Особенности адаптации вычислительных алгоритмов под параллельную архитектуру графических акселераторов

С.М. Вишняков, С.В. Ковальчук, А.В. Бухановский

В работе обсуждаются вопросы отображения вычислительных алгоритмов на параллельную архитектуру GPU-акселератора. В качестве примера рассматривается задача параметрической оптимизации методом случайного поиска.

1. Введение

В последние годы интенсивное развитие получила специфическая отрасль высокопроизводительных вычислений – расчеты на системах с параллельными акселераторами, как дополнительными устройствами, принимающими на себя существенную часть вычислительной нагрузки работающего приложения [1]. К таким устройствам относятся, в частности, GPU (Graphic Processor Unit)-устройства, предназначенные для работы с графикой. Современные GPU по сути являются многопроцессорными системами SIMD-архитектуры с достаточно высокой (до 0,5 Тфлопс) пиковой производительностью. По сравнению с традиционными архитектурами (например, кластерами), они обладают несопоставимо низкой характеристикой «цена/производительность», что стимулирует интерес к использованию GPU не только для обработки графической информации, но и для решения произвольных вычислительных задач [2]. В частности, к таким задачам относятся сортировки больших объемов данных [3], решения систем линейных уравнений [4], уравнений математической физики [5-7] и пр.

Параллельное программирование для GPU традиционно использует графический программный инструментарий, например, на основе библиотеки OpenGL [8] или DirectX [9]. Для удобства работы с ними применяются пакеты более высокого уровня, например, Gg [10], HLSL [11] и OpenGL Shading Language [12]. Однако реализация вычислительных задач общего плана требует использования более универсальных средств, облегчающих отображение задач различного типа на архитектуру GPU-устройств и позволяющих абстрагироваться от особенностей работы графических алгоритмов. Например, компанией nVidia разрабатывается набор библиотек CUDA SDK [13], предназначенный для использования вычислительных мощностей видеокарт и, в перспективе, специализированных ускорителей вычислений для решения вычислительных задач общего назначения. Кроме того, существует ряд проектов по разработке методов и средств, предназначенных для отображения вычислительных алгоритмов на GPU-устройства различных производителей, например, язык C$ [14].

Несмотря на высокую производительность, вычислительные акселераторы на базе GPU являются специфическим классом многопроцессорных систем, характеризуемым существенными ограничениями на масштабируемость, использование памяти и управление данными и задачами. Потому представляет интерес изучение способов отображения различных типов вычислительных алгоритмов на GPU-архитектуру с целью выявления ключевых особенностей этого процесса и факторов, влияющих на получаемую производительность. В данной работе рассматриваются особенности этого процесса на примере алгоритма параметрической оптимизации методом случайного поиска.

2. Особенности архитектуры GPU-устройств и средства отображения вычислительных алгоритмов

Акселератор на базе GPU представляет собой набор вычислительных узлов (мультипроцессоров), состоящих из некоторого числа арифметико-логических устройств (АЛУ). Он имеет SIMD-архитектуру. Иначе говоря, в любой момент времени все АЛУ одного мультипроцессора выполняют одинаковую последовательность инструкций над разными наборами данных, расположенных в памяти GPU-устройства. В данной работе в качестве примера рассматривается

акселератор nVidia GeForce 8800 GTX, который имеет 16 мультипроцессоров, по 8 АЛУ в каждом.

Поскольку в SIMD-устройствах каждый мультипроцессор конфигурируется определенным образом для выполнения заданной последовательности команд на множестве входных данных, то перед разработчиком стоит задача формирования последовательности инструкций, решающей поставленную задачу, конфигурации устройства и передачи в устройство потока входных данных. При использовании nVidia CUDA SDK, эта процедура выглядит следующим образом (рис. 1): разработчик описывает ядро (kernel), то есть процедуру, которая будет исполняться на GPU над потоком данных и при запуске ядра на GPU задает ему конфигурацию на основе описанной ниже иерархии. Каждый поток, физически выполняющийся на АЛУ мультипроцессора, исполняет инструкции, описанные в ядре. При этом, благодаря SIMD-архитектуре, на каждом мультипроцессоре несколько потоков параллельно выполняют одну и ту же последовательность инструкций. Логически потоки объединяются в блоки, ограничивающие возможность обмена данными между потоками. Потоки могут обмениваться данными через общую память только внутри одного блока. Кроме того, потоки из одного блока выполняются на одном и том же мультипроцессоре.

