УДК 519.684.4

АНАЛИЗ ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНОЙ РЕАЛИЗАЦИИ МЕТОДА ГАУССА

© 2013 г. ***Д.А. Баранов, И.В. Влацкая***

Оренбургский госуниверситет

mois@mail.osu.ru

Поступила в редакцию 19.012.2012

Описываются некоторые особенности стандарта OpenCL, проявляющиеся при использовании на аппаратном обеспечении архитектуры CUDA. В частности, анализируется влияние размера локальной рабочей группы на время вычислений. В качестве примера используется OpenCL-реализация метода Гаусса решения систем линейных алгебраических уравнений.

Ключевые слова: GPGPU, OpenCL, SIMT, параллельные вычисления, метод Гаусса, системы линейных алгебраических уравнений

Современную информационную инфраструктуру сложно представить без многопроцессорных вычислительных комплексов. Параллельные вычисления используются для удовлетворения большинства информационных потребностей человека, начиная от теоретических исследований до сугубо практических приложений, таких, как прогнозирование погоды. Не удивительно, что растущая потребность в эффективной обработке больших объёмов данных привела к появлению большого количество подходов, технологий и, как следствие, программно-аппаратных архитектур для исполнения параллельных алгоритмов обработки данных. Особый интерес представляет относительно новое направление «General-Purpose Graphics Processing Units» (GPGPU) — программирование графического процессора общего назначения. GPGPU появилось в результате активного развития графических процессоров в последнее десятилетие под влиянием игровой и киноиндустрии. На данный момент графические процессоры обладают значительно большей теоретической пиковой производительностью, чем центральные процессоры той же ценовой категории.

До появления GPGPU решение задач общего назначения на графических процессорах было крайне затруднительно. Объясняется это особенностью архитектуры современных графических процессоров, оптимизированной для решения задач обработки геометрических и графических данных. Ситуация изменилась с появлением разработок AMD (на тот момент ATI) и NVIDIA для программирования графических процессоров собственного производства: ATI Stream SDK и NVIDIA CUDA (Compute Unified Device Architecture). Несмотря на то, что графические процессоры AMD и NVIDIA решали одинаковые задачи, их технологии GPGPU несовместимы и работают исключительно на аппаратном обеспечении фирмы-производителя. Впоследствии группа Khronos, в которую вошли AMD, NVIDIA и другие производители графических процессоров и вычислительных устройств, разработала стандарт GPGPU OpenCL, целью которого была поддержка большого количества платформ. Следует отметить, что AMD позже отказалась от собственной разработки в пользу OpenCL. На данный момент OpenCL работает на графических процессорах как AMD, так и NVIDIA, а также на любых x86 и amd64-совместимых центральных процессорах. Поэтому OpenCL является универсальным инструментом разработки высокопроизводительных параллельных алгоритмов, обеспечивающим работу большей части современной компьютерной техники.

Тем не менее, стандарт OpenCL относительно молод (первая версия была опубликована 9 декабря 2008), и на данный момент существуют плохо документированные особенности, которые отрицательно влияют на его применимость. Так, анализ источников информации по данной тематике показал, что многие программисты сталкиваются с проблемами производительности OpenCL, особенно на аппаратном обеспечении производства NVIDIA. Существуют результаты экспериментов, подтверждающие эти проблемы, и результаты сравнительных испытаний с другими технологиями GPGPU, в частности CUDA. При этом некоторые авторы явно заостряют внимание на преимуществе технологии CUDA над OpenCL в производительности.

Работа выполнена в соответствии с проектом N 14.B37.21.0176 Федеральной целевой программы "Научные и научно-педагогические кадры инновационной России" на 2009–2013 гг.

В качестве примера на рис. 1 приведён график зависимости времени расчётов клеточных автоматов игры «Жизнь» от размеров ребра кубического поля из работы [1].

