С. Кукаева

Нижегородский государственный университет им. Н.И. Лобачевского

ТЕСТИРОВАНИЕ РЕАЛИЗАЦИИ МОАГП С ИСПОЛЬЗОВАНИЕМ РАЗВЕРТОК ПЕАНО НА GPU

Постановка задачи и используемые подходы. Рассматривается конечномерная задача оптимизации (задача нелинейного программирования)

$$f(y) \to inf, y \in Y \subseteq R^N$$

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

т.е. задача отыскания экстремальных значений целевой (минимизируемой) функции $f(y)$ в области Y, задаваемой координатными ограничениями [1, 4].

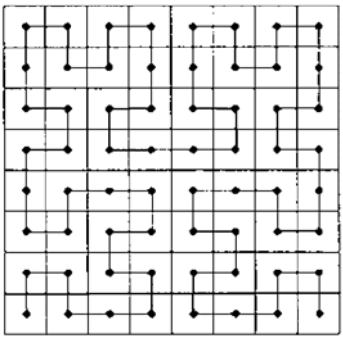

Рис. 1. Образ отрезка [0, 1] при отображении пеаноподобной разверткой третьего порядка

Не уменьшая общности, далее будем считать, что поставлена задача нахождения минимума. Одним из подходов к решению многомерной задачи оптимизации является сведение многомерной задачи к одномерной. Это может сделано разными способами. Наиболее эффективными являются многошаговая схема оптимизации и редукция размерности на основе кривых Пеано, о которой далее и пойдет речь. Данная схема основана на известном фундаментальном факте, согласно которому N-мерный гиперпараллелепипед и отрезок [0, 1] вещественной оси являются равномощными множествами, и отрезок [0, 1] может быть однозначно и непрерывно отображен на гиперпараллелепипед. Такие отображения и называются кривыми (или развертками) Пеано [1, 2].

На рис. 1 изображен образ отрезка [0, 1] при отображении пеаноподобной разверткой при разбиении третьего порядка (отметим, что в данной работе используется 8-й порядок разбиения).

В качестве основного алгоритма решения была выбрана модификация информационно-статистического алгоритма глобального поиска (АГП) – многомерный обобщенный алгоритм глобального поиска (МОАГП), в котором в качестве характеристики интервала используется величина

$$R(i) = m^N \sqrt{|x_i - x_{i-1}|} + \frac{(z_i - z_{i-1})^2}{m^N \sqrt{|x_i - x_{i-1}|}} - 2(z_i + z_{i-1}),$$

где

$$m = f(x) = \begin{cases} rM, M > 0 \\ 1, M = 0 \end{cases} (r-1),$$

$$M = \max_{1 \le i \le \tau} \frac{(z_i - z_{i-1})}{\sqrt[N]{|x_i - x_{i-1}|}},$$

$$z_j = f\left(y(x_j)\right), 1 \le j \le \tau,$$

$y(x_j)$ – точка N-мерного гиперпараллелепипеда, являющаяся образом точки x_j отрезка [0, 1].

В данной работе используется одна развертка на всю область поиска минимума.

Искомый глобальный минимум будем искать согласно следующему алгоритму:

1. Запускаем K потоков на GPU, пронумерованных 0, 1, …, K–1 соответственно.

2. Отрезок [0, 1] также разбиваем на K отрезков, пронумерованных 0, 1, …, K–1.

3. Потоку i соответствует отрезок с тем же номером. Каждый поток ищет минимум на своем отрезке. Важно, что потоки не обмениваются информацией относительно найденных точек, таким образом, каждый поток находит свой минимум на своем участке. Данные о найденных минимумах передаются на CPU.

4. Находим минимальное из тех значений, что найдены в каждом потоке. Эта часть является последовательно исполняемой на CPU.

В данной версии реализация поиска минимума отдельным потоком абсолютно идентична реализации поиска минимума на CPU.

Результаты вычислительных экспериментов. В наших численных экспериментах было использовано два класса (простой и сложный) 2-мерных тестовых ND-функций, порожденных GKLS-генератором [3]. Этот генератор обладает несколькими преимуществами, которые позволяют использовать его в качестве хорошего инструмента для сравнения численных алгоритмов.

Он генерирует классы 100 тестовых с одинаковым числом локальных минимумов и предоставляет полную информацию о каждой функции: ее размерность, значения всех локальных минимумов, их координаты, области притяжения и т.д. Существует возможность с легкостью генерировать более сложные или более простые тестовые классы. Только 5 параметров должны быть определены пользователем, а остальные генерируются случайным образом. Важная особенность генератора состоит в полной повторяемости экспериментов: если вы используете те же самые 5 параметров, то при каждом запуске генератор будет порождать такой же класс функций [3].

Важно, что вычисления значений функций производится не на CPU, а на GPU. Параметры, необходимые для вычисления значения функции, заносятся в специальную структуру, которая передается в глобальную память GPU. Используя эти данные, на GPU каждый поток вычисляет значение в необходимой точке.