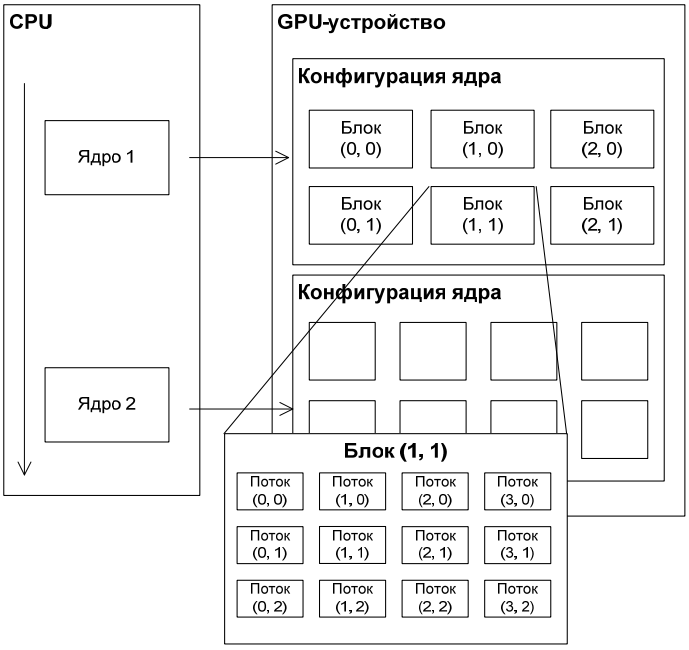

Рис. 1. Иерархия элементов конфигурации GPU-устройства

Таким образом, перед запуском ядра разработчик задает его конфигурацию (grid): размер блока и общее число блоков. Для более удобного структурирования конкретных задач в CUDA предусмотрена возможность конфигурирования блоков и потоков в двумерные сетки. Данная возможность обеспечивает удобство при обработке данных, имеющих двумерную структуру: изображений, матриц и пр.

Ключевым моментом отображения вычислительного алгоритма на GPU-архитектуру является определение ядра, соответствующего решению поставленной задачи. При этом следует учитывать, что каждый из потоков, исполняющих код ядра, для получения входных данных должен обращаться к своей области памяти устройства. На рис. 2 представлена общая схема преобразования последовательного циклического алгоритма в его GPU-реализацию.

При исполнении на CPU-системе алгоритм многократно последовательно вычисляет некоторую функцию $f(X)$ на различных наборах данных X_i. При исполнении на GPU указанная функция преобразуется к виду $f'(X)$, удобному для исполнения в рамках одного потока и затем вычисляется на GPU параллельно, на разных наборах данных. Разумеется, данный подход является эффективным лишь в том случае, когда итерации цикла, вычисляющего $f(X)$ независимы.

Заметим также, что при описании вычислительного ядра необходимо учитывать ряд важных особенностей работы GPU-устройств. Во-первых, SIMD-архитектура накладывает ограничения на использование ветвящихся конструкций (if/else). Это связано с тем, что потоки, исполняющиеся на одном и том же мультипроцессоре, должны оперировать одной и той же последовательностью инструкций, что может нарушаться при несовпадении условий ветвления у разных потоков. При нарушении этого ограничения может происходить сильное ухудшение производительности. Во-вторых, следует учитывать особенности работы GPU-устройств с памятью: определение ядра таким образом, чтобы одновременно выполняющиеся потоки читали данные из соседних участков памяти, существенно увеличивает производительность. Кроме того, необходимо учитывать трудоемкость обращения к памяти GPU-устройства: например, чтение из памяти для акселераторов nVidia занимает 400-600 тактов, тогда как операция сложения занимает всего 4 такта. В-третьих, следует учитывать архитектуру конкретного GPU-устройства, на котором выполняется вычислительная задача. Например, неэффективно задавать конфигурацию ядра с числом блоков меньшим числа мультипроцессоров, имеющимся на устройстве. Более подробно эти и остальные особенности GPU-архитектуры рассмотрены в [13].

