доступными и для отечественных пользователей через традиционный доступ либо как узлы Украинского Национального грида [3]. Очевидно, что при решении конкретной задачи для получения оптимальной производительности необходимо задействовать одновременно несколько GPU, доступных на одном кластере либо грид-узле [4]. Но не всегда такой подход может дать положительный результат. Особенно, когда на одном грид-узле или кластере используются GPU различной архитектуры, разных поколений и производителей [5]. Для получения максимальной производительности необходимо решать задачу оптимального выбора GPU из доступных, для того, чтобы программа выполнялась за минимальное время. При этом, если выбраны GPU различной архитектуры, то следует решить еще сопутствующую задачу по универсализации программ-ядер на OpenCL для GPU различной архитектуры.

GPU состоит из унифицированных процессоров. Традиционно в процессорах существует два типа математики: векторная и скалярная. В случае векторной математики данные (операнды) представляются в виде n-мерных векторов, при этом над большим массивом данных выполняется всего одна операция. В скалярной математике операции выполняются над

парой чисел. Очевидно, что векторная обработка увеличивает скорость вычислений за счет того, что обработка целого набора (вектора) данных выполняется за один такт.

На сегодняшний день определились лидеры по производству GPU: NVIDIA и AMD. У этих производителей есть определенные отличия в архитектуре GPU. Так, например, в GPU NVIDIA уже практически не поддерживаются векторные операции, а в AMD — наоборот поддерживаются. При этом на GPU различных поколений скорость вычислений будет разной для разной длины векторов определенных типов данных.

Программа, которая использует GPU, состоит из двух частей: основной или хостовой программы, которая выполняется на CPU, и из программ-ядер (одной или нескольких), которые выполняются на GPU и написаны на специальных языках, например, OpenCL [6].

OpenCL (Open Computing Language) — открытый, нетребующий лицензионных отчислений, стандарт для универсального параллельного программирования разных типов процессоров. Стандарт предоставляет программистам переносимый и эффективный доступ ко всей мощности гетерогенных вычислительных платформ.

Программа-ядро, написанная на языке программирования OpenCL, выполняется за разное время на разных GPU [4]. Даже GPU одного производителя различных поколений имеют разную производительность при почти одинаковых параметрах. И не всегда GPU из нового модельного ряда будет превосходить GPU из предыдущего модельного ряда по скорости обработки данных, хотя, может превосходить, например, по объему памяти. Это подтверждается проведенными экспериментами. Если программа выполняется за очень малый промежуток времени (секунды), то проблема оптимального выбора GPU может не стоять так остро. Но, если для решения определенной задачи, программа работает много часов, как, например, при решении задач быстрого синтеза 2D по 2D модели или 3D сейсмограмм по 2.5D модели [7]. В таком случае при неоптимальном выборе доступных GPU на кластере или в грид-сети для выполнения программ-ядер OpenCL время работы всей программы может существенно увеличиваться. Здесь временные потери могут измеряться несколькими часами. Для таких задач предложенный метод очень актуален.

1. ПРОБЛЕМЫ ВЫБОРА GPU ИЗ МНОЖЕСТВА ДОСТУПНЫХ GPU ДЛЯ ВЫПОЛНЕНИЯ ПРОГРАММ-ЯДЕР НА OPENCL

Одним из подходов для выбора GPU компания производитель GPU NVIDIA предложила следующий механизм выбора GPU в виде функции C++ для хостовой программы, которая запускает программы-ядра на определенных GPU:

```
cl_device_id oclGetMaxFlopsDev(cl_context cxGPUContext)
{...
    // CL_DEVICE_MAX_COMPUTE_UNITS
    cl_uint compute_units;
```

```
    clGetDeviceInfo(cdDevices[current_device],
    CL_DEVICE_MAX_COMPUTE_UNITS, sizeof(compute_units),
    &compute_units, NULL);
    // CL_DEVICE_MAX_CLOCK_FREQUENCY
    cl_uint clock_frequency;
    clGetDeviceInfo(cdDevices[current_device],
    CL_DEVICE_MAX_CLOCK_FREQUENCY, sizeof(clock_frequency),
    &clock_frequency, NULL);
     max_flops = compute_units * clock_frequency;
...}
```

Так, максимальная производительность GPU равна произведению количества параллельных вычислительных ядер на максимальную тактовую частоту устройства, формула (1).

$$max\_flops = compute\_units * clock\_frequency. \quad (1)$$

где $compute\_units$ — количество параллельных ядер, $clock\_frequency$ — максимальная тактовая частота. Ниже в таблице приведены основные параметры GPU разных производителей и разных поколений, полученные программно.