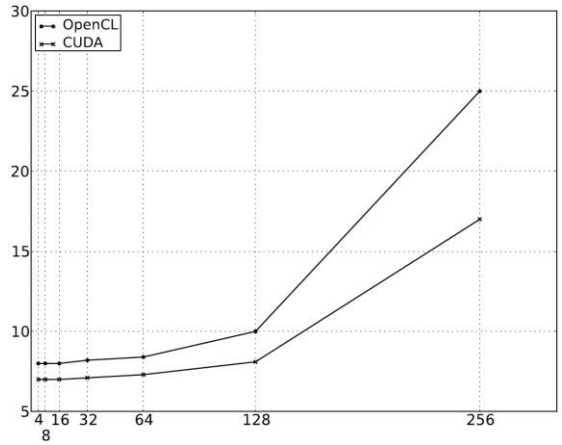

Рис. 1. Зависимость времени расчетов от размера ребра кубического «поля» при использовании CUDA и OpenCL

Кроме того, некоторые результаты экспериментов показывают наличие «выбросов» времени вычислений в зависимости от объёма входных данных [2].

Для подробного изучения причины появления «выбросов» времени вычислений необходимо рассмотреть важные особенности архитектуры графических процессоров NVIDIA и стандарта OpenCL. Следует заметить, что как бы не называлась аппаратная архитектура того или иного поколения графических процессоров NVIDIA (например Fermi, Kepler, Tesla), фактически все они являются подмножеством архитектуры CUDA.

CUDA является проприетарной (закрытой) программно-аппаратной архитектурой и, следовательно, пригодна лишь для графических процессоров NVIDIA. В то же время OpenCL является открытым стандартом, разработанным группой Khronos, в состав который входит множество производителей вычислительных устройств. Стандарт OpenCL разработан для использования на большом количестве платформ и устройств, в том числе на CPU и GPU. Следовательно, область его применения значительно шире. Тем не менее, архитектура CUDA оказала большое влияние на разрабатываемый стандарт, поскольку к тому времени CUDA уже заняла большую часть рынка высокопроизводительных вычислений.

Основой архитектуры CUDA является масштабируемый массив многопоточных мультипроцессоров (Streaming Multiprocessors, SMs). При запуске вычислений CUDA разделяет вычислительные потоки на блоки и распределяет их по мультипроцессорам. На одном мультипроцессоре могут выполняться несколько блоков процессов, но один блок процессов не может одновременно выполняться на нескольких мультипроцессорах. Мультипроцессоры разработаны для одновременного выполнения большого количество потоков, для этого используется специальная архитектура, называемая SIMT (Single-Instruction, Multiple-Thread — Одна команда, множество потоков). Кроме того, используется конвейер команд для достижения параллелизма на уровне команд в пределах одного потока. Однако, следует помнить, что ядра графического процессора устроены гораздо проще CPU, в частности отсутствует предсказание ветвлений. Более подробно архитектура CUDA описана в [3, 4].

Для проведения вычислительного эксперимента, необходимого для анализа проблемы выбросов, использовался метода Гаусса [5] решения систем линейных алгебраических уравнений (СЛАУ). Рассмотрим простейшую реализацию этого метода на OpenCL.

Реализация алгоритма метода Гаусса с использованием OpenCL разделена на четыре функции («ядра» в терминологии OpenCL), по два на прямой и обратный ход. На каждом шаге прямого хода вызывается сначала первое (gauss_fwd_pre), затем второе ядро (gauss_fwd). Пусть текущая итерация имеет номер k, размерность СЛАУ — n. В первом ядре прямого хода производится вычисление делителей для каждой строки на текущей итерации:

$$A_{ik}^{k+1} = \frac{A_{ik}^{k}}{A_{kk}^{k}},\ i = \overline{k+1,n}. \qquad (1)$$

Во втором ядре прямого хода вычисляются новые значения матрицы A и вектора b:

$$A_{ij}^{k+1} = A_{ij}^{k} - A_{kj}^{k} A_{ik}^{k},\ i = \overline{k+1,n}, j = \overline{k+1,n},$$
$$b_{i}^{k+1} = b_{i}^{k} A_{ik}^{k},\ i = \overline{k+1,n}. \qquad (2)$$

Заметим, что формула (2) не обнуляет элементы k-го столбца, это необходимо, т. к. все элементы A_{ij}^{k+1} вычисляются параллельно и используют значения k-го столбца. Обнуление этого столбца должно производиться после вычисления всех элементов A_{ij}^{k+1}, однако, это не обязательно — алгоритм в дальнейшем просто игнорирует эти элементы (т. е. все элементы ниже главной диагонали), считая их нулевыми.

