УДК 519.853.4

И.Г. Лебедев, К.А. Баркалов

Нижегородский государственный университет
им. Н.И. Лобачевского, Нижний Новгород, Россия

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНОГО АЛГОРИТМА ГЛОБАЛЬНОГО ПОИСКА НА GPU

Рассматриваются параллельные алгоритмы решения задач многоэкстремальной оптимизации. Целевая функция удовлетворяет условию Липшица с неизвестной априори константой. Подобные задачи широко распространены в приложениях. Описываемый алгоритм использует подход, основанный на идее редукции размерности с помощью кривой Пеано, непрерывно и однозначно отображающей отрезок вещественной оси на *N*-мерный куб.

Одним из перспективных направлений в области параллельной глобальной оптимизации является использование графических ускорителей (GPU), востребованных в решении целого ряда вычислительно трудоемких задач. Однако в области глобальной оптимизации потенциал графических ускорителей до конца еще не раскрыт. С использованием GPU в основном распараллеливают алгоритмы, которые так или иначе основаны на идее случайного поиска. В силу своей вероятностной природы алгоритмы подобного типа, вообще говоря, не гарантируют сходимость к глобально-оптимальному решению, что невыгодно отличает их от детерминированных методов.

Для многих детерминированных алгоритмов липшицевой глобальной оптимизации с гарантированной сходимостью предложены их параллельные варианты. Однако указанные версии алгоритмов распараллелены на CPU с использованием технологий MPI и/или OpenMP, реализации на GPU отсутствуют. В настоящей статье приведены результаты исследования GPU-реализации параллельного алгоритма глобального поиска, разработанного в рамках информационно-статистического подхода.

С целью экспериментального подтверждения теоретических свойств рассматриваемого параллельного алгоритма проведены вычислительные эксперименты на серии из нескольких сотен тестовых задач разной размерности. При этом результаты работы последовательного алгоритма показывают его превосходство над другими известными методами глобальной оптимизации. Одновременно параллельный алгоритм показывает хорошее ускорение (как GPU-, так и CPU-версии), а GPU-версия параллельного алгоритма показывает ускорение от 2 до 6 раз по сравнению с наиболее быстрой параллельной CPU-версией.

Ключевые слова: глобальная оптимизация, многоэкстремальные функции, редукция размерности, характеристические алгоритмы, параллельные алгоритмы, графические процессоры.

I.G. Lebedev, K.A. Barkalov

N.I. Lobachevsky State University of Nizhniy Novgorod,
Nizhniy Novgorod, Russian Federation

A GPU IMPLEMENTATION OF A PARALLEL GLOBAL SEARCH ALGORITHM

The present paper considers parallel algorithms for solution of problems of multiextremal optimization. The objective function satisfies a Lipschitz condition with a priori unknown constant *L*. Similar problems are widespread in applications. The algorithm uses an approach based on dimention reduction with Peano curve, which continuously and unambiguously maps a unit interval onto an *N*-dimensional cube.

One of the promising directions of development in the domain of parallel global optimization is use of graphics processing units (GPUs), required for solution of computationally complex problems. However, in the domain of global optimization the potential of GPUs is not completely unlocked. GPUs are generally used for parallelization of algorithms, which one way or another are based on the concept of random search Owing to their stochastic nature algorithms of this type, generally speaking, do not guarantee convergence to a globally optimal solution, and that sets them apart from deterministic methods.

For many deterministic algorithms of Lipschitzian global optimization with guaranteed convergence parallel variants are proposed. However, these versions of algorithms are parallelized on CPU using MPI and/or OpenMP technologies; there are no implementations on GPU. The present article contains results of study of GPU implementation of the global search parallel algorithm developed according to the information-statistical approach.

For the purpose of experimental demonstration of theoretical properties of the considered parallel algorithms computing experiments on a series of several hundred test problems of different dimension were performed. The results of work of the sequential global search algorithm show its superiority over other known methods of global optimization. At the same time the parallel algorithm shows a good speed up and low redundancy (both GPU and CPU version), and the GPU version of the parallel algorithm shows a speed up of 2 to 6 times in comparison with the fastest trial parallel CPU version.

Keywords: global optimization, multiextremal functions, dimension reduction, characteristical algorithms, parallel algorithms, graphics processing units.

Введение

В данной статье рассматриваются параллельные алгоритмы решения задач многоэкстремальной оптимизации. В многоэкстремальных задачах возможность достоверной оценки глобального оптимума принципиально основана на наличии априорной информации о функции, позволяющей связать возможные значения оптимизируемой функции с известными значениями в точках осуществленных поисковых испытаний. Весьма часто такая информация о решаемой задаче представляется в виде предположения, что целевая функция φ удовлетворяет условию Липшица с неизвестной априори константой L. При этом целевая функция может быть задана программно реализуемым алгоритмом вычисления ее значений в точках области поиска. Подоб-

ные задачи широко распространены в приложениях (задачи оптимального проектирования объектов и технологических процессов в различных технических отраслях, задачи идентификации моделей по данным наблюдений в научных исследованиях, оптимизация процессов управления и т.п. [1]).

Задачи многоэкстремальной оптимизации обладают узкой сферой возможностей аналитического исследования и высокой трудоемкостью численного решения, поскольку в последнем случае для них характерен экспоненциальный рост вычислительных затрат с ростом размерности. Использование современных параллельных вычислительных систем расширяет сферу применения методов глобальной оптимизации и в то же время ставит задачу эффективного распараллеливания процесса поиска. Именно поэтому разработка эффективных параллельных методов для численного решения задач многоэкстремальной оптимизации и создание на их основе программных средств для современных вычислительных систем является основным вектором развития в данной области.