Табл. 1. Основные параметры GPU

Параметр	Значение				
CL_DEVICE_NAME	Tesla M2050	GeForce GTX 480	GeForce GTX 260	Cayman	ATI RV770
CL_DEVICE_MAX _COMPUTE_UNITS	14	15	27	24	10
CL_DEVICE_MAX_ CLOCK_FREQUENCY	1147 MHz	1401 MHz	1242 MHz	830 MHz	0 MHz

Как видно из таблицы, значения $clock\_frequency$ для ATI RV770 равно 0. Согласно формуле (1) max_flops=0, что не соответствует действительности. Заметим, что параллельные ядра состоят из разного количества унифицированных процессоров в зависимости от поколения GPU (чем новее GPU, тем больше унифицированных процессоров может быть в одном параллельном ядре). Как показано на рис. 1–2, более новый GPU от NVIDIA Tesla M2050 работает быстрее, чем GPU раннего выпуска GeForce GTX 260, особенно с большими объемами данных.

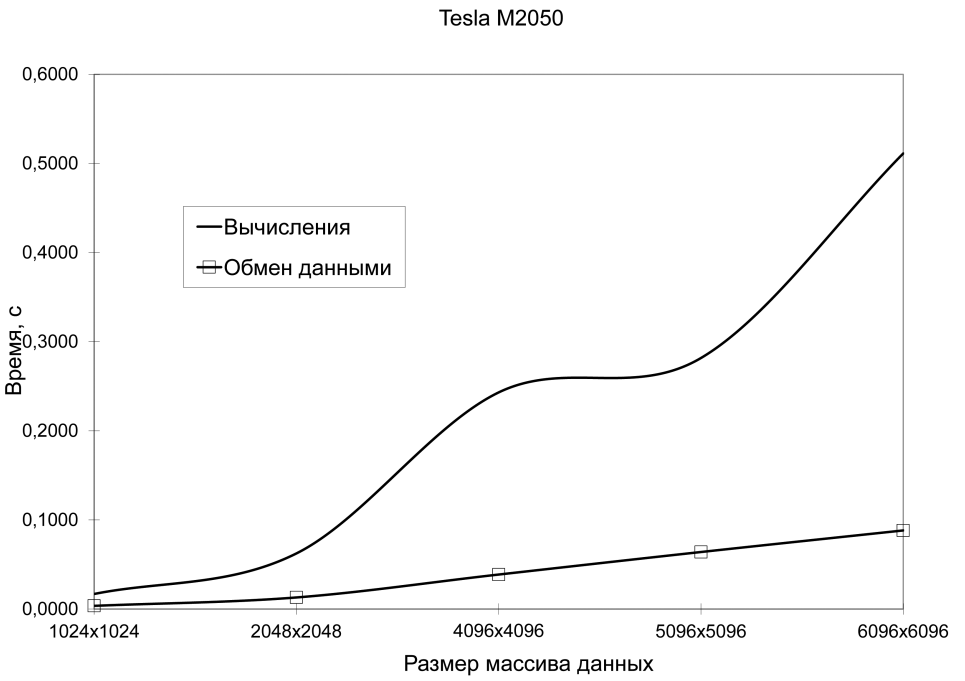

Рис. 1. Зависимость времени вычислений и времени обмена данными от объема данных для GPU NVIDIA Tesla M2050

