Решение задачи поиска наименьшего расстояния до политопа с использованием графических ускорителей.

Е.А. Нурминский
Институт автоматики и процессов управления ДВО РАН
e-mail: nurmi@dvo.ru

П. Л. Поздняк
e-mail: pozpl@dvo.ru

31 мая 2011 г.

Аннотация

Представлена реализация метода подходящих аффинных подпространств для графических ускорителей. Описываются модификации алгоритма, особенности реализации для архитектуры графических ускорителей, а также результаты численных экспериментов. Получен прирост производительности по сравнению с обычным процессором почти в десять раз для задач размерностью 10^5

Ключевые слова — проекция, графические ускорители

1 Введение

В данной работе мы рассмотрим задачу проекции начала координат на n-мерный выпуклый многогранник X, заданный своими вершинами. Решение этой задачи широко используется при построении алгоритмов негладкой оптимизации [1], распознавании образов и автоматической классификации [2], компьютерной томографии и радиационной терапии [3]. Для практических приложений характерны большие размерности исходного пространства и большое количество вершин политопа X, а к алгоритмам решения этой задачи предъявляются жёсткие требования по вычислительной эффективности и точности. Эти обстоятельства мотивировали поиск новых алгоритмов решения задачи проекции и способов её реализации с использованием нестандартных вычислительных архитектур. В данной работе рассмотрена возможность ускорения вычислений с помощью графических ускорителей GPU (Graphics Processing Unit). Использование 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 позволяет увеличить скорость вычислений до 10-11 раз при решении задач высокой размерности.

2 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 идёт 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:

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

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

Определим 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 нульмерным 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 подаётся 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. При размерности пространства n трудоёмкость 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-ой итерации имеется базис I_k, соответствующий ему подходящий подсимплекс S_kи аффинное подпространство $H_k = \mathrm{aff}\,\{S_k\}$. Следующие действия описывают выполнение k-ой 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. Если это не так, то 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 $i_H \in \mathcal{M} \backslash I$. Образуем новое индексное множество $I_s = I_k \cup \{i_H\}$, полагаем $w_s = z^k$ и, занулив счётчик внутренних итераций $s = 0$, начинаем выполнение внутреннего цикла.

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 $H_{k+1} = \hat{H}_s^k, I_{k+1} = I_s$ 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 найденный таким образом μ_{max} принадлежит некоторой грани симплекса S_k т.е.

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

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

(d) Завершить выполнение итерации внутреннего цикла и вернуться к его началу.

4. Завершить выполнение итерации внешнего цикла и вернуться к его началу.

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

2.1 Вычислительные возможности GPU

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

В настоящей работе была предпринята попытка использования графических ускорителей для решения задачи 1 большой размерности. Современные графические ускорители обладают высокой степенью физического параллелизма, на сегодняшний день графические ускорители NVIDIA (http://www.nvidia.com) содержат от 8 до 240 ядер на одном чипе. Высокая степень физического параллелизма GPU позволяет эффективно решать задачи с параллелизмом по данным, с большим количеством арифметических операций по отношению к количеству обращений к памяти.

Для реализации высокоуровневого доступа к аппаратным возможностям GPU для разработчиков неграфических приложений компанией NVIDIA была разработана 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 этой 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, построенные на вычислительной архитектуре Fermi. Основой этой архитектуры является массив процессоров, разделённый на группы многопоточных мультипроцессоров SM (Streaming Multiprocessors), каждый из которых, в свою очередь, содержит по 8 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, причём создание потока, планирование и распределение ресурсов происходит полностью на устройстве. Каждый поток программы на CUDA отображается непосредственно на физический поток находящийся на GPU.Таким образом, используемый нами GPU поддерживает до 30720 параллельных потоков.

Мультипроцессор SM работает согласно модели SIMD (Single Instruction, Multiple Data), то есть 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 килобайт на каждый мультипроцессор. Стоит отметить, что для анонсируемой GF300 эта цифра увеличится до 64 килобайт. Для хранения больших объёмов данных на устройстве имеется встроенная (global) память объёмом 1 Гигабайт и скоростью доступа 140ГБ/сек. Для сравнения скорость доступа к оперативной памяти для DDR3 составляет порядка 20ГБ/сек (для модулей PC3-19200). Также через общую память предполагается обмен данными между различными мультипроцессорами.

Для сравнительных экспериментов по приросту производительности CPU/GPU, нами была использована машина, на базе процессора Intel Xeon E7330 с тактовой частотой 2.40ГГц. В экспериментах использовались многопоточные версии библиотек blas-atlas и lapack-atlas.

