Д.Б. Бражкин

Нижегородский государственный университет им. Н.И. Лобачевского

РЕШЕНИЕ ЗАДАЧ ГЛОБАЛЬНОЙ ОПТИМИЗАЦИИ НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ

В настоящее время широкое распространение получили вычисления общего назначения на графических картах (General-Purpose Computing on Graphics Processing Units, GPGPU). Статистика показывает, что производительность современных видеокарт растет значительно более высокими темпами, чем производительность центральных процессоров.

NVidia CUDA (Compute Unified Device Architecture) [1] представляет собой программно-аппаратное решение, предназначенное для осуществления вычислений на GPU как на обычном вычислительном устройстве, без необходимости использования графических API. CUDA фактически представляет собой расширение языка *C*. Каждая новая версия технологии позволяет использовать более широкий спектр возможностей: снимаются некоторые ограничения, повышается общая производительность, появляются новые функции.

Высокая пиковая мощность графических карт (в сравнении с CPU) позволяет использовать их при решении многих практических задач, в том числе таких трудоемких, как поиск глобального оптимума многомерных функций.

Постановка задачи. Рассматривается конечномерная задача глобальной оптимизации (задача нелинейного программирования) вида:

$$f(y) \to \min, y \in Y,$$

$$Y = \left\{ y \in D : g_j(y) \le 0, 1 \le j \le m \right\},$$

$$D = \left\{ y \in R^N : y_i \in \left[a_i, b_i \right], 1 \le i \le N \right\},$$

т.е. задача отыскания экстремальных значений целевой (минимизируемой) функции на множестве, задаваемой координатными и функциональными ограничениями на выбор допустимых точек (векторов) $y = (y_1, y_2, ...)$. В данной модели величины a_i, b_i, $1 \le i \le N$ (границы измерения варьируемых параметров задачи – координаты вектора y) являются заранее заданными константами.

Предполагается, что целевая функция $f(y)$, а также левые части ограничений $g_j(y)$ удовлетворяют условию Липшица с соответствующими (возможно, неизвестными) константами: L, $L_j (1 \le j)$, т.е.

$$|f(y_1) - f(y_2)| \le L \|y_1 - y_2\|, y_1, y_2 \in D,$$

$$|g_j(y_1) - g_j(y_2)| \le L_j \|y_1 - y_2\|, 1 \le j \le m, y_1, y_2 \in D.$$

Применение развёрток в задачах с ограничениями. Рассмотрим кривую Пеано $y(x), x \in [0,1]$, отображающую однозначно и непрерывно отрезок вещественной оси на многомерный гиперпараллелепипед,

$$D = \{y \in R^N : y_i \in [a_i, b_i]\}, 1 \le i \le N,$$

т.е.

$$D = \{y(x) : 0 \le x \le 1\}.$$

Непрерывность целевой функции $f(x)$ исходной многомерной задачи обеспечивает соотношение $\min_{y \in D} f(x) = \min_{x \in [0,1]} f(y(x))$, и возможность вместо многомерной задачи решать одномерную задачу $f(y(x)) \to \min, x \in [0,1]$.

Свойства различных вариантов разверток, а также алгоритмы их построения подробно рассмотрены в [3, 5]. В данной работе реализована кусочно-линейная пеаноподобная развертка. Достоинством данной схемы является ее простота, однако развертка такого типа обладает некоторыми недостатками, для их устранения возможно применение других схем – неинъективной или множественной развертки.

Для решения полученной одномерной задачи используется индексный метод. Подробное описание вычислительной схемы метода рассмотрено в [3].

Программная реализация. По сравнению с предыдущей версией приложения, рассматриваемой в [2], структура претерпела значительные изменения. Теперь графический процессор полностью выполняет всю математическую составляющую алгоритма, а центральный процессор лишь инициализирует данные и отвечает за интерфейс приложения. Общая схема приложения представлена на рис. 1.

Параллельная схема реализации. Распараллеливание алгоритма связано со спецификой устройства графического процессора и представления его как вычислительного устройства при использовании технологии NVidia CUDA. При организации вычислений GPU представляется в виде набора нескольких мультипроцессоров, каждый из которых обладает собственной «быстрой» памятью и способен обрабатывать данные в несколько потоков [1].