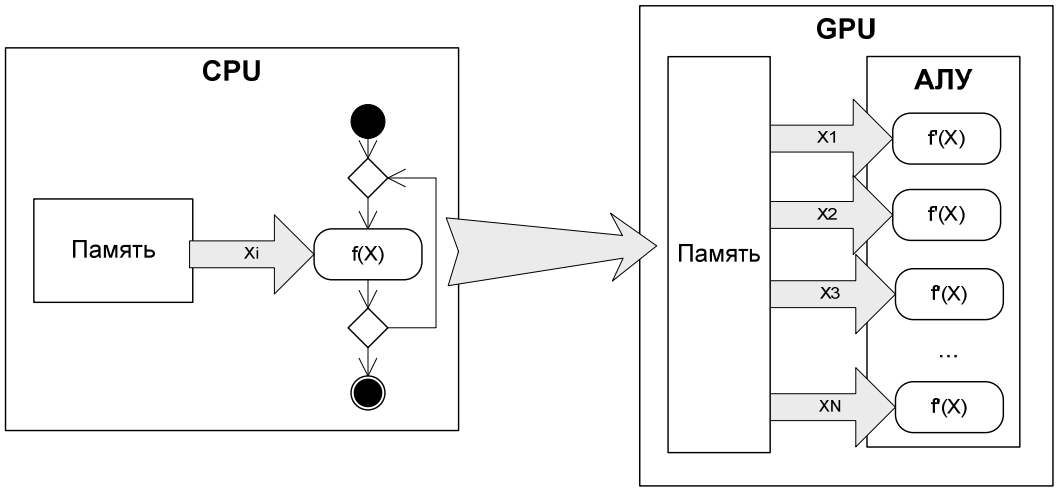

Рис. 2. Схема отображения вычислительного алгоритма на GPU-устройство

3. Отображение алгоритма параметрической оптимизации методом случайного поиска на архитектуру GPU-устройства

В качестве примера вычислительного алгоритма, отображаемой на GPU-архитектуру, в данной работе была выбрана задача параметрической аппроксимации двумерной функции с использованием метода случайного поиска [15].

В процессе работы данного алгоритма оптимизации производится случайный выбор направления в пространстве поиска, $V \in R^n$. В данной работе используется вариация метода адаптивного случайного поиска с линейной тактикой:

$$V_n = V_{n-1} + \Delta V_n \xi^{(j)}, \tag{1}$$

где $\xi^{(j)}$ - единичный орт в случайном направлении j. В этом случае шаг ΔV_n в случайно выбранном направлении повторяется до тех пор, пока он не перестанет приводить к уменьшению значения целевой функции $J(V)$:

$$\Delta V_n = \begin{cases} a\xi, \text{ если } \Delta J_{n-1} \geq 0, \\ \Delta V_{n-1}, \text{ если } \Delta J_{n-1} < 0. \end{cases} \tag{2}$$

Здесь a - начальный размер шага для нового направления. Для изменения длины шага в процессе работы алгоритма используется параметр релаксации. Решение об изменении шага принимается в зависимости от успешности предыдущей операции:

$$|\Delta V_n| = \begin{cases} \gamma_1 |\Delta V_{n-1}|, \text{ в случае успеха,} \\ \gamma_2 |\Delta V_{n-1}|, \text{ в противном случае.} \end{cases} \quad (3)$$

Для определения параметров релаксации используется следующее соотношение:

$$\gamma_1^p \gamma_2^{(1-p)} = 1. \quad (4)$$

Данный алгоритм имеет широкое применение, в частности, при решении задачи аппроксимации спектров морского волнения [16]. Одной из особенностей этой задачи является необходимость обработки больших массивов входных данных, что с одной стороны, приводит к большой трудоемкости вычислений, а с другой, позволяет эффективно использовать распараллеливание по данным. Это позволяет эффективно использовать вычислительные мощности GPU-устройств, поскольку для каждого набора входных данных решается одна и та же задача.

Другим вариантом использования GPU является параллельная реализация расчета целевой функции $J(V)$. Вычислительные мощности GPU используются для подсчета значений целевой функции для набора входных данных, обрабатываемого на одном этапе работы алгоритма оптимизации N спектров, выполняющемся на CPU. Схемы параллельной реализации для обоих случаев приведены на рис. 3.