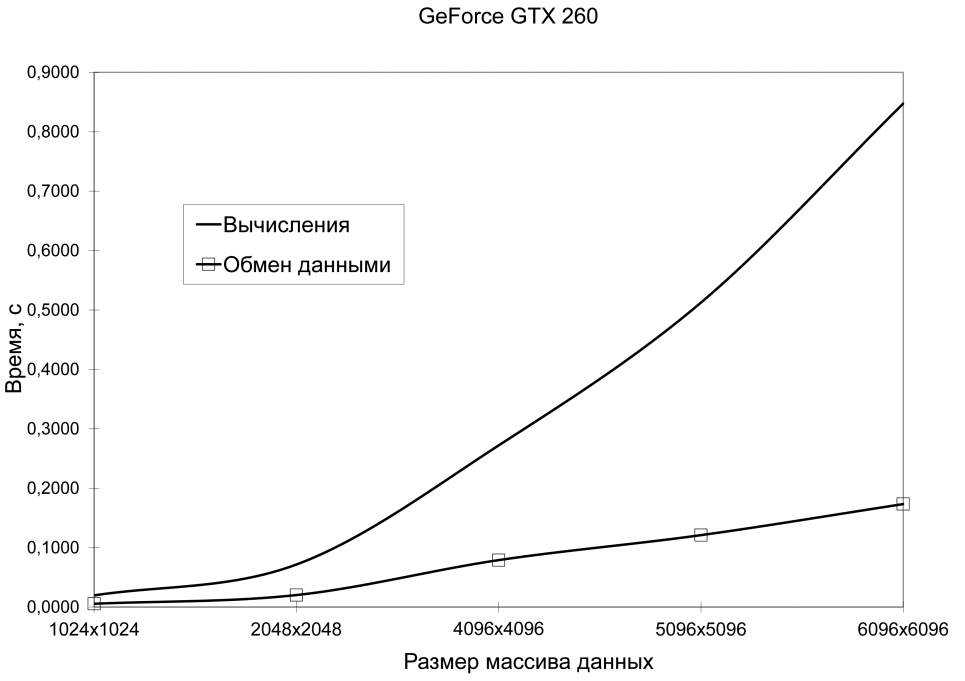

Рис. 2. Зависимость времени вычислений и времени обмена данными от объема данных для GPU NVIDIA GeForce 260

Но, по данным таблицы, $compute\_units$ для Tesla M2050 меньше, чем для GeForce GTX 260 и $clock\_frequency$ для Tesla M2050 тоже меньше, чем для GeForce GTX 260. Это противоречит результатам экспериментов, приведенных на рис. 1–2. Похожая ситуация и с GPU Cayman, что входит в состав графической карты последнего поколения AMD Radeon HD 6990. Формула (1) при решении задач на кластере с GPU даст не оптимальный результат при выборе GPU.

При одновременном использовании GPU различных архитектур возникает еще одна проблема — использование особенностей "векторных" и "скалярных" GPU для получения оптимальных вычислений. На рис. 3 показано, что при запуске программы OpenCL на CPU время выполнения программы при увеличении длинны вектора данных уменьшается.