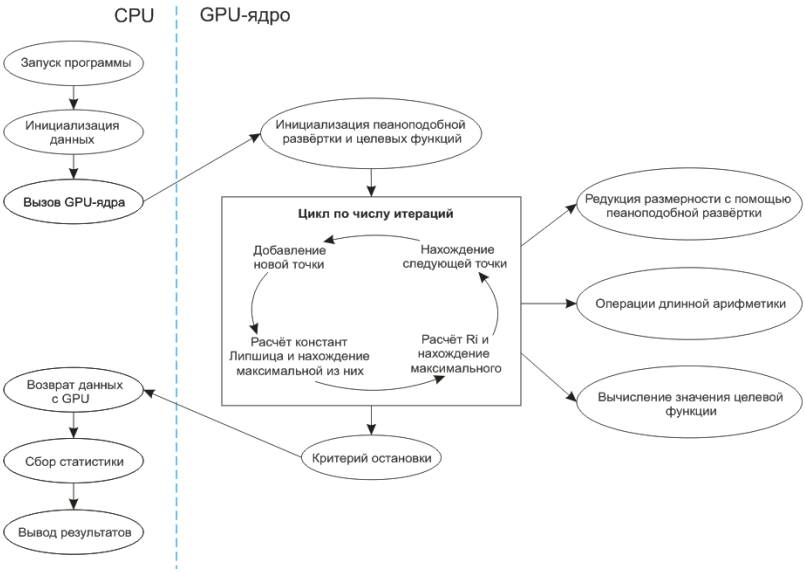


Рис. 1. Общая схема приложения

Для распараллеливания любого алгоритма на GPU удобнее всего разделить обрабатываемые данные между потоками таким образом, чтобы взаимодействие между ними сводилось к минимуму. Это связано с архитектурой GPU, а именно с низкой скоростью обмена данными между вычислительными блоками. В данном случае целесообразно разделить между потоками массив точек предшествующих испытаний. Наиболее трудоёмкими шагами алгоритма являются нахождение констант Липшица, а также вычисление характеристик интервалов. Распараллеливание на этих этапах реализовано схожим образом. Рассмотрим в качестве примера вычисление характеристик интервалов.

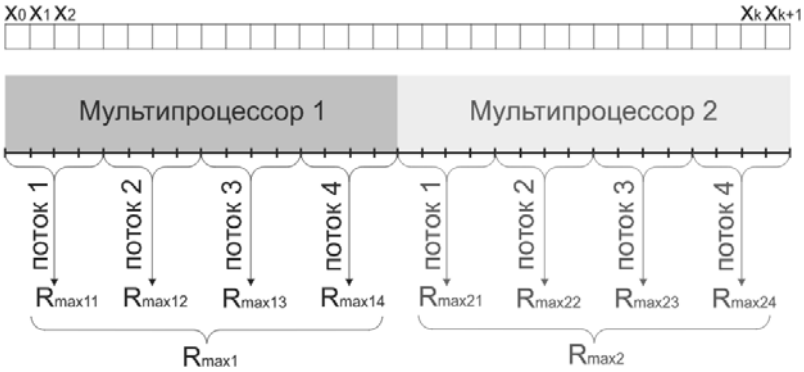


Рис. 2. Схема распараллеливания вычисления характеристик

Примем номер мультипроцессора за *bid*, $0 \le bid < B$, где B – количество мультипроцессоров, а номер потока в нём – за *tid*, $0 \le tid < T$, где T – количество таких потоков. На каждой итерации алгоритма массив точек предшествующих испытаний делится между мультипроцессорами на равные группы. Число точек в массиве равно $k+2$, таким образом, мультипроцессор с номером *bid* получает на вход часть массива с индексами (рис. 2)

$$\left(\frac{bid}{B}(k+1), \frac{bid+1}{B}(k+1) \right).$$

Они, в свою очередь, также поровну разделяются между потоками в мультипроцессоре, на промежутки

$$\left(\frac{bid + \frac{tid}{T}}{B}(k+1), \frac{bid + \frac{tid+1}{T}}{B}(k+1) \right).$$

Каждый поток обрабатывает свою часть отрезка, после чего происходит сбор данных и переход к следующему шагу алгоритма.

Стоит также отметить, что периодически требуется синхронизировать все потоки графического ядра. Технология NVidia CUDA позволяет синхронизировать лишь потоки в пределах одного блока потоков. Однако использование некоторых особенностей технологии позволило реализовать эффективную глобальную синхронизацию потоков.

Использование чисел расширенной точности. При реализации многомерного алгоритма глобальной оптимизации возникает следующая проблема: для достижения точности $\varepsilon = (1/2)^m$ в R^N на отрезке [0, 1] требуется точность $(1/2)^{m \cdot N}$, где m – уровень построения развёртки, а N – размерность пространства. При использовании типа *float* максимально возможная точность ограничена условием $Nm < 23$. Использование типа *double* позволяет расширить этот диапазон до $Nm < 52$, однако в силу особенностей архитектуры GPU значительно снижается производительность. На момент создания приложения никакой информации о сторонних библиотеках для работы с числами расширенной точности для GPU не было. Единственным выходом из сложившейся ситуации являлось создание собственной библиотеки.