Таблица 1

Параметры тестовых классов функций

Класс n	N	m	f^*	d	r_g
Простой	2	10	–1,0	0,66	0,33
Сложный	2	10	–1,0	0,90	0,20

В табл. 1 представлены параметры для каждого из используемых классов функций, где f^* – значение функции в глобальном минимуме, d – максимальное расстояние от точки глобального минимума до центра параболоида, r_g – радиус области притяжения глобального минимума функции. Глобальный оптимум считается найденным, если точка найденного оптимума попала в некоторую окрестность ρ настоящего минимума, которая в нашем случае равна 0,02.

Параметр надежности алгоритма для «простого» и «сложного» классов функций r выбирался равным 2,1. Это значение было выбрано экспериментально, поскольку при нем удается получить решение всех задач тестового набора при относительно небольшом числе испытаний. Вторым важным параметром параллельной реализации алгоритма являлось число K начальных отрезков, на которые разбивается исходный отрезок. Следует напомнить, что в каждой подобласти независимо от других запускается параллельный поток, в котором выполняется алгоритм поиска глобального минимума. В каждом потоке разрешается выполнить по M испытаний. Тем самым общее число выполненных испытаний составит $K{\cdot}M$.

Данные, представленные на рис. 2 и 3, отражают тот факт, что за счет выбранного принципа параллельной реализации алгоритма глобального поиска в его независимо выполняемых частях не происходит обменов информацией и неизбежно возникает избыточность по числу измерений.

Верхняя кривая соответствует однопоточной реализации, которая обладает наибольшей надежностью при заданном общем количестве выполняемых измерений целевой функции. Увеличение числа параллельных потоков, каждый из которых связывался со своим интервалом первоначального разбиения области поиска $[0;1]$, приводило к росту количества избыточных измерений.

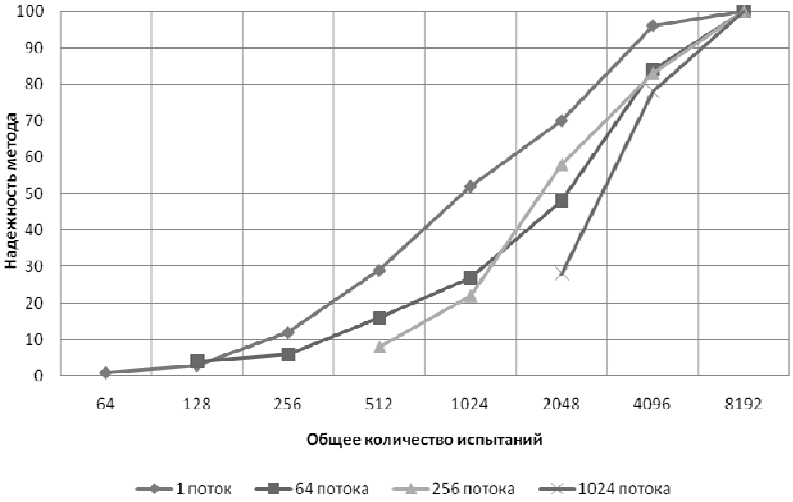

Рис. 2. Простой класс функций

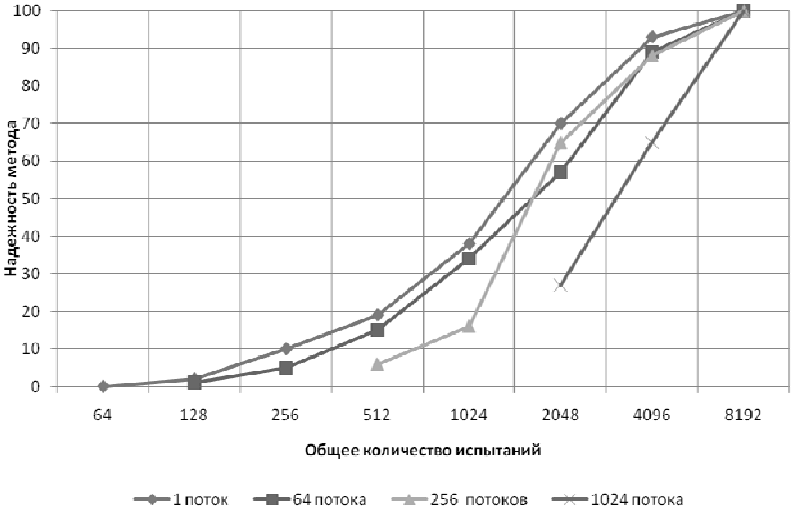

Рис. 3. Сложный класс функций

Оценим коэффициент избыточности, сравнивая затраты по числу измерений целевой функции в многопоточном и однопоточном вариантах реализации на GPU при уровне надежности

0,88 (для «простого» класса функций) и 0,89 (для «сложного» класса функций). По этой методике для значения числа потоков $M = 512$ получим коэффициент избыточности равный 1,1 (для «простого» класса функций) и 1,22 (для «сложного» класса функций). Наибольший коэффициент избыточности за весь период работы алгоритмов наблюдался при уровне надежности 0,5–0,6 и был примерно равен 2 (для сложного класса функций) и ≈4 (для простого класса функций).