Одним из перспективных направлений в области параллельной глобальной оптимизации (как, впрочем, и во многих областях, связанных с программной реализацией трудоемких алгоритмов) является использование графических ускорителей (GPU). На протяжении последнего десятилетия графические ускорители стремительно наращивали производительность, чтобы удовлетворять непрерывно растущие запросы разработчиков графических приложений. Кроме того, за последние несколько лет изменились некоторые принципы разработки графической аппаратуры, в результате чего она стала более программируемой. Сегодня графический ускоритель – это гибко программируемый массивно-параллельный процессор с высокой производительностью, востребованный в решении целого ряда вычислительно трудоемких задач [2].

Однако в области глобальной оптимизации потенциал графических ускорителей до конца еще не раскрыт. С использованием GPU в основном распараллеливают алгоритмы, которые так или иначе основаны на идее случайного поиска [3–6]); обзор данного направления GPU-алгоритмов оптимизации представлен в работе [8]. В силу своей вероятностной природы алгоритмы подобного типа, вообще говоря, не гарантируют сходимость к глобально-оптимальному решению, что невыгодно отличает их от детерминированных методов.

Для многих детерминированных алгоритмов липшицевой глобальной оптимизации с гарантированной сходимостью предложены их параллельные варианты [8–11]. Однако указанные версии алгоритмов распараллелены на CPU с использованием технологий MPI и/или OpenMP, реализации на GPU отсутствуют. В настоящей статье приведены результаты исследования GPU-реализации параллельного алгоритма глобального поиска, разработанного в рамках информационно-статистического подхода, представленного в монографии [12].

Многомерный алгоритм глобального поиска

Рассмотрим задачу поиска глобального минимума N-мерной функции $\varphi(y)$ в гиперинтервале $D = \{y \in R^N : a_i \le y_i \le b_i,\ 1 \le i \le N\}$. При этом будем предполагать, что функция удовлетворяет условию Липшица с априори неизвестной константой L:

$$\varphi(y^*) = \min\{\varphi(y) : y \in D\}, \tag{1}$$

$$\left|\varphi(y_1) - \varphi(y_2)\right| \le L\left\|y_1 - y_2\right\|,\ \ y_1, y_2 \in D,\ \ 0 < L < \infty. \tag{2}$$

В данной работе мы будем использовать подход, основанный на идее редукции размерности с помощью кривой Пеано $y(x)$, непрерывно и однозначно отображающей отрезок вещественной оси [0,1] на N-мерный куб:

$$\left\{y \in R^N : -2^{-1} \le y_i \le 2^{-1},\ 1 \le i \le N\right\} = \left\{y(x) :\ 0 \le x \le 1\right\}. \tag{3}$$

Вопросы численного построения отображений типа кривой Пеано и соответствующая теория подробно рассмотрены в работах [12, 13].

Использование подобного рода отображений позволяет свести многомерную задачу (1) к одномерной задаче

$$\varphi(y^*) = \varphi(y(x^*)) = \min\{\varphi(y(x)) : x \in [0,1]\}.$$

Важным свойством является сохранение ограниченности относительных разностей функции: если функция $\varphi(y)$ в области D удовлетворяла условию Липшица, то функция $\varphi(y(x))$ на интервале [0,1] будет удовлетворять равномерному условию Гельдера

$$\left|\varphi(y(x_1)) - \varphi(y(x_2))\right| \le H\left|x_1 - x_2\right|^{1/N},\ x_1, x_2 \in [0,1],$$

где константа Гельдера H связана с константой Липшица L соотношением

$$H = 4Ld\sqrt{N},\ d = \max\{b_i - a_i : 1 \le i \le N\}.$$

Исходя из этого, не ограничивая общности, можно рассматривать минимизацию одномерной функции $f(x) = \varphi(y(x))$, $x \in [0,1]$, удовлетворяющей условию Гельдера.

Рассматриваемый алгоритм решения данной задачи предполагает построение последовательности точек x^k, в которых вычисляются значения минимизируемой функции $z^k = f(x^k)$. Процесс вычисления значения функции (включающий в себя построение образа $y^k = y(x^k)$) будем называть испытанием, а пару (x^k, z^k) – результатом испытания. Множество пар $\{(x^k, z^k)\}$, $1 \le k \le n$ составляют поисковую информацию, накопленную методом после проведения n шагов. В нашем распоряжении имеется $P \ge 1$ вычислительных элементов, и в рамках одной итерации метода будем проводить P испытаний одновременно (синхронно). Обозначим $k(n)$ общее число испытаний, выполненных после n параллельных итераций.

На первой итерации метода испытание проводится в произвольной внутренней точке x^1 интервала [0,1]. Пусть выполнено $n \ge 1$ итераций метода, в процессе которых были проведены испытания в $k = k(n)$ точках x^i, $1 \le i \le k$. Тогда точки $x^{k+1}, ..., x^{k+p}$ поисковых испытаний следующей $(n + 1)$-й итерации определяются в соответствии с правилами:

Шаг 1. Перенумеровать точки множества $X_k = \{x^1, ..., x^k\} \cup \cup\{0\} \cup \{1\}$, которое включает в себя граничные точки интервала [0,1], а также точки предшествующих испытаний, нижними индексами в порядке увеличения значений координаты, т.е.

$$0 = x_0 < x_1 < ... < x_{k+1} = 1.$$

Шаг 2. Полагая $z_i = f(x_i), 1 \le i \le k,$ вычислить величины

$$\mu = \max_{1 \le i \le k} \frac{|z_i - z_{i-1}|}{\Delta_i},\ M = \begin{cases} r\mu, \mu > 0, \\ 1, \mu = 0, \end{cases}$$

где $r > 1$ является заданным параметром метода, а $\Delta_i = (x_i - x_{i-1})^{1/N}$.