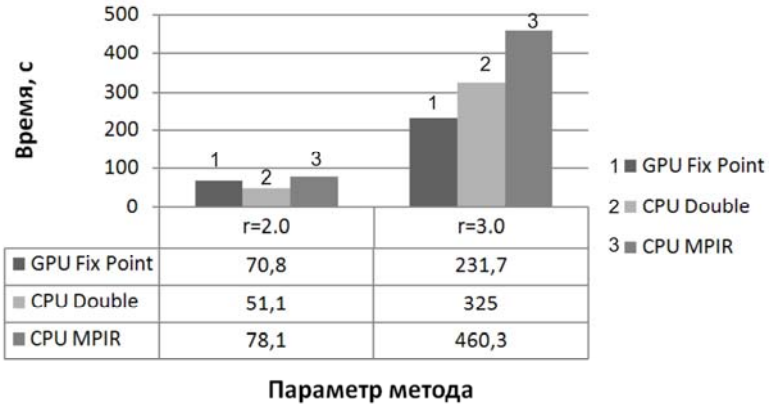
Из операций с числами расширенной точности требуется сложение, вычитание, сравнение, обнуление, битовый сдвиг, а также приведение типов. В качестве базового типа для создания библиотеки использовался 32-битный *int*. Это позволило использовать операции битового сдвига, выполняемые на GPU достаточно быстро [4].

В силу специфики задачи «длинные» числа расположены в интервале (0,1), для их хранения и обработки достаточно реализации чисел с фиксированной точкой, вычисления с которой выполняются быстрее аналогичных вычислений с плавающей точкой.

Результаты вычислительных экспериментов. Для тестирования полученной реализации был использован набор функций Растригина, имеющих вид $f(x) = nA + \sum_{i=1}^{n}(x_i^2 - A\cos(2\pi x_i))$; глобальный оптимум этой функции равен 0 при $x_i = 0$.

Эксперименты проводились с тремя реализациями алгоритма – две версии работают на CPU и одна версия на GPU. Тестовая конфигурация: Asus P5Q-E P45, Core 2 Duo E8500@3,8GHz, 2 Gb RAM, NVidia 9800 GT, MS Windows 7, NVidia CUDA 3.0.

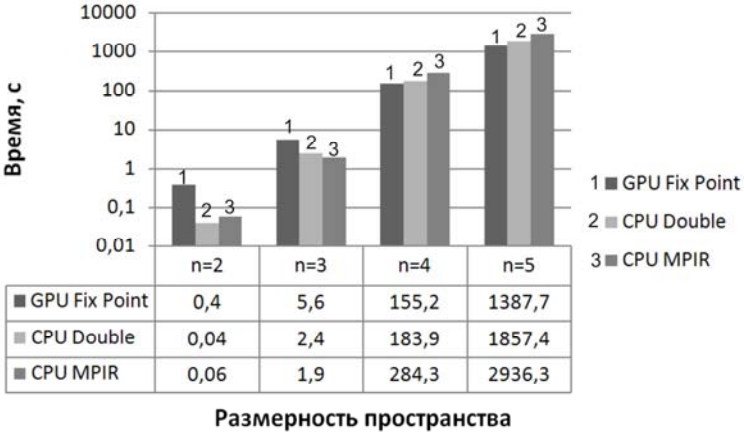
Первый график (рис. 3) показывает время работы различных реализаций алгоритма в зависимости от параметра индексного метода.



	r=2.0	r=3.0
GPU Fix Point	70,8	231,7
CPU Double	51,1	325
CPU MPIR	78,1	460,3

Рис. 3. Время работы реализаций в зависимости от параметра метода

Второй эксперимент (рис. 4) – тестирование алгоритма при различной размерности решаемой задачи. Как видно из первых двух графиков, GPU-версия с ростом сложности задачи показывает лучшие результаты, чем версии на CPU.



	n=2	n=3	n=4	n=5
GPU Fix Point	0,4	5,6	155,2	1387,7
CPU Double	0,04	2,4	183,9	1857,4
CPU MPIR	0,06	1,9	284,3	2936,3

Рис. 4. Время работы реализаций в зависимости от размерности задачи

Третий эксперимент показывает масштабируемость GPU-реализации по числу используемых вычислительных блоков. Вычислительный блок на GPU представляет собой группу из 8 ядер. То есть если видеокарта содержит 112 ядер, то эти ядра сгруппированы в 14 блоков. В связи с этим эксперимент проводился с подключением 1, 2, 4, 8 и всех 14 вычислительных блоков. Результаты представлены на рис. 5. График показывает практически линейный рост производительности. Это объясняется тем, что сложность решаемой задачи высока и, как следствие, основная нагрузка ложится на вычислительные блоки, а не на шину данных.