Рассмотрим обратный ход метода Гаусса. На каждой итерации обратного хода вызываются два ядра: gauss_bwd_prepare и gauss_bwd. В

первом ядре обратного хода вычисляется x_k :

$$x_k = \frac{b_k^k}{A_{kk}^k} \quad (3)$$

Очевидно, для первого шага эта формула работает, а для последующих шагов это становится возможным благодаря второму ядру:

$$b_i^{k+1} = b_i^k - x_k A_{ik}^k,\ i = \overline{1,k-1}. \quad (4)$$

Как видно из (4), вычисленное значение x_k сразу используется для модификации следующих уравнений. Фактически происходит перенос компонент $x_k A_{ik}$ в правую часть уравнений, но в левой части они не обнуляются, а просто игнорируются в дальнейшем. Ядро (3), очевидно, не оптимально, поскольку использует всего один процессор, но его использование более оправдано, чем пересылка из памяти графического процессора в оперативную память и обратно.

При таком итерационном подходе есть два способа задания размеров глобальной рабочей области, поскольку на каждом шаге (как прямого, так и обратного хода) она будет сужаться. Первый способ заключается в задании фиксированного размера рабочей области, тогда ядра будут вызываться для всех элементов матрицы на каждом шаге. Однако с каждым шагом требуется вычислить всё меньшее количество элементов матрицы, что влечёт дополнительные накладные расходы. Кроме того, в коде ядер необходимо предусмотреть выход процесса без вычислений в зависимости от итерации и индекса элемента с помощью условного оператора. Второй подход заключается в изменении размеров глобальной рабочей области на каждом шаге. При таком подходе нет необходимости использовать условные операторы, т. к. будут вычисляться только «правильные» элементы, что должно снизить накладные расходы.

Описываемые ниже наблюдения были получены при использовании первого подхода, т. к. второй подход значительно усложняет анализ экспериментальных данных. Следует отметить, что при использовании данной реализации алгоритма отмечаются временные выбросы в виде локальных максимумов на матрицах некоторой размерности, т. е. значительное падение производительности. График зависимости времени выполнения алгоритма от размерности матрицы изображён на рис. 2.

Подробное исследование проблемы показало, что «выбросы» встречаются на матрицах, для которых выполняется условие:

$$\exists!k,\ \left[N+1,k\right] = 0,\ k \neq 1 \quad (1)$$

где

- N — размерность СЛАУ;
- $[a, b]$ — остаток от деления a на b;
- N+1 — размерность СЛАУ + вектор-столбец B.

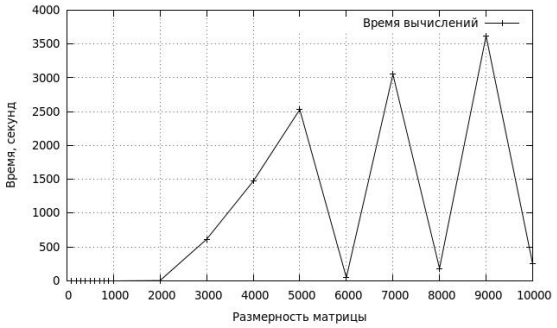

Рис. 2. Зависимость времени выполнения алгоритма от размерности матрицы

Фактически выражение (5) означает, что N+1 — простое число. Анализ рекомендаций по программированию для CUDA от NVIDIA [3] позволил сделать предположение о существовании проблемы оптимального разделения глобальной рабочей области на рабочие группы (следует заметить, что в описываемой реализации размер рабочий группы не задавался явно и, следовательно, устанавливался драйвером). Действительно, согласно документации, размер рабочей группы должен быть делителем размера глобальной рабочей области и при этом не превышать некоторого ограничения конкретного вычислительного устройства (величина ограничения может быть получена с помощью функции OpenCL clGetDeviceInfo, вызванной с параметром CL_DEVICE_MAX_WORK_GROUP_SIZE).