Шаг 3. Для каждого интервала (x_{i-1}, x_i), $1 \le i \le k+1$, вычислить характеристику в соответствии с формулами

$$R(1) = 2\Delta_1 - 4\frac{z_1}{M}, \quad R(k+1) = 2\Delta_{k+1} - 4\frac{z_k}{M},$$

$$R(i) = \Delta_i + \frac{(z_i - z_{i-1})^2}{M^2 \Delta_i} - 2\frac{z_i + z_{i-1}}{M}, \quad 1 < i < k+1.$$

Шаг 4. Характеристики $R(i)$, $1 \le i \le k+1$, упорядочить в порядке убывания:

$$R(t_1) \ge R(t_2) \ge \ldots \ge R(t_{k-1}) \ge R(t_{k+1}),$$

и выбрать P наибольших характеристик с номерами интервалов t_j, $1 \le j \le P$.

Шаг 5. Провести новые испытания в точках x^{k+j}, $1 \le j \le P$, вычисленных по формулам

$$x^{k+j} = \frac{x_{t_j} + x_{t_j - 1}}{2}, \quad t_j = 1, \; t_j = k+1,$$

$$x^{k+1} = \frac{x_{t_j} + x_{t_j - 1}}{2} - \operatorname{sign}(z_{t_j} - z_{t_j - 1}) \frac{1}{2r} \left[\frac{|z_{t_j} - z_{t_j - 1}|}{\mu} \right]^N, \quad 1 < t_j < k+1.$$

Алгоритм прекращает работу, если выполняется условие $\Delta_{t_j} \le \varepsilon$ хотя бы для одного номера t_j, $1 \le j \le P$; здесь $\varepsilon > 0$ есть заданная точность. В качестве оценки глобально-оптимального решения задачи (1) выбираются значения

$$f_k^* = \min_{1 \le i \le k} f(x^i), \; x_k^* = \arg, \quad \min_{1 \le i \le k} f(x^i).$$

Обоснование данного способа организации параллельных вычислений описано в работах [12, 14]. Модификации, учитывающие наличие ограничений-неравенств в задаче, а также информацию о производной целевой функции, представлены в работах [15–17].

Проиллюстрируем работу алгоритма с использованием редукции размерности при минимизации двумерной многоэкстремальной функции вида

$$\varphi(y) = -\left\{ \left(\sum_{i=1}^{7}\sum_{j=1}^{7} A_{ij} g_{ij}(y) + B_{ij} h_{ij}(y) \right)^2 + \left(\sum_{i=1}^{7}\sum_{j=1}^{7} C_{ij} g_{ij}(y) - D_{ij} h_{ij}(y) \right)^2 \right\}^{1/2},$$

где

$$y = (y_1, y_2) \in R^2,\ 0 \le y_s \le 1,\ s = 1, 2,$$
$$g_{ij}(y) = \sin(i\pi y_1)\sin(j\pi y_2),$$
$$h_{ij}(y) = \cos(i\pi y_1)\cos(j\pi y_2),$$

а коэффициенты $A_{ij}, B_{ij}, C_{ij}, D_{ij}$ взяты равномерно в интервале [–1, 1]. В эксперименте использовались параметры метода $r = 3$ и $\varepsilon = 10^{-3}$, параметр построения развертки $m = 12$. Число параллельно выполняемых испытаний $P = 2$ и $P = 4$. При $P = 2$ число итераций составило 569, а число испытаний 1138. При $P = 4$ число итераций сократилось до 270 и было проведено 1080 испытаний. На рис. 1 приведены линии уровня функции, а также отмечены точки испытаний, проведенных методом при использовании $P = 4$.

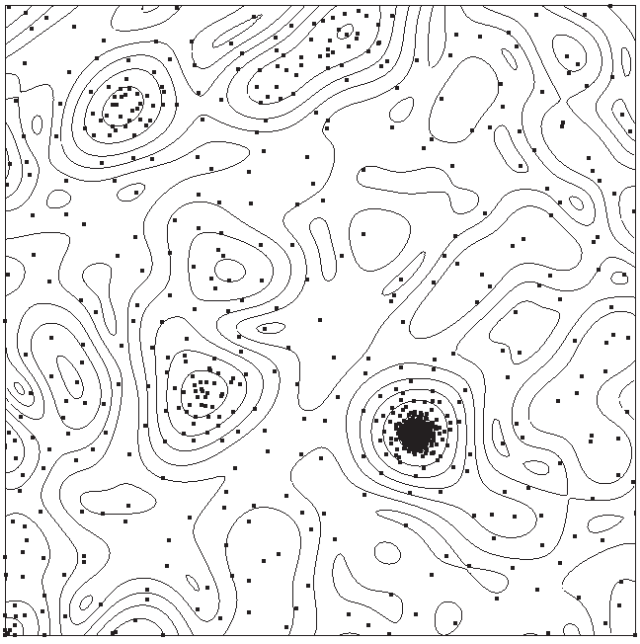

Рис. 1. Минимизация двумерной функции параллельным алгоритмом

Реализация на GPU

Какие же преимущества даст GPU при реализации алгоритмов глобальной оптимизации? Во-первых, CPU имеет лишь небольшое (до 16 в передовых моделях) число ядер, работающих на высокой тактовой частоте независимо друг от друга. Каждое ядро CPU является мощным вычислительным устройством. GPU же, напротив, работает на низкой тактовой частоте и имеет сотни более простых вычислительных элементов. Во-вторых, значительную долю кристалла CPU занимает быстродействующая кеш-память, в то время как практически весь GPU состоит из арифметико-логических блоков. Поэтому GPU особенно эффективны в задачах, число арифметических операций в которых велико по сравнению с операциями над памятью.