2.2 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 шаге внешнего цикла алгоритма приближения z^H, которое 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 $\hat{x}^i$, это условие можно переписать в виде $z^H X \geq \|z^H\|^2 e$, где $e = (1, \ldots, 1)$. Реализация этой проверки сводится к умножению матрицы на вектор и применения к полученному результату векторной версии алгоритма редукции [7] для выбора наибольшего значения в полученном векторе.

Самой ресурсоёмкой в вычислительном плане частью алгоритма является проекция нуля на базисное подпространство H_B,получаемого 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 $X_B = \{\hat{x}^i, i \in B\}$ осуществляются для случая $n_b \leq n + 1$ по формуле

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

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

$$\begin{array}{c} X_B \mu = 0, \\ e^T \mu = 0 \end{array} \tag{6}$$

Последнее вызвано тем, что при $\|B\| = n + 1$, матрица $A_B = X_B^T X_B$ является вырожденной. Основной по трудоёмкости 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

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

Таким образом, вычисление обратной матрицы Грамма происходит следующим образом: На k-ом 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 пересчёт обратной матрицы Грамма, так как при добавлении нового вектора v в базис к уже существующей матрице Грамма Γ_k добавляются справа и снизу векторы $\tau = X_b^T * v$ и τ^T соответственно. Также этот алгоритм использует операции BLAS 3-го уровня [10], эффективные для исполнения на GPU для матриц большой размерности.

В ходе вычислительных экспериментов было установлено, что для этой подзадачи выгодно применять смешанную модель вычислений CPU-GPU. Объяснением этого выбора может служить относительно низкая производительность GPU на задачах малой размерности, а также большая начальная латентность вычислений (в среднем 8 миллисекунд 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, то 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 нашей сборки при размерности вектора 10^5 это составляет 128 векторов.

3 Численные эксперименты

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

3.1 Тестирование на центрированных симплексах

3.1.1 Тест ПОЛНЫЙ-СИМПЛЕКС-С

Первый тип тестовых множеств представлял собой $(n-1)$- мерные симплексы с центром тяжести в начале координат, переведённые в n мерное пространство сдвигом по n-ой 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 используется 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 проведены две серии численных экспериментов. Первая серия численных экспериментов проводилась для множеств размерности $n \times n$ где n изменялось от 100 до 6000. На графике 1 показана зависимость времени нахождения проекции от величины n в логарифмических масштабах по общим координатным осям.

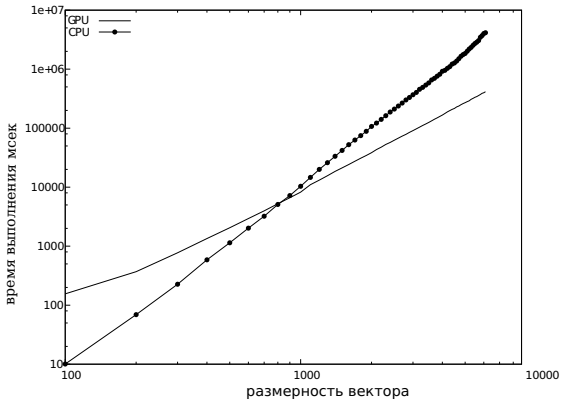

Рис. 1: Центрированные симплексы с изменением количества векторов и размерности пространства (тест ПОЛНЫЙ-СИМПЛЕКС-С).

Хорошо видна степенная зависимость времени исполнения, как CPU, так и GPU реализаций с общим показателем степени $n_{\mathrm{CPU}} = 1.96$, $n_{\mathrm{GPU}} = 1.25$. Улучшение показателя сходимости в GPU реализации можно объяснить заменой скалярных вычислений векторными. Уменьшение показателя степени для GPU на 0.71, а не на 1 происходит из за того, что физически на мультипроцессоре в один такт обсчитываемый не весь вектор, а только его часть. Также, можно заметить, что для задач размерности меньше чем 1000×1000 более эффективной является CPU реализация. Это можно объяснить маленькой утилизацией вычислительных ресурсов GPU, а также его латентностью вычислений.

3.1.2 Тест ПОДСИМПЛЕКС-С

Вторая серия численных экспериментов проведена для множеств, построенных по тому же алгоритму но, содержащих фиксированное количество векторов 10^3. Изменяется только размерность векторов, которая варьируется 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, удалось получить для векторов максимальной размерности порядка 10^5, что позволяет предположить дальнейшее увеличение превосходства GPU с ростом размерности. Общие показатели степени для этой серии алгоритмов составляют 1.4 для GPU и 2 для CPU.