Для большинства графических процессоров это ограничение составляет 1024 — это максимальное число вычислительных процессов в составе одной группы, оно составляется из произведения чисел всех рабочих процессов по всем измерениям внутри группы. Кроме того, особенности архитектуры графических процессоров, в частности концепция «Warp» от NVIDIA, позволяют максимизировать производительность параллельного алгоритма путём задания размеров рабочей группы кратными специальной величине, которая так же может быть запрошена у вычислительного устройства. Эта величина называется «Preferred Work Group Size Multiplier» (PWGSM) и её можно получить с помощью функции OpenCL clGetKernelWorkGroupInfo, вызванной с параметром CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE. Для графических процессоров NVIDIA эта величина равна 32. Фактически

«Warp» – это минимальная группа вычислительных процессов (до 32), которой может управлять планировщик графического процессора.

Практика подтвердила, что в случае простых чисел в качестве размерности глобальной рабочей области драйвер автоматически устанавливает размер рабочей группы равным единственному делителю размера глобальной рабочей области, меньшему 1024, т. е. 1. Очевидно, что большое число рабочих групп, состоящих из 1 элемента, не может быть эффективно распределено по вычислительным устройствам графического процессора. В этом случае большая часть времени работы алгоритма уходит на работу планировщика и сводит на нет преимущества SIMT-архитектуры.

Заметим, что многие авторы уделяют большое внимание работе с памятью графических процессоров, например [6]. Однако обычно опускаются важные вопросы разделения глобальной рабочей области на рабочие группы, что можно объяснить относительной новизной технологий GPGPU. Этот факт следует учитывать при разработке рабочих программ по предметам, связанным с параллельным программированием, наравне с работами [7–9].

Для решения проблемы выбросов было проведено исследование производительности описанной выше реализации метода Гаусса в зависимости от размера рабочей группы и размерности СЛАУ.

Прежде чем перейти к описанию и анализу результатов исследования, необходимо рассмотреть ещё одну очевидную проблему, а именно неделимость размеров глобальных рабочих областей, потенциально используемых во многих практических задачах на PWGSM. В некоторых задачах размерность глобальной рабочей области имеет лишь один делитель меньше 1024, равный единице. В такой ситуации можно разделить глобальную рабочую область на две части способом, удовлетворяющим следующим условиям:

$$\left[SZ1, OGS \cdot K\right] = 0, K = \left\{M, OSG\right\}, \quad (6)$$

$$SZ2 = M - OSG \cdot K, \quad (7)$$

где

- M — размерность матрицы ($M = N+1$ в описанном примере);
- OSG — оптимальный размер группы;
- $SZ1$ — размер первой части глобальной рабочей области;
- $SZ2$ — размер второй части глобальной рабочей области;
- $\{a, b\}$ — целая часть от деления a на b.

Как видно, размер второй части является остатком от деления и может быть произвольным числом меньше OSG. Способ вычисления второй части, в общем случае, зависит от специфики задачи и может производиться как отдельной реализацией алгоритма (например последовательной CPU-версией), так и повторным вызовом OpenCL-ядра с установкой соответствующего размера глобальной рабочей области (для этой цели можно создавать отдельную копию ядра). Очевидно величина остатка $SZ2$ будет влиять на общее время работы алгоритма, поэтому оптимальное значение OSG потенциально может не привести к наиболее быстрому достижению результата.

Ниже приведены результаты экспериментов с различными размерами локальной рабочей группы, устанавливаемыми для второго ядра описанной выше реализации. Размер матрицы так же варьируется.

На рис. 3 приведён график зависимости времени вычислений от размера рабочей группы для размерностей матрицы 100-900 с шагом 100.