В данной работе рассмотрена полная GPU-реализация индексного метода глобальной оптимизации. Проведённые эксперименты показали, что на сложных вычислительных задачах графический процессор показывает отличную производительность и его использование в качестве вычислительного устройства себя оправдывает.

Наибольшую трудность вызывает грамотное использование вычислительных возможностей GPU. Всё-таки в первую очередь это устройство для обработки и вывода графической информации, обладающее, однако, огромным вычислительным

потенциалом. С появлением новых средств разработки эта задача значительно упрощается, и если несколько лет назад использование GPU в качестве средства вычисления было лишь исследовательской задачей для энтузиастов, то теперь подобные возможности находят применение даже в коммерческих продуктах.

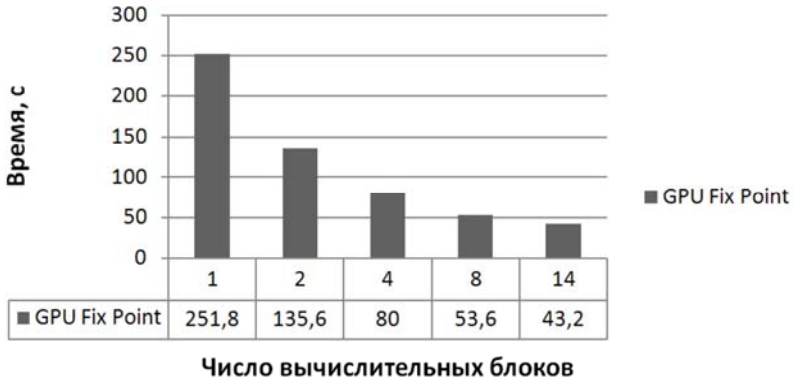


Рис. 5. Масштабируемость GPU-реализации

В перспективе видится несколько возможностей для развития проекта: это и применение других типов развёрток, и дальнейшая оптимизация кода, возможно использование новых методов и технологий.

Список литературы

1. NVidia CUDA C Programming Guide. – URL: http:// developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/ NVIDIA_CUDA_ProgrammingGuide.pdf.

2. Бастраков С.И., Бражкин Д.Б.. Решение задач глобальной оптимизации с использованием смешанных CPU-GPU вычислений // Технологии Microsoft в теории и практике программирования: материалы конф. – Н. Новгород: Изд-во Нижегород. гос. ун-та, 2009. – С. 23–29.

3. Городецкий С.Ю., Гришагин В.А. Нелинейное программирование и многоэкстремальная оптимизация: учеб. пособие. – Н. Новгород: Изд-во Нижегород. гос. ун-та, 2007.

4. Кнут Д. Искусство программирования. Т. 2. Получисленные алгоритмы. – М.: Вильямс, Addison Wesley Longman, 2000.

5. Стронгин Р.Г. Численные методы многоэкстремальной оптимизации (информационно-статистические алгоритмы). – М.: Наука, 1978.

Л.Н. Бутымова, В.Я. Модорский,
А.Ф. Шмаков, Д.Ф. Гайнутдинова

Пермский государственный технический университет

АНАЛИЗ КОЛЕБАТЕЛЬНЫХ РЕЖИМОВ АКУСТИЧЕСКОГО ТРАНСФОРМАТОРА

В целом ряде машиностроительных конструкций имеются каналы, заполненные жидкостью. Требуется создание методик, позволяющих прогнозировать поведение в динамической системе «жидкость–конструкция» при внешнем колебательном воздействии. Кроме того, авторам неизвестны модели, объясняющие эффект направленного движения жидкости в каналах упругих конструкций при их периодическом изгибе.

В полной постановке требуются моделирование источника колебаний, оптимизация его конструкции, обеспечение эффективности технологического процесса, моделирование самого технического объекта, что предполагает совместное решение гидродинамической задачи и задачи теории упругости (рис. 1).

В Центре высокопроизводительных вычислительных систем имеется необходимое программное обеспечение и оборудование для решения такого класса задач. В частности, Flow Vision, ABAQUS, ANSYS CFD, ANSYS CFX, STAR CD, LS DYNA, ANSYS WorkBench.