Решение задачи поиска наименьшего расстояния до политопа с использованием графических ускорителей

Е. А. Нурминский, П. Л. Поздняк
Институт автоматики и процессов управления ДВО РАН, Владивосток, Россия
e-mail: nurmi@dvo.ru, pozpl@dvo.ru

Представлена реализация метода подходящих аффинных подпространств для графических ускорителей. Описываются модификации алгоритма, особенности реализации для архитектуры графических ускорителей и результаты численных экспериментов. Для задач размерностью 10^5 по сравнению с типовым вычислительным устройством стандартной архитектуры получен прирост производительности почти в десять раз.

Ключевые слова: проекция, графические ускорители.

Введение

Решение задачи проекции начала координат на n-мерный выпуклый многогранник X, заданный своими вершинами, широко используется при построении алгоритмов негладкой оптимизации [1], распознавании образов и автоматической классификации [2], в компьютерной томографии и радиационной терапии [3]. Для практических приложений характерны большие размерности исходного пространства и большое количество вершин политопа X, а к алгоритмам решения данной задачи предъявляются жесткие требования по вычислительной эффективности и точности. Эти обстоятельства мотивировали поиск новых алгоритмов решения задачи проекции и способов ее реализации с использованием нестандартных вычислительных архитектур. В работе исследована возможность ускорения вычислений с помощью графических ускорителей GPU (Graphics Processing Unit). Применение GPU для решения задач большой размерности хорошо зарекомендовало себя в таких областях как молекулярная динамика [4], магнитно-резонансная томография [5] и т. д. В нашей задаче использование GPU при решении задач высокой размерности позволяет увеличить скорость вычислений до 10–11 раз.

1. Метод подходящих аффинных подпространств

Рассматривается задача проекции начала координат на выпуклый многогранник X в n-мерном евклидовом пространстве E со скалярным произведением xy и нормой $\|x\| = \sqrt{xx}$. Другими словами, речь идет о нахождении точки $x^* \in X$ такой, что

$$\|x^*\|^2 = \min_{x \in X} \|x\|^2. \tag{1}$$

В данной задаче предполагается, что множество X является политопом, т. е. выпуклой оболочкой конечного набора точек $\hat{X}$ пространства E:

$$\hat{X} = \{\hat{x}^1, \hat{x}^2, \ldots, \hat{x}^m\} = \{\hat{x}^i, i \in \mathcal{M}\},$$

где $\mathcal{M} = \{1, 2, \ldots, m\}$ — индексное множество вершин политопа X. Для множества $Z \subset \hat{X}$ с индексным множеством $\mathrm{ind}\,(Z) \subset \mathcal{M}$ аффинную ($\mathrm{aff}\,(Z)$) и выпуклую ($\mathrm{co}\,(Z)$) оболочки определим традиционным образом:

$$\mathrm{aff}\,(Z) = \{x = \sum_{x \in Z, i \in \mathrm{ind}\,(Z)} \lambda_i \hat{x}^i, \sum_{i \in \mathrm{ind}\,(Z)} \lambda_i = 1\}, \tag{2}$$

$$\begin{aligned} \mathrm{co}\,(\hat{X}) = \{x = &\sum_{x \in Z, i \in \mathrm{ind}\,(Z)} \lambda_i \hat{x}^i, \\ &\sum_{i \in \mathrm{ind}\,(Z)} \lambda_i = 1, \lambda_i \geq 0, i \in \mathrm{ind}\,(Z)\}. \end{aligned} \tag{3}$$

Выразим подходящий подсимплекс X как выпуклую оболочку аффинно независимого набора точек $S_I = \{\hat{x}_i, i \in I \subset \mathcal{M}\}$ таких, что

$$\min_{z \in \mathrm{aff}\,\{\hat{x}^i, i \in I\}} \|z\|^2 = \min_{z \in \mathrm{co}\,\{\hat{x}^i, i \in I\}} \|z\|^2.$$

Подходящим нуль-мерным симплексом является, например, любая точка $\hat{x}^i, i \in \mathcal{M}$.