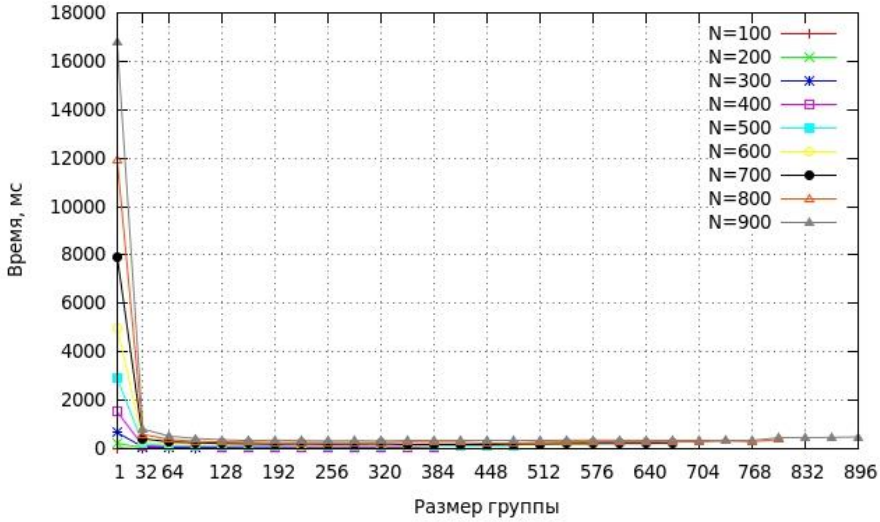

Рис. 3 Зависимость времени вычислений от размера рабочей группы

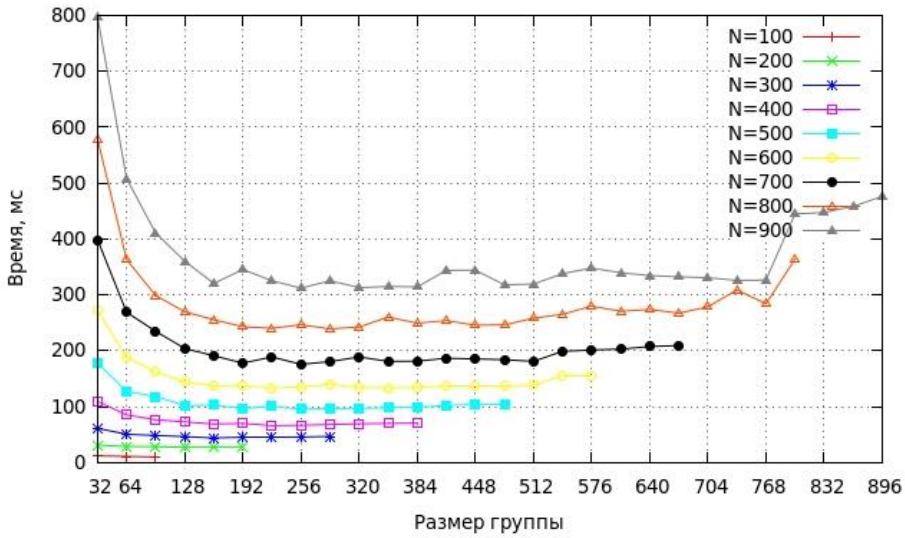

Рис. 4 Зависимость времени вычислений от размера рабочей группы, секция 32-896

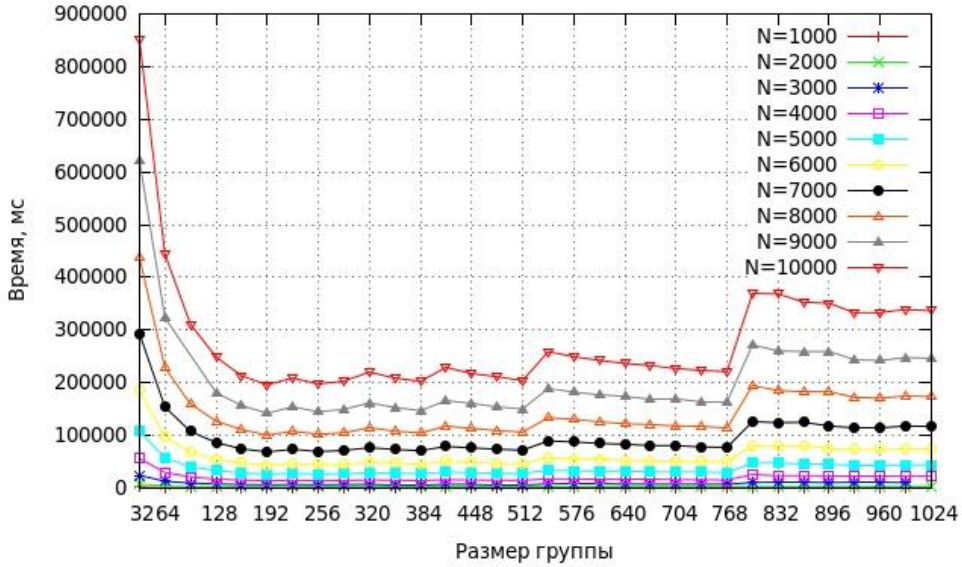

Рис. 5 Зависимость времени вычислений от размера рабочей группы для матриц большой размерности

Как видно из графика, размер группы, равный единице, максимально неэффективен на любом размере матрицы. Рассмотрим данный график более детально, для этого необходимо отбросить результат с размером группы равным единице. Результат приведён на рис. 4.