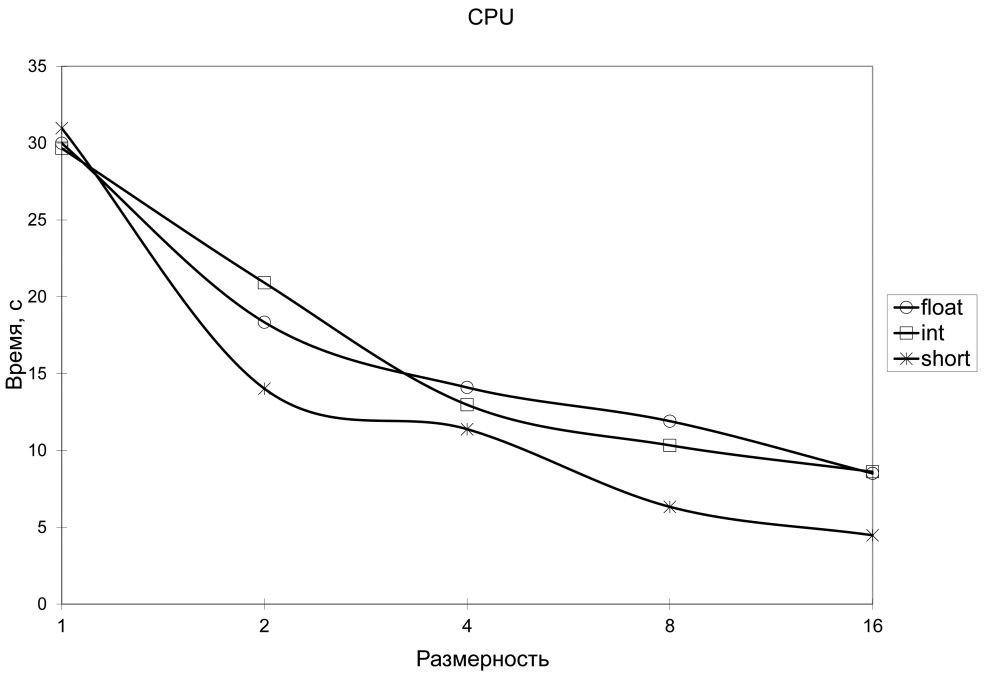

Рис. 3. Зависимость времени вычислений от типа данных и длинны вектора данных на CPU

На рис. 4–6 показано, что для GPU наилучший результат работы программы достигается для определенных длин векторов. При этом зависимость времени работы программы от длинны вектора данных нелинейная.

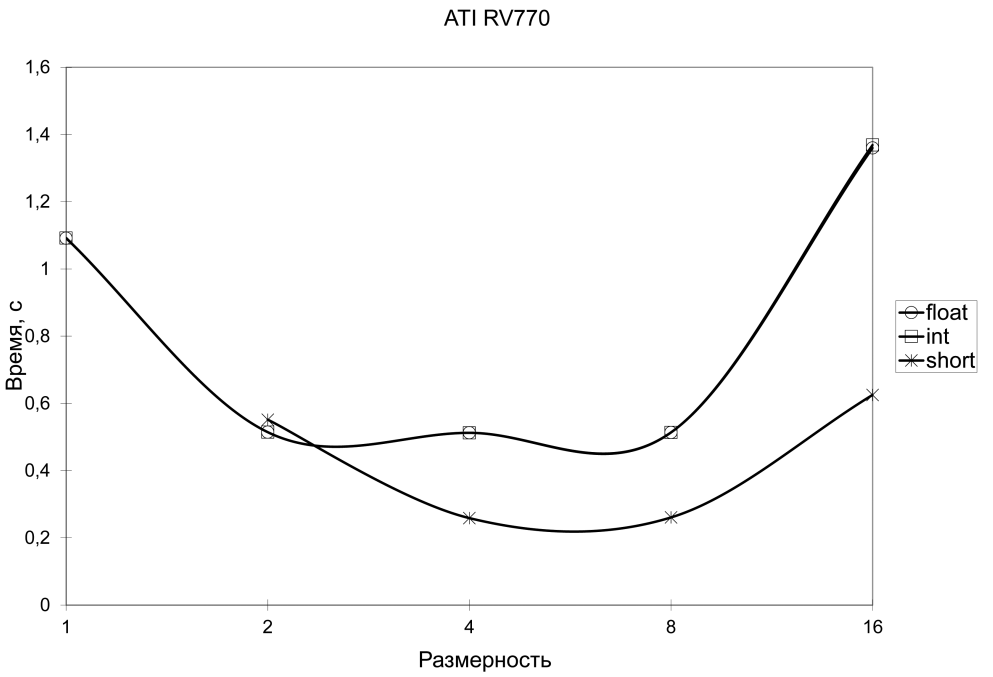

Рис. 4. Зависимость времени вычислений от типа данных и длинны вектора данных на ATI RV770