Применительно к методам глобальной оптимизации операцией, которую можно эффективно реализовать на GPU, является параллельное вычисление сразу многих значений целевой функции. Естественно, что для этого требуется реализовать на GPU процедуру вычисления значения функции. Пересылки данных от CPU к GPU будут минимальные: требуется лишь передать на GPU координаты точек испытаний и получить обратно значения функции в этих точках. Функции, определяющие обработку результатов испытаний в соответствии с алгоритмом и требующие работы с большим объемом накопленной поисковой информации, могут быть эффективно реализованы на CPU.

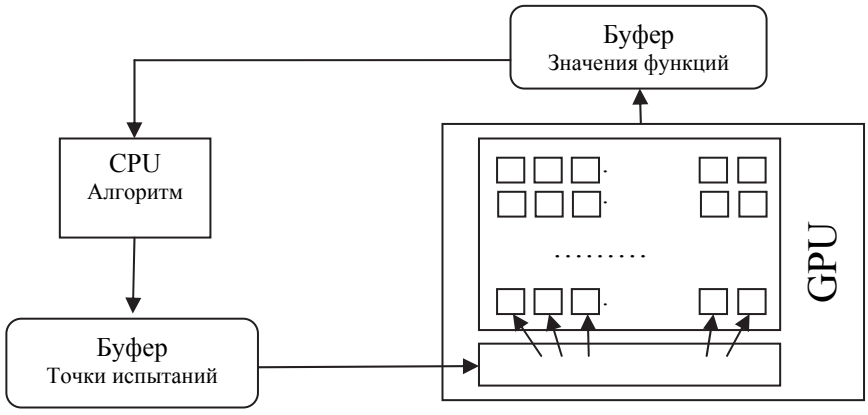

Рис. 2. Схема информационных обменов в GPU-алгоритме

Общая схема организации вычислений с использованием GPU приведена на рис. 2. В соответствии с данной схемой на CPU выполняются шаги 1–4 параллельного алгоритма глобального поиска. Координаты P точек испытаний, вычисленные на шаге 4 алгоритма, накапливаются в промежуточном буфере, а затем передаются на графический процессор. На GPU происходит вычисление значений функции в этих точках, после чего результаты испытаний (снова через промежуточный буфер) передаются на CPU.

Результаты численных экспериментов

Вычислительные эксперименты проводились на одном из узлов высокопроизводительного кластера ННГУ им. Н.И. Лобачевского, под управлением операционной системы семейства Windows (HPC Server 2008). Узел располагает двумя процессорами Intel Sandy Bridge E5-2660 2.2 GHz, 64 Gb RAM, и графическим процессором NVIDIA Tesla M2090. Центральный процессор является восьмиядерным (т.е. всего на узле доступно 16 физических ядер и 16 виртуальных в режиме HyperThreading), а на графическом процессоре доступны 16 мультипроцессоров (512 ядер). Для реализации GPU-алгоритма использовалась технология CUDA.

Отметим, что известные тестовые задачи из области многомерной глобальной оптимизации характеризуются малым временем вычисления значений целевой функции. Обычно подобный расчет сводится к суммированию нескольких (порядка размерности задачи) значений элементарных функций. Например, известная тестовая функция Растригина, глобальный минимум которой расположен в $y^* = (0, ..., 0)$ с $\varphi(y^*) = 0,$ задается выражением

$$\varphi(y) = 10N + \sum_{i=1}^{N} \left[y_i^2 - 10\cos(2\pi y_i) \right], \quad D = \{-2{,}2 \le y_i \le 1{,}8, \ 1 \le i \le N\}. \quad (4)$$

Исходя из этого с целью имитации вычислительной трудоемкости, присущей прикладным задачам оптимизации, расчет целевой функции во всех проводимых экспериментах был усложнен дополнительными вычислениями, не меняющими вид функции и расположение ее минимумов.

Перед проведением основных экспериментов произведем предварительную серию запусков для определения основных параметров параллельного алгоритма. Главным из них является число потоков P, поскольку от него напрямую зависит время работы программы. Также отметим, что важным параметром для запуска любой GPU-программы (в отличие от параллельной программы на CPU) является размер блока потоков (*thread block*). Отдельные потоки на GPU группируются в блоки потоков одинакового размера, при этом каждый блок потоков выполняется на отдельном мультипроцессоре. Потоки внутри блока потоков могут эффективно взаимодействовать между собой с помощью общих данных в разделяемой памяти и синхронизации.

Предварительная серия экспериментов будет выполнена на тестовой задаче (4) при $N = 5$. Будем варьировать число P используемых ядер на GPU и размер PB блока потоков, все остальные параметры метода не изменяются ($r = 2{,}3$, $\varepsilon = 0{,}01$, параметр построения кривой Пеано $m = 10$). На рис. 3 приведены графики зависимости времени работы GPU-алгоритма с фиксированным размером блока потоков PB в зависимости от числа потоков P. На рис. 4 отображена зависимость числа выполненных испытаний в секунду параллельным алгоритмом с фиксированным размером блока потоков PB в зависимости от числа параллельных потоков P.