Из графика следует, что для большинства размерностей матрицы оптимальный размер группы лежит в пределах 192-384 элемента. Тем не менее, на матрицах размерности < 1000 остаток *SZ*2 может оказывать значительное влияние на результат, поэтому необходимо рассмотреть матрицы большей размерности. На рис. 5 изображён график зависимости времени вычислений от размера рабочей группы для размерностей матрицы 1000-10000 с шагом 1000.

Рассмотрим секцию 128-384 графика подробнее, она приведена на рис. 6.

Согласно графику, оптимальным размером рабочей группы является 192, что подтверждается табличными данными (с подробными результатами экспериментов, а также с исходным кодом можно ознакомиться в открытом репозитории на Github [10]).

Очевидно, что значение 192 не является оптимальным для любого вычислительного устройства. Тем не менее, это значение достаточно просто выводится из характеристик устройства, которые можно получить через функции OpenCL. Характеристики вычислительного устройства (в данном случае графического процессора), на котором производился данный эксперимент, можно получить на сайте производителя [11]. Однако там не указан один важный параметр — число мультипроцессоров (Comput Unit в терминологии OpenCL). Как уже упоминалось, один вычислительный блок (рабочая группа) не может быть разделён на не-

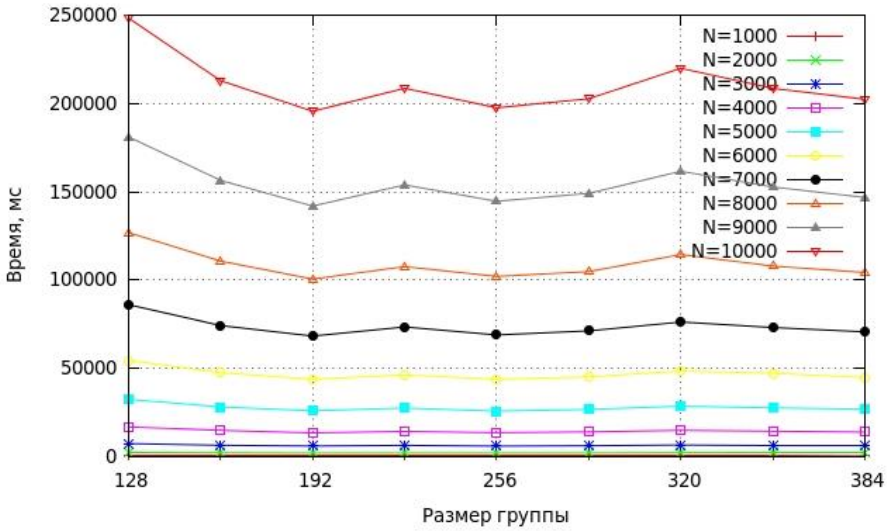

Рис. 6 Зависимость времени вычислений от размера рабочей группы для матриц большой размерности, секция 128-384

сколько мультипроцессоров, но несколько блоков могут выполняться на одном мультипроцессоре. Число Compute Unit вычислительного устройства можно получить с помощью вызова функции clGetDeviceInfo, вызванной с параметром CL_DEVICE_MAX_COMPUTE_UNITS.

Для оптимальной загрузки всех ядер вычислительного устройства необходимо минимизировать работу планировщика и обеспечить отсутствие «простаивающих» ядер. Для этого размер рабочей группы должен быть кратен числу ядер мультипроцессора и PWGSM. Кроме того, размер рабочей группы должен превышать общее число ядер вычислительного устройства. Вычисление числа ядер мультипроцессора осуществляется следующим образом:

$$mpCoreCount = \frac{totalCoreCount}{mpCount},$$

где

- totalCoreCount — общее число ядер («потоки») устройства, в данном случае 144;
- mpCount — число мультипроцессоров, для данного вычислительного устройства это число 3.