Метод подходящих аффинных подпространств состоит в построении последовательности множеств $I_k \subset \mathcal{M}$ таких, что $\mathrm{aff}\,(\hat{x}^i, i \in I_k)$ представляет собой подходящее аффинное подпространство и расстояние от нуля

$$r_k = \min_{\mathrm{aff}\,\{\hat{x}^i, i \in I_k\}} \|x\|^2$$

до этих подпространств строго монотонно убывает. Множества I_k и соответственно $\hat{X}_k = \mathrm{co}\,\{\hat{x}^i, i \in I_k\}$ будем называть базисом.

В методе подходящих аффинных подпространств на вход каждой итерации подается базис, построенный на предыдущих шагах. В результате выполнения итерации алгоритма строится новый базис со строго меньшим расстоянием до начала координат, чем предыдущий. Сама итерация содержит внутренний цикл, в котором, возможно, происходит удаление части вершин из базиса. Для размерности пространства n трудоемкость итерации внутреннего цикла составляет не более $O(n^2)$ при добавлении вектора в базис и не более $O(\kappa n^2)$ при удалении вершин из базиса, где κ — количество векторов в получившемся базисе.

Описание алгоритма выглядит следующим образом. Положим, что в начале k-й итерации имеются базис I_k, соответствующий ему подходящий подсимплекс S_k и аффинное подпространство $H_k = \mathrm{aff}\,\{S_k\}$. Дальнейшие действия описывают выполнение k-й итерации.

Внешний цикл алгоритма:

1) решаем задачу нахождения элемента минимальной нормы для аффинного подпространства H_k

$$\min_{z \in H_k} \|z\|^2 = \|z^k\|^2; \tag{4}$$

2) если $x^i z^k \geq \|z^k\|^2$ для всех $i \in \mathcal{M} \setminus I_k$, то $z^k z^i \geq \|z^k\|^2$ для всех $i \in \mathcal{M}$ и z^k — решение задачи (1), и работа алгоритма на этом прекращается;

3) при невыполнении п. 2 выбираем x^i_H из условия

$$\|x^i_H z^k\|^2 = \min_{x^i z^k < \|z^k\|^2} \|x^i z^k\|^2,$$

где $i_H \in \mathcal{M} \setminus I$. Образуем новое индексное множество $I_s = I_k \cup \{i_H\}$, полагаем $w_s = z^k$ и, занулив счетчик внутренних итераций $s = 0$, выполняем следующий внутренний цикл:

a) образуем аффинное подпространство $\hat{H}^k_s = \{x^i, i \in I_s\}$;

b) решаем задачу проекции на аффинное подпространство $\hat{H}^k_s$

$$\min_{z \in H_s} \|z\|^2 = \|z^H\|^2;$$

c) если $z^s \in X$, то полагаем $H_{k+1} = \hat{H}^k_s, I_{k+1} = I_s$ и выходим из внутреннего цикла. Иначе полагаем

$$u^\mu = \mu z^s + (1 - \mu) w_s,$$

где находим максимальное μ такое, что $u^\mu \in X$, т. е.

$$\mu_{\max} = \max_{u^\mu \in X} \mu.$$

Легко видеть, что найденное таким образом $\mu_{\max}$ принадлежит некоторой грани симплекса S_k, т. е.

$$u^{\mu_{\max}} = w^{s+1} = \sum_{i \in I_s} \theta_i \hat{x}^i, \quad \sum_{i \in I_s} \theta_i = 1, \quad \theta_i > 0, \quad i \in I_s, \quad H \subseteq I_s.$$

Положим, что ς — индекс вектора в индексном пространстве I_s, соответствующий выбранному $\mu_{\max}$. Удалим этот вектор из базиса. В результате получим некоторый подходящий подсимплекс $\tilde{S}_k = S_k \backslash x^\varsigma$. После нахождения такого подсимплекса полагаем $I_{s+1} = I_s \backslash \varsigma, \hat{H}^k_{s+1} = \hat{H}^k_s \backslash x^\varsigma$ и переходим к следующей итерации внутреннего цикла;

d) завершим выполнение итерации внутреннего цикла и вернемся к его началу;

4) завершим выполнение итерации внешнего цикла и вернемся к его началу.