(а) (б)

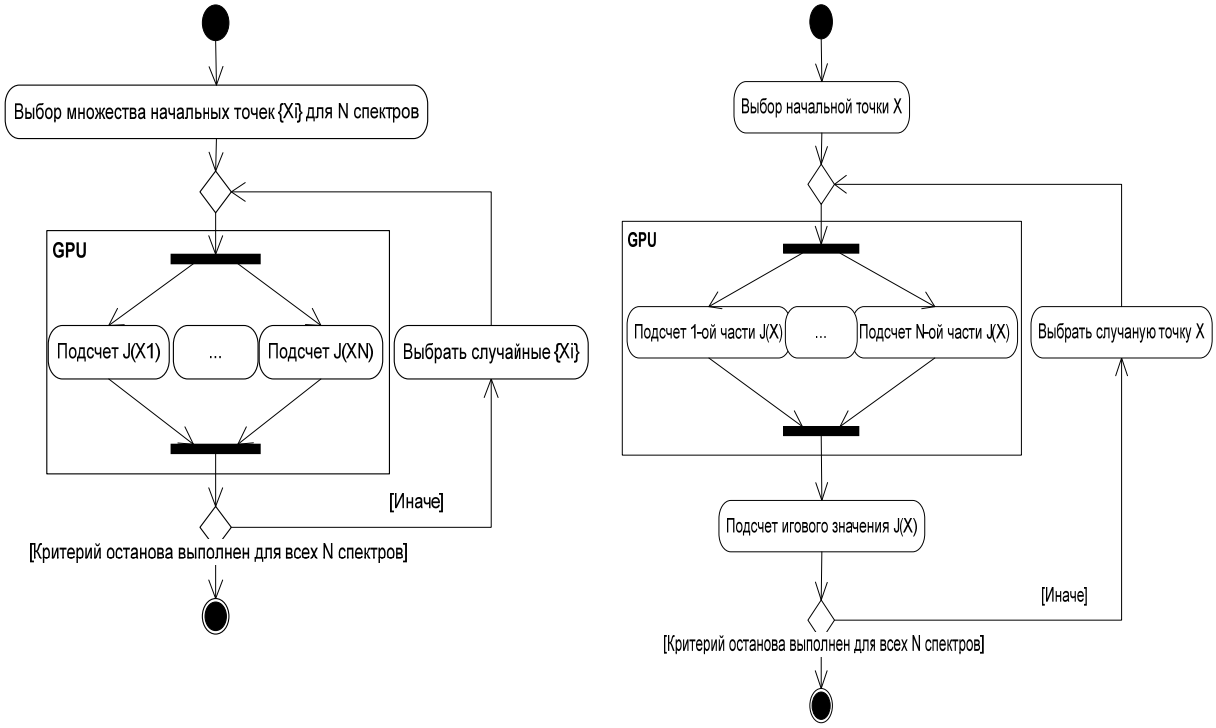

Рис. 3. Схемы параллельных алгоритмов для GPU: (а) распараллеливание по данным; (б) распараллеливание вычисления целевой функции

4. Анализ параллельной производительности

Анализ параллельной производительности проведен для алгоритма, основанного на декомпозиции по данным (рис. 3а), при котором в каждом потоке выполняется подсчет значения целевой функции для своего набора входных данных. Предметом исследования являлось влияние конфигурации системы на получаемое ускорение, а так же оценка составных частей времени работы алгоритма. Кроме того был исследован эффект применения различных типов оптимизаций вычислительного ядра. Для оценки ускорения время работы с использованием GPU-

устройства nVidia GeForce 8800 GTX сравнивалось со временем обработки тех же данных на CPU Intel Core 2 Duo 2.3 GHz.

На рис. 4 представлена зависимость получаемого ускорения от конфигурации GPU-устройства (число потоков в блоке, умноженное на число блоков) ядра.