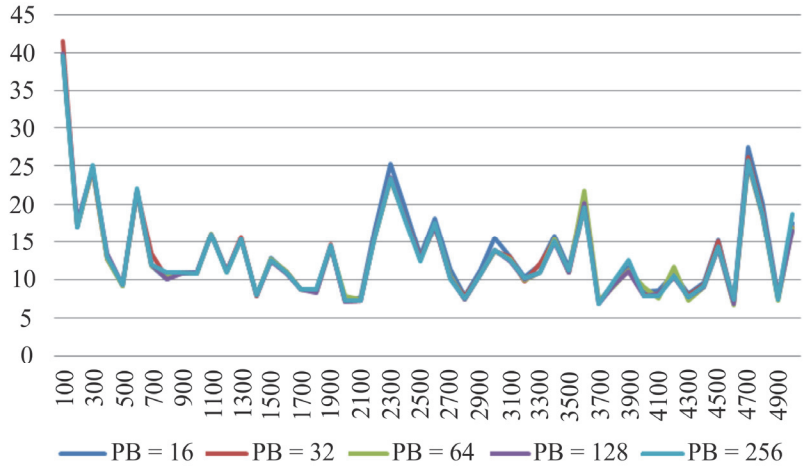

Рис. 3. Время работы GPU-алгоритма с разным фиксированным размером блока потоков PB в зависимости от количества потоков P

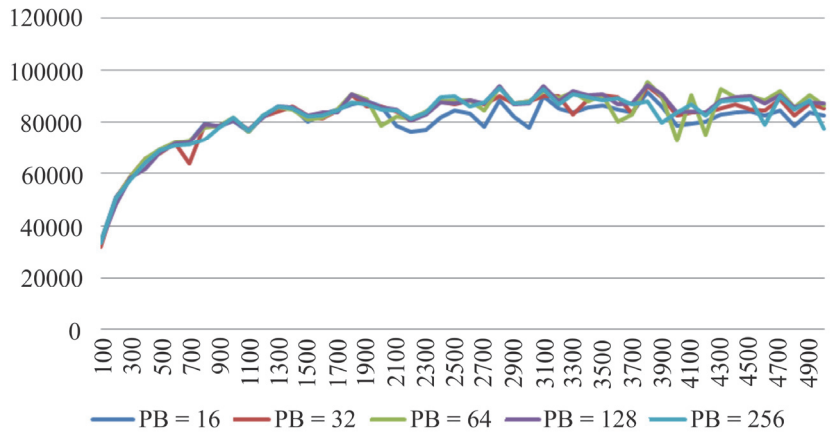

Рис. 4. Число испытаний в секунду GPU-алгоритма с разным фиксированным размером блока потоков *PB* в зависимости от количества потоков *P*

Как видно из графиков на рис. 3 и 4, наблюдается ускорение алгоритма при увеличении числа испытаний на одной итерации вплоть до числа ядер на GPU, после чего наступает насыщение, но при этом точно указать количество потоков, при котором программа будет работать быстрее всего, не удается. Таким образом, в дальнейшем будем проводить эксперименты с числом потоков от 100 до 1000 с шагом 100.

Для подбора размера блоков потоков рассмотрим табл. 1, где приведено время работы GPU-алгоритма в зависимости от *P* и *PB*.

Таблица 1

Время работы GPU-алгоритма в зависимости от размера блока потоков

PB	*P* = 100	*P* = 300	*P* = 500	*P* = 700	*P* = 900	*P* = 1000
PB = 16	39,43	24,84	9,25	11,80	10,90	10,92
PB = 32	41,50	24,52	9,45	13,43	10,91	10,80
PB = 64	39,71	24,54	9,24	11,78	10,85	10,76
PB = 128	39,85	24,66	9,40	11,91	10,86	10,90
PB = 256	39,61	25,08	9,30	12,03	10,96	10,74

В силу слабой информационной зависимости размер блока потоков *PB* не оказывает определяющего влияния, поэтому будем использовать *PB* = 64 во всех последующих экспериментах.

После фиксации параметров GPU-запуска проведем основные вычислительные эксперименты на серии многомерных многоэкстре-

мальных задач. В работах [18, 19] описан GKLS-генератор, позволяющий порождать задачи многоэкстремальной оптимизации с заранее известными свойствами: количеством локальных минимумов, размерами их областей притяжения, точкой глобального минимума, значением функции в ней и т.п.

Ниже приведены результаты численного сравнения трех последовательных алгоритмов – DIRECT [0], DIRECT*l* [20] и алгоритма глобального поиска (АГП) (результаты работы первых двух алгоритмов приводятся по работе [21]). Численное сравнение проводилось на классах функций Simple и Hard размерности 4 и 5 из работы [21], так как решение задач размерности 2 и 3 требует малого числа итераций, то использование GPU для решения этих задач нецелесообразно. Глобальный минимум y^* считался найденным, если алгоритм генерировал точку испытания y^k в δ-окрестности глобального минимума, т.е. $\|y^k - y^*\| \le \delta$. При этом размер окрестности выбирался (в соответствии с [21]) как $\delta = \|b-a\|\sqrt[N]{\Delta}$, здесь N – размерность решаемой задачи; a и b – границы области поиска D, параметр $\Delta = 10^{-6}$ при $N = 4$ и $\Delta = 10^{-7}$ при $N = 5$. При использовании метода АГП для класса Simple выбирался параметр $r = 4{,}5$, для класса Hard $r = 5{,}6$; параметр построения кривой Пеано был фиксированный $m = 10$. Максимально допустимое число итераций $K_{\max} = 1\,000\,000$.