Данный алгоритм находит решение задачи проекции за конечное число итераций и для него показана глобальная “лучше чем линейная” скорость сходимости [6]. Как видно из приведенного описания, большую долю в алгоритме составляют матрично-векторные операции, что дает возможность использования графических ускорителей для ускорения его работы на задачах больших размерностей.

1.1. Вычислительные возможности GPU

В последнее время увеличение параллелизма стало основным механизмом роста производительности процессоров. Технология многоядерности позволяет использовать параллельные алгоритмы для решения задач, не привлекая дорогостоящего специализированного оборудования.

В настоящей работе предпринята попытка применения графических ускорителей для решения задачи (1) большой размерности. Современные графические ускорители имеют высокую степень физического параллелизма. Так, графические ускорители

NVIDIA (http://www.nvidia.com) содержат от 8 до 240 ядер на одном чипе. Это позволяет эффективно решать задачи с параллелизмом по данным с большим количеством арифметических операций по отношению к количеству обращений к памяти.

С целью реализации высокоуровневого доступа к аппаратным возможностям GPU для разработчиков неграфических приложений компанией NVIDIA была создана программно-аппаратная архитектура CUDA (Compute Unified Device Architechure). В состав CUDA входят API (Application Programming Interface) для работы с GPU, специальный SDK (Software Development Kit) и компилятор языка C. Использование этой технологии позволяет писать масштабируемые параллельные программы, используя простое расширение языка C, при этом не подстраивая код под особенности конкретной модели GPU.

Согласно идеологии данной архитектуры GPU используется как сопроцессор к главному процессору (CPU). Программа на CUDA состоит из одного или более последовательных потоков, выполняющихся на CPU, и одного или более ядер (kernels), выполняющихся на GPU. Таким образом, часть программы, отличающаяся параллелизмом по данным, может быть выделена в отдельную процедуру, выполняющуюся на GPU.

Для поддержки CUDA на аппаратном уровне NVIDIA было разработано семейство продуктов GeForce, Quadro и Tesla, построенных на вычислительной архитектуре Fermi. Основой этой архитектуры является массив процессоров, разделенный на группы многопоточных мультипроцессоров SM (Streaming Multiprocessors), каждый из которых в свою очередь содержит по восемь последовательных SP (Scalar Processor) процессорных ядер.

В отличие от ранних графических ускорителей архитектура Fermi предоставляет набор инструкций, воспринимаемых SP-ядром, близкий к тому набору, который программист ожидает от ядра обычного процессора. В частности, это полная поддержка целочисленной арифметики и арифметики с плавающей точкой. Используемая в наших экспериментах GeForce 280GTX содержит 240 последовательных процессорных ядер SP, сгруппированных в 30 мультипроцессоров SM.

Каждое многопоточное последовательное процессорное ядро SP может поддерживать до 128 потоков, причем создание потока, планирование и распределение ресурсов происходят полностью на устройстве. Каждый поток программы на CUDA отображается непосредственно на физический поток, находящийся на GPU. Таким образом, используемый GPU поддерживает до 30 720 параллельных потоков.

Мультипроцессор SM работает согласно модели SIMD (Single Instruction, Multiple Data), т. е. все потоки мультипроцессора выполняют одну и ту же инструкцию. Обмен данными между SP-ядрами мультипроцессора может происходить при помощи высокоскоростной области памяти (shared memory), объем которой для GF280 составляет 16 КБ на каждый мультипроцессор. Следует отметить, что для анонсируемой GF300 эта цифра увеличится до 64 КБ. Для хранения больших объемов данных на устройстве имеется встроенная (global) память объемом 1 ГБ и скоростью доступа 140 ГБ/с. Для сравнения — скорость доступа к оперативной памяти для DDR3 составляет порядка 20 ГБ/с (для модулей PC3-19200). Через общую память предполагается также обмен данными между различными мультипроцессорами.