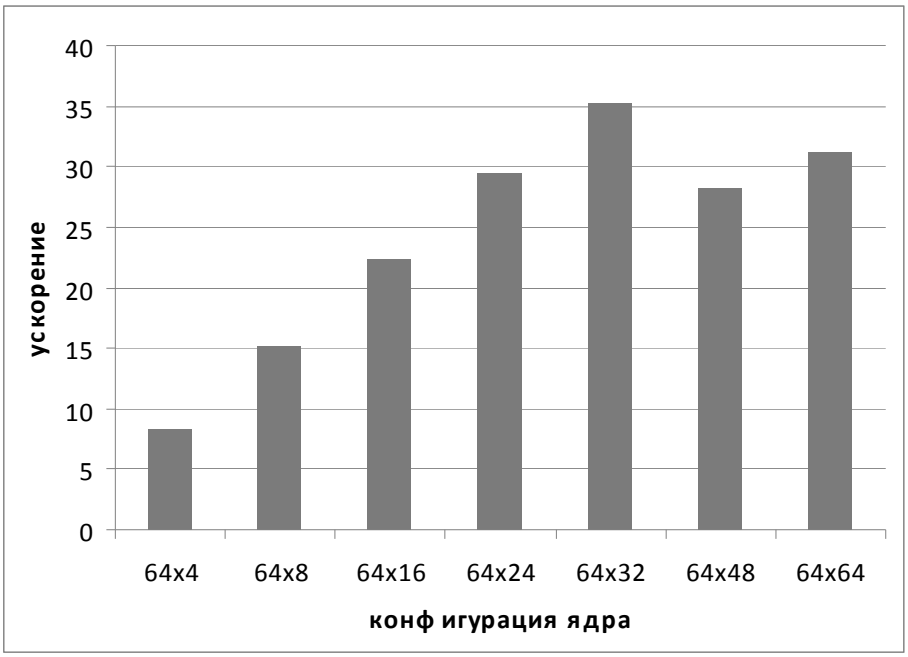

Рис.4. Зависимость ускорения от конфигурации ядра

Рост ускорения при увеличении числа блоков от 4 до 16 объясняется тем, что в исследуемой системе 16 мультипроцессоров. Таким образом, при конфигурации в 4 и 8 блоков часть мультипроцессоров остается бездействующей. При дальнейшем увеличении числа блоков все мультипроцессоры загружены работой, тем не менее, ускорение продолжает расти. Этот факт объясняется особенностью обращения мультипроцессоров к памяти: если на одном мультипроцессоре, в соответствии с конфигурацией, выполняется одновременно несколько блоков, то, в то время как процессы одного блока ждут данные из памяти, на мультипроцессоре могут выполняться арифметические операции других блоков. Схема на рис. 5 поясняет данное утверждение: первоначально при увеличении числа блоков (а, значит, и количества входных данных) общее время работы практически не меняется.

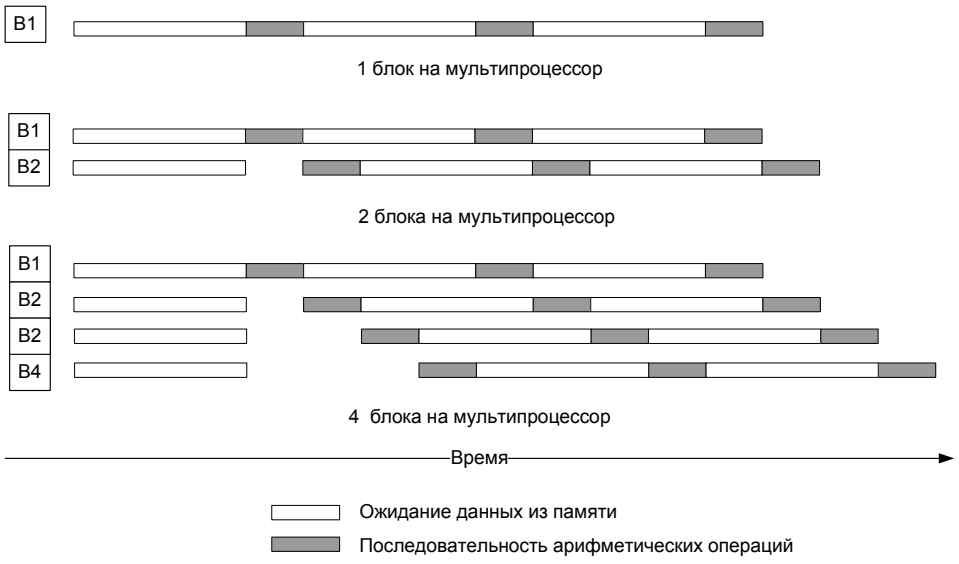

Рис. 5. Одновременное выполнение блоков на мультипроцессоре GPU-устройства.