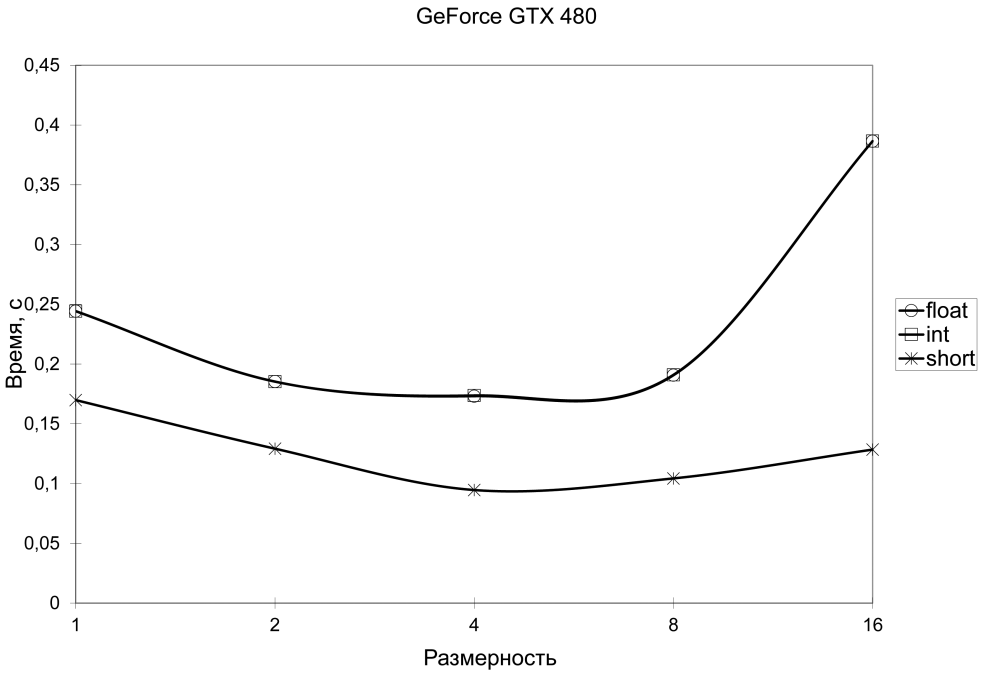

Рис. 5. Зависимость времени вычислений от типа данных и длинны вектора данных на GeForce 480

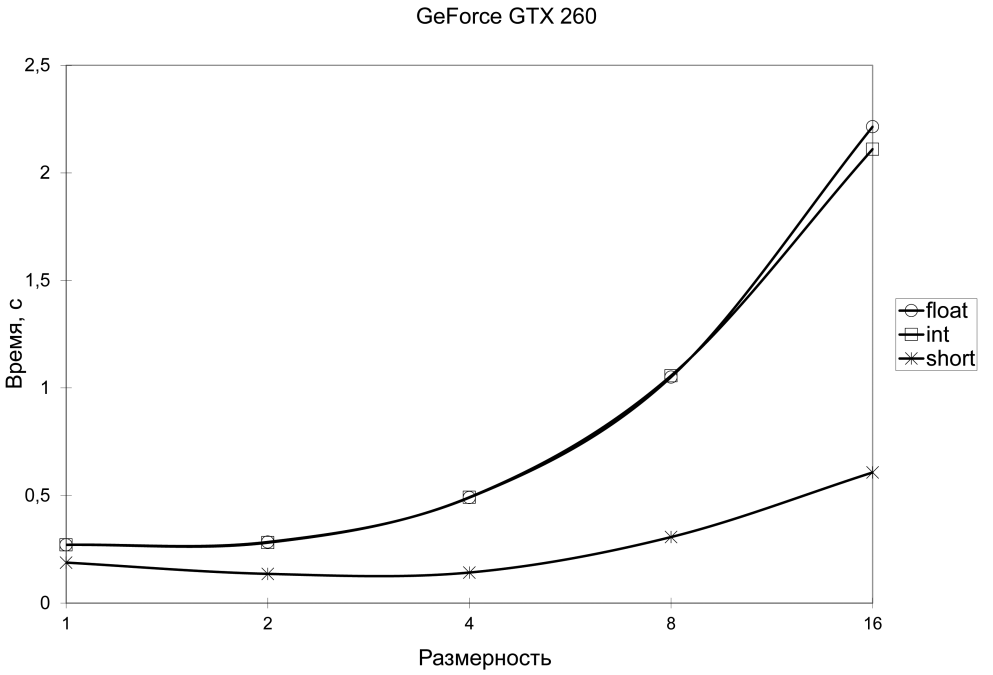

Рис. 6. Зависимость времени вычислений от типа данных и длинны вектора данных на GeForce 260