Для сравнительных экспериментов по приросту производительности CPU/GPU были использованы машина на базе процессора Intel Xeon E7330 с тактовой частотой 2.40 ГГц и многопоточные версии библиотек BLAS-ATLAS и LAPACK-ATLAS.

1.2. Особенности реализации метода подходящих аффинных подпространств для графических ускорителей

Основной концепцией реализации метода подходящих аффинных подпространств для GPU является перенос матрично-векторных операций на графический ускоритель. В алгоритме можно выделить несколько шагов, использующих такие операции. Во-первых, это проверка оптимальности построенного на i-м шаге внешнего цикла алгоритма приближения z^H, которая производится с помощью системы неравенств $\hat{x}^i z^H \geq \|z^H\|^2, i = 1, \ldots, N$. Введя матрицу X, столбцами которой являются векторы $\hat{x}^i$, данное условие можно переписать в виде $z^H X \geq \|z^H\|^2 e$, где $e = (1, \ldots, 1)$. Реализация такой проверки сводится к умножению матрицы на вектор и применению к полученному результату векторной версии алгоритма редукции [7] для выбора наибольшего значения элемента массива в полученном векторе.

Самой ресурсоемкой в вычислительном плане частью алгоритма является проекция нуля на базисное подпространство H_B, получаемое во внешнем цикле, где

$$H_B = \{x = \sum_{i \in B} \mu_i x^i, \sum_{i \in B} \mu_i = 1\}$$

и $B \subset \{1, 2, \ldots, N\}$ описывает набор базисных векторов. Количество элементов множества B обозначим через n_b. Вычисление барицентрических координат нового приближения в имеющемся базисе $X_B = \{\hat{x}^i, i \in B\}$ проводится для случая $n_b \leq n + 1$ по формуле

$$\mu = (X_B^T X_B)^{-1} e / e^T (X_B^T X_B)^{-1} e; \tag{5}$$

для случая $n_b = n + 1$ барицентрические координаты являются решением системы

$$X_B \mu = 0, \; e^T \mu = 0. \tag{6}$$

Последнее вызвано тем, что при $\|B\| = n+1$ матрица $A_B = X_B^T X_B$ является вырожденной. Основной по трудоемкости операцией здесь является обращение матрицы Грамма $X_B^T X_B$. Исходя из особенностей алгоритма для реализации этой процедуры был выбран алгоритм блочного обращения [8, 9], в котором для блочной матрицы

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

с невырожденной подматрицей A_{11} и невырожденной обратной матрицей $D = A_{22} - A_{21} A_{11}^{-1} A_{12}$ обратная матрица A^{-1} вычисляется как

$$A^{-1} = \begin{pmatrix} A_{11}^{-1} + A_{11}^{-1} A_{12} D^{-1} A_{21} A_{11}^{-1} & -A_{11}^{-1} A_{12} D^{-1} \\ -D^{-1} A_{21} A_{11}^{-1} & D^{-1} \end{pmatrix}. \tag{7}$$

Таким образом, вычисление обратной матрицы Грамма происходит следующим образом: на k-м шаге алгоритма предполагается, что уже есть обратная левая верхняя подматрица матрицы Грамма Γ_{k-1}^{-1} размерности $(k-1) \times (k-1)$. Применяя формулу (7) к подматрице Грамма Γ_k, получим обратную ей матрицу Γ_k^{-1}, затем переходим к шагу $k + 1$.

Использование блочного алгоритма позволяет снизить затраты на пересчет обратной матрицы Грамма, поскольку при добавлении нового вектора v в базис к уже существующей матрице Грамма Γ_k добавляются справа и снизу соответственно векторы

$\tau = X_b^T \cdot v$ и τ^T соответственно. Этот алгоритм также использует операции BLAS третьего уровня [10], эффективные для исполнения на GPU для матриц большой размерности.