В тот момент, когда число блоков достигает некоторого критического значения, определяемого соотношением числа арифметических операции и обращений к памяти внутри ядра, происходит резкое увеличение времени работы (в примере, изображенном на рис.8 такое «насыщение» произойдет при увеличении конфигурации ядра до 5 блоков). Этот факт объясняет дальнейший спад ускорения при увеличении числа блоков до 48 и 64 (рис. 4).

На рис. 6 представлены результаты, полученные в процессе измерения составных частей времени работы алгоритма на GPU-устройстве:

$$T = T_{malloc} + T_{memcpyin} + T_{\ker nel} + T_{memcpyout} + T_{free} \tag{5}$$

которое состоит не только из времени выполнения ядра $T_{\ker nel}$, но и времени, затрачиваемого на выделение T_{malloc} и освобождение T_{free} памяти на устройстве и копирования данных $T_{memcpyin}$, $T_{memcpyout}$. Время исполнения ядра практически не меняется с увеличением числа блоков от 4 до 32 – сначала из-за неполной загруженности устройства, затем – из-за описанного выше эффекта. В то время как при увеличении числа блоков до 48 происходит резкий скачок. Время, затрачиваемое на работу с памятью, растет при этом линейно (так как с увеличение числа блоков увеличивается и количество входных и выходных данных). Заметим, что это время в данном случае составляет существенную часть от общего времени. Таким образом, можно сделать вывод о том, что в задачах, использующих повторяющиеся наборы данных, следует избегать повторного копирования и выделения памяти там, где это возможно.