Соответственно программа на OpenCL без поддержки векторных операций может дать не оптимальное время вычислений на тех GPU, которые поддерживают векторные операции. Если программа на OpenCL рассчитана на поддержку векторных операций, то на некоторых GPU (рис. 4) при определенных длинах векторов можем потерять в производительности (получим неоптимальные вычисления). Разрабатывать и поддерживать несколько вариантов программ-ядер на OpenCL возможно лишь в том случае, когда программа небольшая. В больших программах это очень трудоемкая задача. Для решения этой проблемы авторами было предложено использование шаблонов функций C++ для универсального использования языка программирования OpenCL для "векторных" и "скалярных" GPU. Суть метода получения универсальности программы OpenCL на "векторных" и "скалярных" GPU [8] состоит в следующем:

- определить типы данных, которые поддерживает GPU;
- определить длинну вектора данных, для которого время вычислений будет минимальным;
- программно (автоматически) настройка программы-ядра OpenCL на работу с оптимальной длинной вектора для конкретного GPU;
- вызвать нужную шаблонную функцию для хостовой программы.

2. Алгоритм оптимизации запуска программ-ядер OpenCL на нескольких GPU

Для оптимального выбора GPU предложен алгоритм, который представлен на рис. 7.

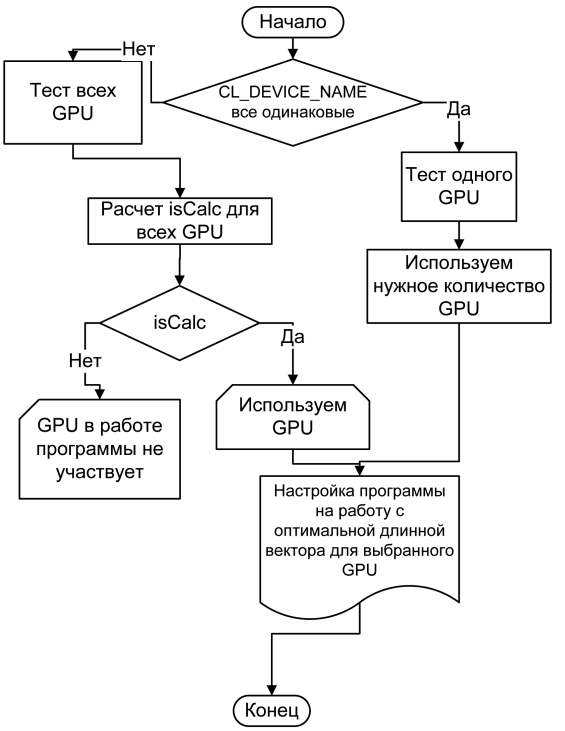

Рис. 7. Алгоритм выбора GPU для программ-ядер на OpenCL и универсализация программ-ядер для работы с векторными и скалярными GPU

Основой алгоритма является набор вычислительных тестов различной сложности, которые необходимо выполнить на GPU. Тесты содержат основные операции, используемые в программе. Время работы теста должно быть существенно меньше времени работы всей программы. Например, тест, который выполняет основные операции с двумя массивами и сохраняет результат в третьем, в случае размерности массивов 4096×4096 выполняется на GPU GeForce GTX 480 0.0157 с, а на GPU GeForce GTX 260 — 0.0369 с. В задачах моделирования сейсмограмм таких операций выполняется десятки тысяч. Поэтому, когда программа-ядро будет выполняться на GPU GeForce GTX 260, а не на GeForce GTX 480, проигрыш во времени будет значительным. Также целесообразно выполнить тест передачи данных определенного объема на GPU, так как это время существенно выше, чем при работе с оперативной памятью компьютера. Для GeForce GTX 260 время загрузки тестового массива 4096×4096 — 0.034229 с, а для GTX 480 — 0.022935 с. Это время является незначительным, если таких операций мало — перед началом вычислений данные загружаются на GPU, а после

окончания всех длительных вычислений результаты с GPU загружаются в оперативную память хоста. Но, если на протяжении длительной работы программы через несколько итераций вычислений происходит обмен данными между GPU и хостом, то суммарное время обмена данными будет существенным и его необходимо учитывать при разработке и проведении тестов.