В ходе вычислительных экспериментов было установлено, что для данной подзадачи предпочтительно применять смешанную модель вычислений CPU-GPU. Причиной этого могут быть относительно низкая производительность GPU на задачах малой размерности и большая начальная латентность вычислений (в среднем 8 мс для GF280). Поэтому пока подпространство H имеет размерность менее некоторого порогового значения вычисления ведутся полностью на CPU. Так как основной операцией для подзадачи нахождения проекции нуля на H является перемножение матриц, пороговое значение размерности подпространства оценивается для конкретной сборки из соображений прироста скорости операции матричного умножения и времени, затрачиваемого на копирование данных из оперативной памяти на память устройства. Для рассматриваемой сборки при размерности вектора 10^5 это составляет 128 векторов.

2. Численные эксперименты

Численные эксперименты проводились на двух типах тестовых множеств, показательных с точки зрения вычислительной нагрузки на алгоритм.

2.1. Тестирование на центрированных симплексах

2.1.1. Тест ПОЛНЫЙ-СИМПЛЕКС-С

Первый тип тестовых множеств представлял собой $(n-1)$-мерные симплексы с центром тяжести в начале координат, переведенные в n-мерное пространство сдвигом по n-й координате. Ниже приводится код на языке матрично-векторных вычислений octave (http://www.gnu.org/software/octave/), реализующий генерацию таких множеств:

```
rand("seed", 12345);
dim=1000, nvec=dim;
eps=0.01;
S=diag(100*rand(1,(dim-1)  ));
S=[S zeros(dim -1, 1)];
c=sum(S,2)/dim;
S -= c * ones(1,dim);
S = [S ; eps*ones(1, dim)];
```

При решении задачи для построенных таким образом симплексов характерно последовательное включение всех вершин в базис. Поэтому в алгоритме при решении задач (5) и (6) эффективно применяется блочное обращение матрицы Грамма $X^T X$, использующее полученную на предыдущем шаге обратную матрицу при добавлении новых векторов в базис. Для данного типа тестовых множеств были поставлены две серии численных экспериментов. Первая серия проводилась для множеств размерности $n \times n$, где n изменялось от 100 до 6000. На рис. 1 представлена зависимость времени нахождения проекции от величины n в логарифмических масштабах по обеим координатным осям. Видна степенная зависимость времени исполнения как CPU-, так и GPU-реализаций с общими показателями степени $n_{\mathrm{CPU}} = 1.96$ (кривая *1*), $n_{\mathrm{GPU}} = 1.25$ (кривая *2*). Улучшение сходимости в GPU-реализации можно объяснить заменой скалярных вычислений

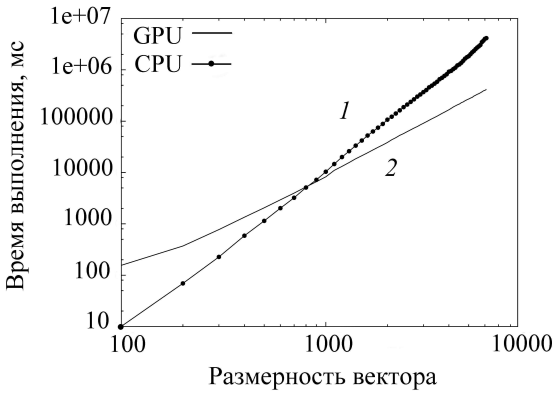

Рис. 1. Центрированные симплексы с изменением количества векторов и размерности пространства (тест ПОЛНЫЙ-СИМПЛЕКС-С)

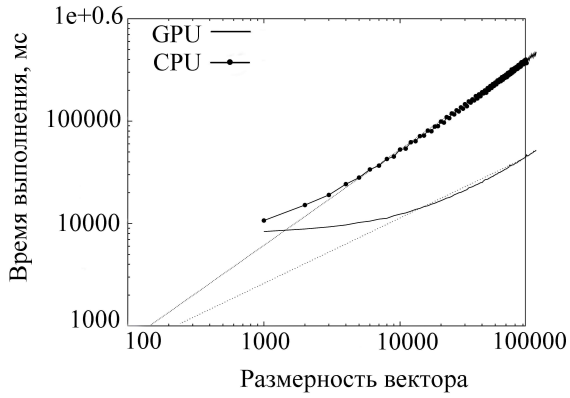

Рис. 2. Центрированные симплексы с фиксированным количеством векторов и изменением размерности пространства (тест ПОДСИМПЛЕКС-С)