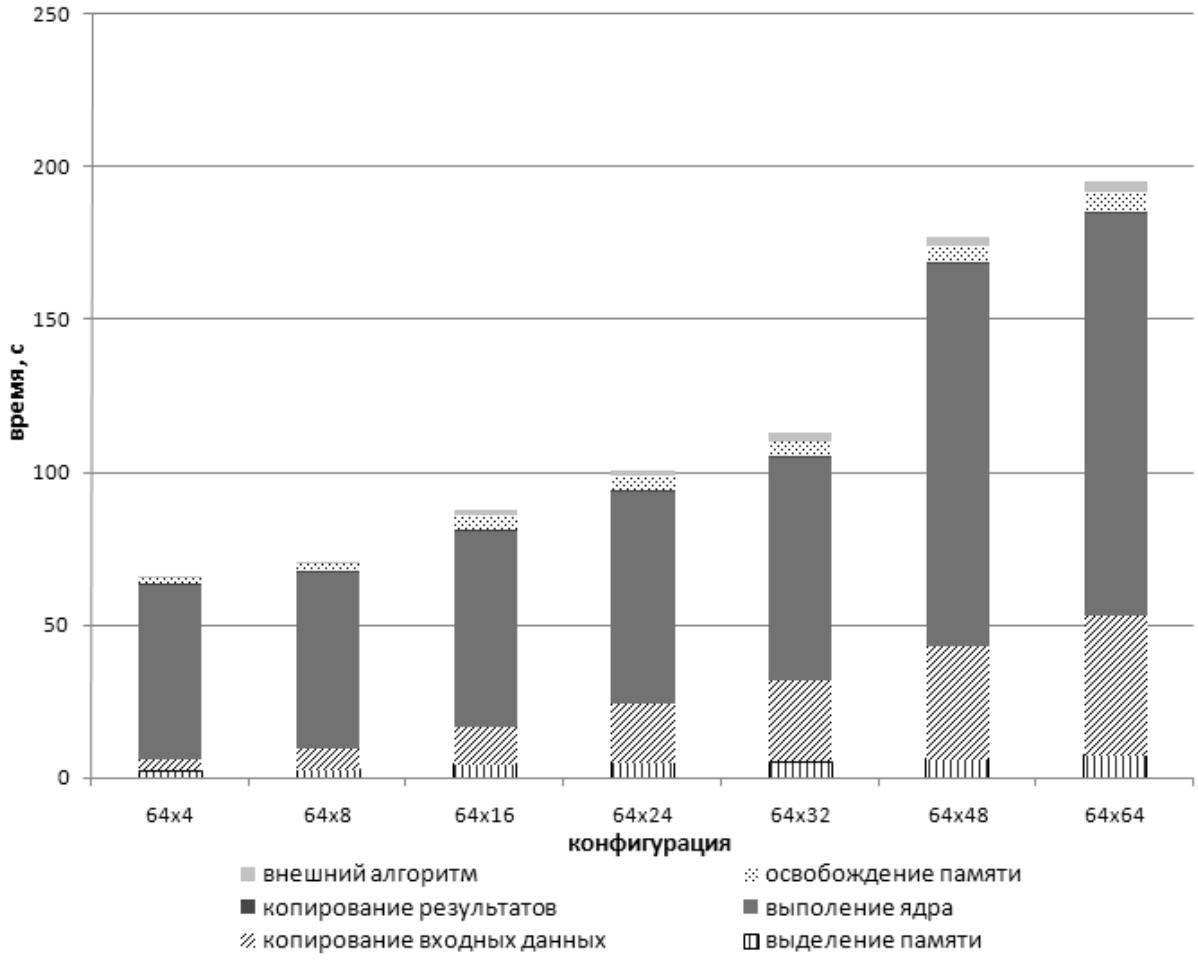

Рис. 6. Составные части времени работы

5. Заключение

Результаты вычислительных экспериментов на задаче аппроксимации климатических спектрах морского волнения показывают, что адаптация алгоритма параметрической оптимизации к архитектуре GPU позволяет получить ускорение более 30 раз по сравнению с реализацией без использования GPU. Учитывая, что стоимость простейшей кластерной системы на основе стандартных комплектующих, обеспечивающих такое же ускорение, примерно в 30 раз превышает стоимость рассматриваемого GPU-устройства, это подтверждает целесообразность применения GPU-устройств при решении вычислительных задач подобного класса. В планах дальнейших исследований – выработка методов отображения других вычислительных алгоритмов на архитектуру GPU-устройств.

Литература

1. Hasle G., Lie K.-A., Quak E. Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF — Springer, 2007. — 558 p.
2. General-Purpose Computation Using Graphics Hardware: [http://gpgpu.org/]
3. Purcell T.J., Donner C., Commarano M., Jensen H.W., Hanrahan P. Photon mapping on programmable graphic hardware // Proceeding of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware — Eurographics Association, 2003. — pp. 41-50.
4. Göddeke M., Strzodka R., Turek S. Accelerating Double Precision FEM Simulations with GPUs // Proceeding of ASIM 2005 — 18th Symposium on Simulation Technique — 2005. — pp. 139-144.
5. Hagen T.R., Henriksen M.O., Hjelmervik J.M., Lie K.-A. Using the graphic processor as a high-performance computational engine for solving system of hyperbolic conservation low // Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF — Springer, 2007, pp. 211-264.
6. Hagen T.R., Hjelmervik J.M., Lie K.-A., Natvig J.R., Henriksen M.O. Visual simulation of shallow-water waves // Simulation Practice and Theory. Special Issue on Programmable Graphics Hardware, 13(9) — 2005. — pp. 716-726.
7. Hagen T.R., Lie K.-A., Natvig J.R. Solving the Euler equation on graphical processing units // Computational Science — ICCS 2006: 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part IV, volume 3994 of Lecture Notes in Computational Science (LNCS) — Springer Verlag, 2006. — pp. 220-227.
8. OpenGL - The Industry Standard for High Performance Graphics: [http://www.opengl.org/]
9. Microsoft's DirectX site: [http://www.microsoft.com/directx]
10. Fernando R., Kilgard M.J. The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics — Adisson-Wesley Longman Publishing Co., 2003.
11. GPUBench : How much does your GPU bench: [http://graphics.stanford.edu/projects/gpubench/]
12. Rost R.J. OpenGL Shading Language — Adisson-Wesley Longman Publishing Co., 2004.
13. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. Ver 1.0. June 2007 — NVIDIA Corporation, 2007.
14. Адинец А.В., Сахарных Н.А. О программировании вычислений общего назначения на графических процессорах // Научный сервис в сети Интернет: многоядерный компьютерный мир. 15 лет РФФИ: Труды Всероссийской научной конференции (24-29 сентября 2007 г., г. Новороссийск) — М.: Издательство МГУ, 2007. — с. 249-256
15. Растригин Л.А. Адаптация сложных систем — Рига: Зинатне, 1981. 375 с.
16. Boukhanovsky A.V., Lopatoukhin L.J., Guedes Soares C. Spectral wave climate of the North Sea — Applied Ocean Research, 2007.