В табл. 2 отражено среднее число итераций k_{av}, которые выполнил метод при решении серии задач из данных классов. Символ «>» отражает ситуацию, когда не все задачи класса были решены каким-либо методом. Это означает, что алгоритм был остановлен по причине достижения максимально допустимого числа итераций $K_{\max}$. В этом случае значение $K_{\max} = 1\,000\,000$ использовалось при вычислении среднего значения числа итераций k_{av}, что соответствует нижней оценке этого среднего значения. Количество нерешенных задач указано в скобках.

Таблица 2

Среднее число итераций

N	Problem class	DIRECT	DIRECT*l*	АГП
4	Simple	>47 282 (4)	18 983	11 953
	Hard	>95 708 (7)	68 754	25 263
5	Simple	>16 057 (1)	16 758	15 920
	Hard	>217 215 (16)	>269 064 (4)	>148 342 (4)

Как видно из табл. 2, АГП превосходит методы DIRECT и DIRECT*l* на всех классах задач по среднему числу итераций. При этом в классе 5-Hard каждый из методов решил не все задачи: DIRECT не решил 16 задач, DIRECT*l* и АГП – по 4 задачи.

Оценим теперь ускорение, которое достигается при использовании параллельного алгоритма глобального поиска в зависимости от числа P используемых ядер. В табл. 3 приведено среднее число итераций, которые выполнил метод при решении серии задач на CPU, а в табл. 4 – среднее время решения одной задачи, в табл. 5 и 6 приведены аналогичные данные для GPU-реализации. Ускорение GPU-алгоритма (табл. 7 и 8) будем измерять относительно CPU-алгоритма, запущенного на 32 ядра (см. столбец $P = 32$ в табл. 3 и 4)

Таблица 3

Среднее число итераций на CPU

N	Problem class	$P = 1$	$P = 2$	$P = 4$	$P = 8$	$P = 16$	$P = 32$
4	Simple	11 953	6 014	2 547	1 413	756	323
	Hard	25 263	16 509	8 970	4 666	1 910	1252
5	Simple	15 920	17 797	8 246	3 773	1 713	781
	Hard	>148 342(4)	98 667	45 410	26 732	11 322	11 745

Таблица 4

Среднее время решения задачи на CPU

N	Problem class	$P = 1$	$P = 2$	$P = 4$	$P = 8$	$P = 16$	$P = 32$
4	Simple	3,02	1,80	1,00	0,71	0,57	0,38
	Hard	6,05	4,48	2,94	1,88	1,36	1,25
5	Simple	4,84	5,84	3,18	1,82	1,44	0,95
	Hard	>42.5	33,77	15,62	14,16	8,37	11,67

Таблица 5

Среднее число итераций на GPU

N	Problem class	$P = 100$	$P = 200$	$P = 300$	$P = 400$	$P = 500$	$P = 800$	$P = 1000$
4	Simple	105	56	53	28	24	18	12
	Hard	352	181	107	87	84	37	33
5	Simple	193	105	62	44	41	55	36
	Hard	2452	809	508	376	353	246	172

Таблица 6

Среднее время решения задачи на GPU

N	Problem class	$P = 100$	$P = 200$	$P = 300$	$P = 400$	$P = 500$	$P = 800$	$P = 1000$
4	Simple	0,16	0,16	0,22	0,16	0,16	0,19	0,15
	Hard	0,50	0,47	0,43	0,44	0,52	0,37	0,41
5	Simple	0,33	0,32	0,29	0,27	0,31	0,60	0,48
	Hard	3,46	2,11	1,98	1,97	2,25	2,54	2,15

Таблица 7

Ускорение по времени $S(P)$ на GPU

N	Problem class	$P = 100$	$P = 200$	$P = 300$	$P = 400$	$P = 500$	$P = 800$	$P = 1000$
4	Simple	2,30	2,35	1,75	2,37	2,38	1,94	2,45
	Hard	2,49	2,68	2,93	2,85	2,41	3,38	3,07
5	Simple	2,90	2,99	3,32	3,58	3,05	1,59	1,98
	Hard	4,09	6,72	7,17	7,19	6,30	5,56	6,59

Таблица 8

Ускорение по итерациям $s(P)$ на GPU

N	Problem class	$P = 100$	$P = 200$	$P = 300$	$P = 400$	$P = 500$	$P = 800$	$P = 1000$
4	Simple	3,07	5,78	6,11	11,43	13,54	17,96	27,42
	Hard	3,55	6,90	11,70	14,38	14,98	33,46	37,40
5	Simple	4,05	7,45	12,57	17,80	18,83	14,19	21,91
	Hard	4,79	14,51	23,12	31,21	33,26	47,68	68,21

Результаты экспериментов показывают преимущество GPU-алгоритма по сравнению с CPU-алгоритмом. Так, алгоритм с использованием максимального количества ядер CPU работает медленнее, чем любая из представленных GPU-реализаций. Также результаты показывают значительное ускорение по числу итераций. Например, для решения задач самого сложного из рассмотренных здесь классов – 5-Hard – потребовалось в среднем всего 353 параллельных итерации на GPU, тогда как даже при использовании всех вычислительных ядер CPU число итераций составляло более 11 тысяч.