По предложенной формуле (2) определяется целесообразность использования конкретного GPU из всего набора доступных GPU для запуска на нем программы-ядра на OpenCL и решения части или всей задачи.

$$isCalc = \begin{cases} true, & \text{если } P_i <= c * max(P_{1..n}); \\ false, & \text{если } P_i > c * max(P_{1..n}). \end{cases} . \tag{2}$$

где P_i — общее время выполнения теста на i-м GPU, c-коэффициент порога $0, ..., 1$, зависит от сложности задачи, $i = 1, ..., n$, n — количество доступных GPU на кластере или грид-узле.

Время P_i — можно представить в виде суммы времени работы программы-ядра и времени операций обмена данными между хостовой программой и программой-ядром, формула (3).

$$P_i = k_1 * T_i + k_2 * T_i'. \tag{3}$$

где T_i — время выполнения одной итерации теста на i-м GPU, k_1 — количество итераций в одном тесте, T_i' — время обмена данными с i-м GPU, k_2 — количество операций обмена данными между хостом и программой-ядром в одном тесте.

3. Выводы

При запуске задач на гетерогенном GPU кластере, архитектура которого заранее не известна, как это зачастую бывает в случае с вычислениями в грид-среде, предложенный метод дает возможность:

- провести быстрое тестирование производительности работы GPU, при этом учесть особенности работы GPU разной архитектуры и разных поколений, применительно к особенностям решаемой задачи;
- оптимально выбрать GPU, исключив из набора менее производительные GPU для конкретной задачи;
- использовать преимущества векторных операций на GPU, поддерживающих векторные операции;
- минимизировать проблемы, которые могут возникнуть, при запуске "векторной" программы на GPU, которые не поддерживают векторные операции;
- и, как результат, выполнить программу за минимально возможное время для конкретного кластера или грид-узла.

Литература

1. Unstructured grid applications on GPU: performance analysis and improvement / [Lizandro Solano-Quinde, Zhi Jian Wang, Brett Bode, Arun K. Somani] — In Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-4), ACM, New York, NY, USA, Article 13, 8 pages.
2. Corrigan A. Running Unstructured Grid Based CFD Solvers on Modern Graphics Hardware / Corrigan A., Camelli F., Rainald L. — 19th AIAA Computational Fluid Dynamics, Jun, 2009.
3. Український національний грід / [Електронний ресурс] : http://ung.in.ua
4. Chris Jang. OpenCL™ Optimization Case Study: GATLAS — Designing Kernels with Auto-Tuning [Електронний ресурс] : http://golem5.org/gatlas/CaseStudyGATLAS.htm / Chris Jang.
5. Marcus Hinders. GPU Computations in Heterogeneous Grid Environments, Joint Research Report [Електронний ресурс] : http://www.techila.fi/technology/technology-docs / Marcus Hinders.
6. OpenCL — The open standard for parallel programming of heterogeneous systems [Електронний ресурс] : http://www.khronos.org/opencl/.
7. Применение CUDA для быстрого синтеза 3D-сейсмограмм по 2.5D-модели /[Мармалевский Н. Я., Мерщий В. В., Роганов Ю. В., Тульчинский В. Г., Ющенко Р. А.] // Вычисления в геологии. — 2011. — № 3. — С. 8–12.
8. Лавренюк А. М. Оптимізація обчислень за допомогою універсального використання програми OPENCL з "векторними" та "скалярними" GPU / А. М. Лавренюк, С. І. Лавренюк // Компьютерная математика. — 2012. — Вып. 1. — С. 102–110.

Факультет кибернетики, Киевский национальный университет имени Тараса Шевченко, ул. Владимирская, 64, Киев, 01601, Украина.
Физико-технический институт, Национальный технический университет Украины "КПИ", пр. Победы, 37, Киев, 03056, Украина.

Поступила 01.02.2013