3.2 Тестирование на случайном множестве

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

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

где ς_i независимые равномерно распределённые на $[0, 1]$ случайные величины, $scale_1$, $scale_2$ константы масштабирования, а $shift$ — константа сдвига. Также, для каждого вектора было проведе-

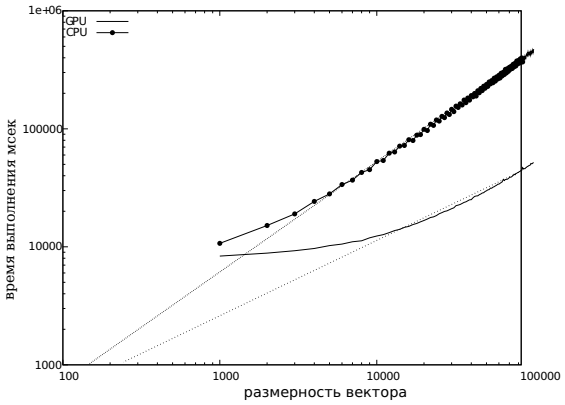

Рис. 2: Центрированные симплексы с фиксированным количеством векторов и изменением размерности пространства (тест подсимплекс).

но масштабирование последней координаты с коэффициентом 10^{-3} Здесь также были проведены две серии тестов с изменением размерности и количества векторов (ПОЛНЫЙ-СИМПЛЕКС-С), а также просто с изменением размерности (ПОДСИМПЛЕКС-С).

Код для octave (http://www.gnu.org/software/octave/), описывающий генерацию множеств приведён 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 показывает характер сходимости схожий с предыдущими измерениями. Показатели сходимости для тестов ПОЛНЫЙ-СИМПЛЕКС-С 1.85 для GPU и 3.4 для CPU. Для тестов ПОДСИМПЛЕКС-С показатели сходимости 1.87, 3.14 для GPU и CPU соответственно.

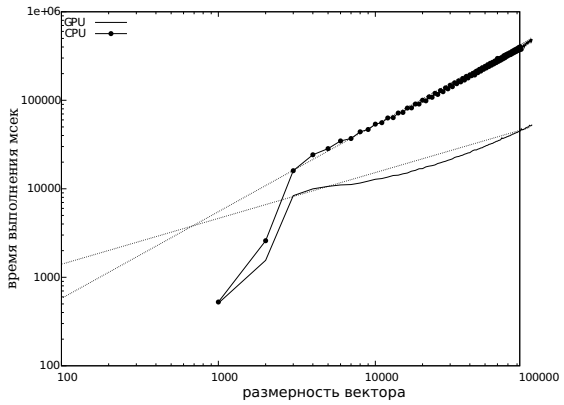

Рис. 3: Случайно сгенерированные симплексы с изменением количества векторов и размерности пространства (тест ПОЛНЫЙ-СИМПЛЕКС-С).

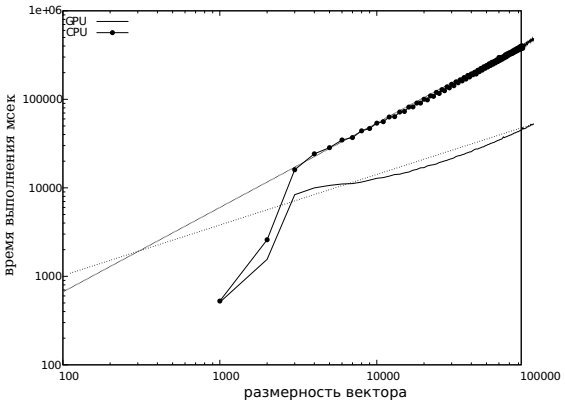

Рис. 4: Случайно сгенерированные симплексы с фиксированным количеством векторов и изменением размерности пространства (тест ПОДСИМПЛЕКС-С).

3.3 Заключение

Результаты численных экспериментов подтверждают возможность практического применения GPU реализации метода подходящих аффинных подпространств для решения задач большой размерности (10^5). При этом, основное ограничение на размерность задачи накладывает количество оперативной памяти на графическом ускорителе. Это утверждение вытекает из того факта, что для тестовых задач, решение которых требовало задействования всей памяти устройства, не было достигнуто точки увеличения степени сходимости алгоритма. Эффективное применение реализации МАП для графических ускорителей снизу ограничивается низкой утилизацией вычислительных ресурсов. Согласно нашим экспериментам превосходство в производительности над CPU версией, начинается для задач с размерностью более чем 1500×1500.

Список литературы

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

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

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

[4] Boyarchenkov A.S., Potashnikov S.I. Using graphic accelerators and CUDA technology for solving problems of molecular dynamic//Computation Methods and Programming. 2009. Vol. 10, N 1. C. 9–23.

[5] Stone S., Haldar J. Accelerating advanced mri reconstructions on GPUs.//5th conference on Computing frontiers. 2008. P. 261–272.

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

[7] Lin H. X., Sips H. J. Parallel vector reduction algorithms and architectures//Journal of Parallel and Distributed Computing. 1988. Vol. 5. N 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.