Следует учитывать, что при параллельном исполнении программного кода в потоках при выполнении очередной итерации в одном потоке (приводящей к вычислению значения целевой функции) в среднем одновременно выполняются аналогичные итерации и в остальных потоках. Таким образом, за период одной итерации в среднем выполняется *M* вычислений значений функции (по количеству выполняемых потоков). Следовательно, с точки зрения обобщенно понимаемого времени ожидания результата решения следует рассматривать операционные характеристики иного, чем на рис. 2, 3 вида, а именно, необходимо перейти к характеристикам типа «надежность – число итераций», представленным в табл. 2, 3. В этом случае становится очевидным преимущество параллельных реализаций алгоритма.

Таблица 2

Простой класс функций

Количество потоков	Количество испытаний в одном потоке						
	2	4	8	16	32	64	128
32	3	5	11	18	36	56	83
64	4	6	16	27	48	84	100
128	6	17	36	65	85	99	100
256	8	22	58	83	100	100	100
512	16	36	88	100	100	100	100
1024	28	78	100	100	100	100	100

Таблица 3

Сложный класс функций

Количество потоков	Количество испытаний в одном потоке						
	2	4	8	16	32	64	128
32	0	3	5	14	31	63	88
64	1	5	15	34	57	89	100
128	3	8	25	56	89	100	100
256	6	16	65	88	100	100	100
512	20	40	89	99	100	100	100
1024	27	65	100	100	100	100	100

Если, например, проанализировать ускорение, вытекающее из табл. 2, 3, то при уровне надежности 1,0 и увеличении количества потоков с 1 до 128, мы получим сокращение числа итераций примерно с 6000 до 128, что показывает ускорение по количеству итераций в 48 раз.

При проведении исследований дополнительно были выполнены оценки ускорения непосредственно по времени. При этом замерялось среднее время до момента определения с заданной точностью $\rho = 0,02$ глобально-оптимального решения (или до исчерпания установленного максимального числа итераций, если решение не определялось). Усреднение проводилось по всем 100 функциям GKLS-генератора. Максимальное число итераций на один поток выбиралось так, чтобы обеспечивался уровень надежности 1,0 (правильное решение 100 задач из 100). В качестве тестового стенда при замерах времени использовалась: GPU- ATI Radeon 4890, CPU – Intel® Core™ 2 Duo, E6559, 2,33 GHz, 4.00 Gb RAM, 64 bit OS. В данной реализации в качестве основного инструмента выбрана технология OpenCL.

Усредненные значения времени были получены для нескольких значений числа M используемых параллельных потоков на GPU. Коэффициент ускорения вычислялся как отношение среднего времени решения на CPU (1 поток) к среднему времени при $M > 1$. Как видно из графика, ускорение составляет 55–60 раз по сравнению с последовательной версией.

Как видно из результатов, использование графических процессоров для вычислений общего назначения в который раз доказало свою эффективность. Конечно, мы получаем определенную избыточность в вычислениях, однако ускорение по времени с ростом числа потоков неуклонно растет. Приведенные результаты показывают, что использование графических карт в данной области оправданно (ускорение работы до 55–60 раз), однако требуется более детально рассмотреть вопросы, связанные с точностью вычислений, эффективным использованием ресурсов самой видеокарты. В связи с этим интересно протестировать некоторые модификации данного алгоритма. К примеру, разбить исходный гипериинтервал на подгипериинтервалы и в каждом из них реализовать отдельную развертку. Также необходимо протестировать работу для больших размерностей (к примеру, 3–5).

Работа выполнена при финансовой поддержке Федерального агентства по науке и инновациям (ФЦП «Научные и научно-педагогические кадры инновационной России», госконтракт № 02.740.11.5018).

Список литературы

1. Городецкий С.Ю., Гришагин В.А. Модели и методы конечномерной оптимизации: учеб. курс. Ч. 2. Нелинейное программирование и многоэкстремальная оптимизация. – Н. Новгород, 2003.

2. Lera D., Sergeyev Ya.D. (2010) Lipschitz and Holder global optimization using space-filling curves // Applied Numerical Mathematics, 60(1–2), 115–129.

3. Software for generation of classes of test functions with known local and global minima for global optimization / M. Gaviano [et al.] // ACM TOMS 29 (4) (2003) 469–480.

4. Гришагин В.А. Параллельные алгоритмы многоэкстремальной оптимизации в областях с вычислимой границей. Высокопроизводительные параллельные вычисления на кластерных системах. – Н. Новгород, 2007.