векторными. Уменьшение показателя степени для GPU на 0.71, а не на 1 происходит в силу того, что физически на мультипроцессоре в один такт обсчитывается не весь вектор, а только его часть. Также видно, что для задач размерности меньше 1000×1000 более эффективной является CPU-реализация. Это можно объяснить небольшой утилизацией вычислительных ресурсов GPU и его латентностью вычислений.

2.1.2. Тест ПОДСИМПЛЕКС-С

Вторая серия численных экспериментов проведена для множеств, построенных по тому же алгоритму, но содержащих фиксированное количество векторов 10^3 (рис. 2). Изменялась только размерность векторов, которая варьировалась от 10^3 до 10^5. На этой серии тестов алгоритм показал практически линейную зависимость времени нахождения проекции от размерности векторов. Наибольший прирост производительности, составляющий примерно 9 раз, получен для векторов максимальной размерности порядка 10^5, что позволяет предположить дальнейшее увеличение преимуществ GPU с ростом размерности. Общие показатели степени для данной серии алгоритмов составляют 1.4 для GPU и 2 для CPU.

2.2. Тестирование на случайном множестве

Это множество тестов строилось таким образом, что исходный набор данных состоял из n векторов $\xi = (\xi_1, \xi_2, \ldots, \xi_m)$, построенных следующим образом:

$$\xi_i = \begin{cases} scale_1 \cdot \left(\varsigma_i - \dfrac{1}{2} \right), & \text{если } i = 1, 2, \ldots, n-1, \\ scale_2 \cdot \varsigma_i + shift, & \text{если } i = n, \end{cases}$$

где ς_i — независимые равномерно распределенные на $[0, 1]$ случайные величины, $scale_1$, $scale_2$ — константы масштабирования, $shift$ — константа сдвига. Для каждого вектора были проведены также масштабирование последней координаты с коэффициентом 10^{-3} и две серии тестов — с изменением размерности и количества векторов (ПОЛНЫЙ-СИМПЛЕКС-С) и с изменением только размерности (ПОДСИМПЛЕКС-С) (рис. 3).

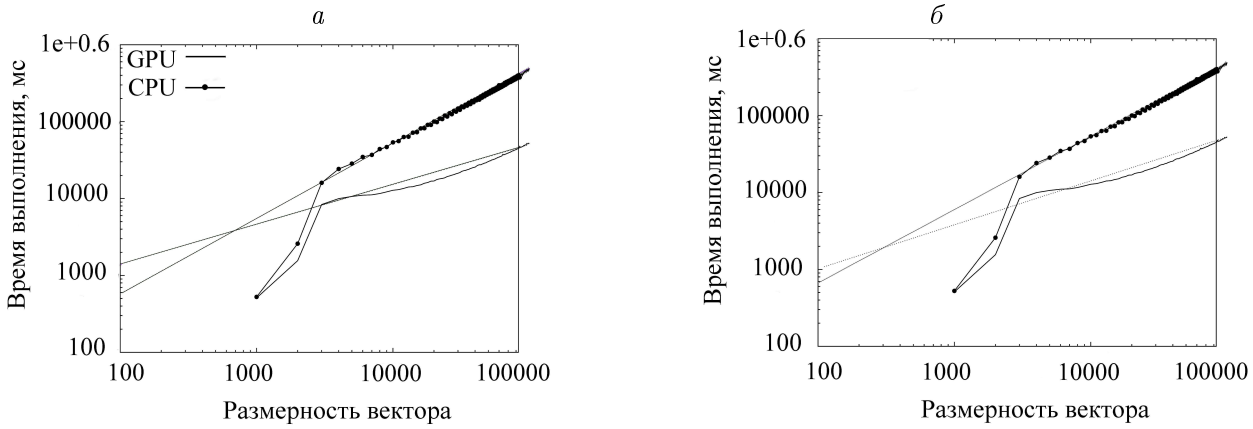

Рис. 3. Случайно сгенерированные симплексы с изменением размерности пространства и количества векторов — ПОЛНЫЙ-СИМПЛЕКС-С (*а*) и изменением размерности пространства и фиксированным количеством векторов — ПОДСИМПЛЕКС-С (*б*)