Следовательно, для данного устройства число ядер мультипроцессора составляет 144/3 = 48. Наименьшее общее кратное 32 и 48, большее 144, равно 192 — это и есть оптимальный размер рабочей группы для данного вычислительного устройства, что подтверждается результатами эксперимента.

Проанализировав полученные результаты, можно заключить, что выбор размера рабочей группы существенно влияет на общую производительность OpenCL-реализации вычислительного алгоритма. Автоматическая установка этого параметра может привести к существенной потере производительности, поэтому размер группы всегда должен устанавливаться вручную, даже если это требует модификации алгоритма (в частности, разделения глобальной рабочей области). Кроме того, на основе анализа экспериментальных данных было определено условие оптимальности размера рабочей группы: оптимальный размер рабочей группы должен превышать общее число ядер вычислительного устройства и при этом быть кратным значению PWGSM и числу ядер мультипроцессора.

Результаты данной работы используются для разработки OpenCL-реализации параллельного алгоритма обучения нечёткой нейронной сети Такаги-Сугено-Канга.

Работа выполнена в соответствии с проектом N 14.B37.21.0176 Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009–2013 гг. Выполнение работ по проекту соответствует выполняемым в последнее время научно-образовательным программ по развитию суперкомпьютерного образования в стране [7–9, 12].

Список литературы

1. Алексеенко А.Е., Казённов А.М. Реализация клеточных автоматов «игра «Жизнь»» с применением технологий CUDA и OpenCL // Математические основы и численные методы моделирования. 2010. Т. 2. № 3. С. 323–326. Московский физико-технический институт.

2. Запрягаев С.А., Карпушин А.А. Применение графического процессора в задаче многократного численного интегрирования на основе обобщённой формулы Симпсона // Вестник Воронежского государственного университета, 2011. № 1. С. 150–156.

3. NVIDIA CUDA C programming Guide, 2012.

4. Запрягаев С.А., Карпушин А.А. Применение графического процессора в ресурсоемких вычислениях на базе библиотеки OpenCL // Вестник Воронежского государственного университета. 2010. № 2.

5. Самарский А.А., Гулин А.В. Численные методы. М.: Наука, 1989.

6. Манушин И.А. Использование технологии OpenCL для разработки высоконагруженных приложений // RSDN Magazine. 2012. № 1. С. 12–20.

7. Гергель В.П., Стронгин Р.Г. Опыт нижегородского университета по подготовке специалистов в области суперкомпьютерных технологий. // Вестник Нижегородского университета им. Н.И. Лобачевского. 2010. № 3(1). С. 191–199.

8. Воеводин В.В., Гергель В.П. Суперкомпьютерное образование: третья составляющая суперкомпьютерных технологий. // Вычислительные методы и программирование: новые вычислительные технологии. 2010. Т. 11, № 2. С. 117–122.

9. Гергель В.П., Линев А.В., Мееров И.Б., Сысоев А.В. Об опыте проведения программ повышения квалификации профессорско-преподавательского состава по направлению высокопроизводительные вычисления. // Открытое и дистанционное образование. 2010. № 3. С. 15–20.

10. http://www.nvidia.ru/object/product-geforce-gt-555m-ru.html (дата обращения: 13.09.2012)

11. https://github.com/Dem0n3D/solecl (дата обращения: 13.09.2012)

12. Воеводин В.В., Гергель В.П., Соколинский Л.Б., Демкин В.П. и др.. Развитие системы суперкомпьютерного образования в России: текущие результаты и перспективы. // Вестник Нижегородского университета им. Н.И. Лобачевского. 2012. № 4. С. 268–274.

ANALYSIS OF THE IMPACT OF SIZE OF THE WORK GROUP TO OPENCL-IMPLEMENTATION OF THE COMPUTING ALGORITHMS WITH THE GAUSS METHOD AS EXAMPLE.

D.A. Baranov, I.V. Vlatskaya

This paper describes some of the features of the OpenCL technical standard, specially on the hardware architecture CUDA. In particular, it analyzes the impact of the size of the work group to the computation time. The Gauss method have been used as an example of application of the OpenCL-implementation.

Keywords: GPGPU, OpenCL, SIMT, parallel computing, Gauss, systems of linear equations