Заключение

Подводя итоги, отметим, что использование графических процессоров для решения задач глобальной оптимизации является новым перспективным направлением, так как высокая производительность современных суперкомпьютеров достигается (в основном) за счет использования ускорителей.

В данной работе рассмотрен параллельный алгоритм решения задач многомерной многоэкстремальной оптимизации и его реализация на GPU. Данный алгоритм разработан в рамках информационно-статистического подхода.

С целью экспериментального подтверждения теоретических свойств рассматриваемого параллельного алгоритма проведены вычислительные эксперименты на серии из нескольких сотен тестовых задач разной размерности. При этом результаты работы последовательного алгоритма показывают его превосходство над другими известными методами глобальной оптимизации. Одновременно параллельный алгоритм показывает хорошее ускорение (как GPU-, так и CPU-версии), а GPU-версия параллельного алгоритма показывает ускорение от 2 до 6 раз по сравнению с наиболее быстрой параллельной CPU-версией.

Библиографический список

1. Global Optimization: Scientific and Engineering Case Studies / ed. J.D. Pinter. – Springer, 2006. – 546 p.

2. Wen-mei Hwu. GPU Computing Gems Emerald Edition (Applications of GPU Computing Series). – Morgan Kaufmann, 2011. – 886 p.

3. An efficient implementation of parallel simulated annealing algorithm in GPUs / A.M. Ferreiro, J.A. García, J.G. López-Salas, C. Vázquez // Journal of Global Optimization. – 2013. – Vol. 57, № 3. – P. 863–890.

4. Zhu W. Massively parallel differential evolution – pattern search optimization with graphics hardware acceleration: an investigation on bound constrained optimization problems // Journal of Global Optimization. – 2011. – Vol. 50, № 3. – P. 417–437.

5. García-Martínez J.M., Garzón E.M., Ortigosa P.M. A GPU implementation of a hybrid evolutionary algorithm: GPuEGO // The Journal of Supercomputing. – 2014. DOI: 10.1007/s11227-014-1136-7

6. GPU implementation of a road sign detector based on particle swarm optimization / L. Mussi [et al.] // Evolutionary Intelligence. – 2010. – Vol. 3, № 3–4. – P. 155–169.

7. Langdon W.B. Graphics processing units and genetic programming: an overview // Soft Computing. – 2011. – Vol. 15, № 8. – P. 1657–1669.

8. Gergel V.P., Strongin R.G. Parallel computing for globally optimal decision making on cluster systems // Future Generation Computer Systems. – 2005. – Vol. 21, № 5. – P. 673–678.

9. Evtushenko Yu.G., Malkova V.U., Stanevichyus A.A. Parallel global optimization of functions of several variables // Computational Mathematics and Mathematical Physics. – 2009. – Vol. 49, № 2. – P. 246–260.

10. Design and implementation of a massively parallel version of DIRECT / J. He, A. Verstak, L.T. Watson, M. Sosonkina // Computational Optimization and Applications. – 2008. – Vol. 40, № 2. – P. 217–245.

11. Paulavicius R., Zilinskas J., Grothey A. Parallel branch and bound for global optimization with combination of Lipschitz bounds // Optimization Methods & Software. – 2011. – Vol. 26, № 3. – P. 487–498.

12. Strongin R.G., Sergeyev Ya.D. Global Optimization with Non-convex Constraints. Sequential and Parallel Algorithms. – Kluwer Academic Publishers, 2000. – 704 p.

13. Sergeyev Ya.D., Strongin R.G., Lera D. Introduction to global optimization exploiting space-filling curves. – Springer, 2013. – 125 p.

14. Параллельные вычисления в задачах глобальной оптимизации / Р.Г. Стронгин, В.П. Гергель, В.А. Гришагин, К.А. Баркалов. – М.: Изд-во Моск. ун-та, 2013. – 280 с.

15. Barkalov K.A., Strongin R.G. A global optimization technique with an adaptive order of checking for constraints // Computational Mathematics and Mathematical Physics. – 2002. – Vol. 42, № 9. – P. 1289–1300.

16. Gergel V.P. A method of using derivatives in the minimization of multiextremum functions // Computational Mathematics and Mathematical Physics. – 1996. – Vol. 36, № 6. – P. 729–742.

17. Gergel V.P. A global optimization algorithm for multivariate functions with lipschitzian first derivatives // Journal of Global Optimization. – 1997. – Vol. 10, № 3. – P. 257–281.

18. Software for generation of classes of test functions with known local and global minima for global optimization / M. Gaviano, D. Lera, D.E. Kvasov, Y.D. Sergeyev // ACM Transactions on Mathematical Software. – 2003. – Vol. 29. – P. 469–480.

19. Сергеев Я.Д., Квасов Д.Е. Диагональные методы глобальной оптимизации. – М.: Физматлит, 2008. – 352 с.

20. Gablonsky J.M., Kelley C.T. A Locally-Biased Form of the DIRECT Algorithm // Journal of Global Optimization. – 2001. – Vol. 21, № 1. – P. 27–37.

21. Sergeyev Ya.D., Kvasov D.E. Global search based on efficient diagonal partitions and a set of Lipschitz constants // SIAM Journal on Optimization. – 2006. – Vol. 16, № 3. – P. 910–937.