Код для octave, описывающий генерацию множеств, показан ниже:

```
rand("seed", 12345);
scale = [100,0.1];
shift = 500;
S = rand(dim - 1, nvec);
S = [scale(1) * (S - sum(S,2)/nvec * ones(1, nvec));
scale(2)*rand(1,nvec)+shift ];
```

Работа алгоритма на сгенерированных таким образом множествах характеризуется присутствием в подавляющем большинстве тестов фазы выведения векторов из базиса. При этом для матрицы Грамма, получившейся при изъятии вектора из базиса, становится невозможным применение ускорения обращения с использованием блочного алгоритма. Несмотря на это, алгоритм демонстрирует характер сходимости, схожий с предыдущими измерениями. Показатель сходимости для тестов ПОЛНЫЙ-СИМПЛЕКС-С равен 1.85 для GPU и 3.4 для CPU. Для тестов ПОДСИМПЛЕКС-С этот параметр составляет 1.87 для GPU и 3.14 для CPU.

Заключение

Результаты проведенных численных экспериментов подтверждают возможность практического применения GPU-реализации метода подходящих аффинных подпространств для решения задач проекции большой размерности (10^5). Основное ограничение на размерность задачи накладывает количество оперативной памяти на графическом ускорителе, что проявляется в том, что для тестовых задач, решение которых требовало использования всей памяти графического ускорителя, точка увеличения показателя степени в характеристике сходимости алгоритма так и не была достигнута. Применение реализации метода подходящих аффинных подпространств для графических ускорителей неэффективно для задач проекции малых размеров, что объясняется низкой утилизацией вычислительных ресурсов GPU. Как показали эксперименты, преимущества в производительности над CPU-версией начинается для задач с размерностью более 1500×1500.

Список литературы

[1] Нурминский Е.А. Численные методы выпуклой оптимизации. М.: Наука, 1991. 160 с.

[2] Boyd S., Vandenberghe S. Convex Optimization. Cambridge Univ. Press, 2004.

[3] CensorY., Jiang M., Louis A.K. Mathematical Methods in Biomedical Imaging and Intensity-Modulated Radiation Therapy (IMRT). Pisa, Italy: Birkhauser-Verlag, 2008. 522 p.

[4] Boyarchenkov A.S., Potashnikov S.I. Using graphic accelerators and CUDA technology for solving problems of molecular dynamic // Comput. Methods and Programming. 2009. Vol. 10, No. 1. P. 9–23.

[5] Stone S., Haldar J. Accelerating advanced MRI reconstructions on GPUs // 5th Conf. on Computing Frontiers. ACM. N.Y. USA, 2008. P. 261–272.

[6] Нурминский Е.А. О сходимости метода подходящих аффинных подпространств для решения задачи о наименьшем расстоянии до симплекса // Журн. вычисл. математики и матем. физики. 2005. Т. 45, вып. 11. С. 1996–2004.

[7] Lin H.X., Sips H.J. Parallel vector reduction algorithms and architectures // J. of Parallel and Distributed Comput. 1988. Vol. 5, No. 2. P. 103—130.

[8] Castillo M., Chan E., Francisco D. Making Programming Synonymous With Programming for Linear Algebra Libraries. Technical Report, 2008.

[9] Нейдеккер Х., Магнус Я.Р. Матричное дифференциальное исчисление с приложениями к статистике и эконометрике. М.: Физматлит, 2002. 495 с.

[10] Barrachina S., Castillo M. Evaluation and tuning of the level 3 CUBLAS for graphics processors // IPDPS. 2008. P. 1–8.

Поступила в редакцию 10 октября 2010 г.,
с доработки — 31 мая 2011 г.