References

1. Global Optimization: Scientific and Engineering Case Studies. Ed. J.D. Pinter. Springer, 2006. 546 p.

2. Wen-mei Hwu. GPU Computing Gems Emerald Edition (Applications of GPU Computing Series). Morgan Kaufmann, 2011. 886 p.

3. Ferreiro A.M., García J.A., López-Salas J.G., Vázquez C. An efficient implementation of parallel simulated annealing algorithm in GPUs. *Journal of Global Optimization*, 2013, vol. 57, no. 3, pp. 863-890.

4. Zhu W. Massively parallel differential evolution – pattern search optimization with graphics hardware acceleration: an investigation on bound constrained optimization problems. *Journal of Global Optimization*, 2011, vol. 50, no. 3, pp. 417-437.

5. García-Martínez J.M., Garzón E.M., Ortigosa P.M. A GPU implementation of a hybrid evolutionary algorithm: GPuEGO. *The Journal of Supercomputing*, 2014. DOI: 10.1007/s11227-014-1136-7

6. Mussi L. [et al.] GPU implementation of a road sign detector based on particle swarm optimization. *Evolutionary Intelligence*, 2010, vol. 3, no. 3–4, pp. 155-169.

7. Langdon W.B. Graphics processing units and genetic programming: an overview. *Soft Computing*, 2011, vol. 15, no. 8, pp. 1657-1669.

8. Gergel V.P., Strongin R.G. Parallel computing for globally optimal decision making on cluster systems. *Future Generation Computer Systems*, 2005, vol. 21, no. 5, pp. 673-678.

9. Evtushenko Yu.G., Malkova V.U., Stanevichyus A.A. Parallel global optimization of functions of several variables. *Computational Mathematics and Mathematical Physics*, 2009, vol. 49, no. 2, pp. 246-260.

10. He J., Verstak A., Watson L.T., Sosonkina M. Design and implementation of a massively parallel version of DIRECT. *Computational Optimization and Applications*, 2008, vol. 40, no. 2, pp. 217-245.

11. Paulavicius R., Zilinskas J., Grothey A. Parallel branch and bound for global optimization with combination of Lipschitz bounds. *Optimization Methods & Software*, 2011, vol. 26, no. 3, pp. 487-498.

12. Strongin R.G., Sergeyev Ya.D. Global Optimization with Non-convex Constraints. Sequential and Parallel Algorithms. Kluwer Academic Publishers, 2000. 704 p.

13. Sergeyev Ya.D., Strongin R.G., Lera D. Introduction to global optimization exploiting space-filling curves. Springer, 2013. 125 p.

14. Strongin R.G. Parallelnye vichisleniya v zadachakh globalnoy optimizatsii [Parallel computing in global optimization problems]. Moscow University Press, 2013. 280 p.

15. Barkalov K.A., Strongin R.G. A global optimization technique with an adaptive order of checking for constraints. *Computational Mathematics and Mathematical Physics*, 2002, vol. 42, no. 9, pp. 1289-1300.

16. Gergel V.P. A method of using derivatives in the minimization of multiextremum functions. *Computational Mathematics and Mathematical Physics*, 1996, vol. 36, no. 6, pp. 729-742.

17. Gergel V.P. A global optimization algorithm for multivariate functions with lipschitzian first derivatives. *Journal of Global Optimization*, 1997, vol. 10, no. 3, pp. 257-281.

18. Gaviano M., Lera D., Kvasov D.E., Sergeyev Y.D. Software for generation of classes of test functions with known local and global minima for global optimization. *ACM Transactions on Mathematical Software*, 2003, vol. 29, pp. 469-480.

19. Sergeyev Ya.D. Diagonalnye metody globalnoy optimizatsii [Diagonal global optimization methods]. Moscow: Fizmatlit, 2008. 352 p.

20. Gablonsky J.M., Kelley C.T. A Locally-Biased Form of the DIRECT Algorithm. *Journal of Global Optimization*, 2001, vol. 21, no. 1, pp. 27-37.

21. Sergeyev Ya.D., Kvasov D.E. Global search based on efficient diagonal partitions and a set of Lipschitz constants. *SIAM Journal on Optimization*, 2006, vol. 16, no. 3, pp. 910-937.

Об авторах

Баркалов Константин Александрович (Нижний Новгород, Россия) – кандидат физико-математических наук, доцент кафедры «Математическое обеспечение ЭВМ» Нижегородского государственного университета им. Н.И. Лобачевского (603950, Н. Новгород, пр. Гагарина, 23, корп. 2, e-mail: barkalov@vmk.unn.ru).

Лебедев Илья Геннадьевич (Нижний Новгород, Россия) – программист кафедры «Математическое обеспечение ЭВМ» Нижегородского государственного университета им. Н.И. Лобачевского (603950, Н. Новгород, пр. Гагарина, 23, корп. 2, e-mail: lebedev.ilya.g@gmail.com).

About the authors

Konstantin A. Barkalov (Nizhniy Novgorod, Russian Federation) – Ph. D. in Physics and Mathematical Sciences, Associate Professor, Software Department, Lobachevskiy State University of Nizhniy Novgorod (23, building 2, Gagarina av., Nizhniy Novgorod, 603950, Russian Federation, e-mail: barkalov@vmk.unn.ru).

Ilya G. Lebedev (Nizhniy Novgorod, Russian Federation) – programmer, Software Department, Lobachevskiy State University of Nizhniy Novgorod (23, building 2, Gagarina av., Nizhniy Novgorod, 603950, Russian Federation, e-mail: lebedev.ilya.g@gmail.com).

Получено 1.10.2014