УДК 517.958:532

ИНТЕГРИРОВАНИЕ УРАВНЕНИЯ ПУАССОНА С ИСПОЛЬЗОВАНИЕМ ГРАФИЧЕСКОГО ПРОЦЕССОРА ТЕХНОЛОГИИ CUDA

Н. М. Евстигнеев[1]

Рассмотрена параллельная реализация циклического многосеточного метода, решающего краевую задачу для уравнения Пуассона в $\mathbb{R}^3$ на графических процессорах с применением платформы NVIDIA CUDA. Выполнено сопоставление результатов расчета задач на графическом процессоре с аналитическим решением краевой задачи Дирихле и с численным решением на ЦПУ смешанной краевой задачи. Сопоставление скорости расчета на видеокарте NVIDIA GeForce 8800 GTX со скалярной версией на процессоре AMD Athlon 64X2 4800+ показало ускорение до 200 раз для дискретной области в 1 000 000 элементов, а сопоставление с рабочей станцией на двух четырехъядерных процессорах Intel(R) Xeon(R) 2.33 ГГц показало ускорение в 40 раз. Работа поддержана РФФИ (коды проектов 08–07–00074а и 06–07–89047а) и программой ОНИТ РАН (проект 1.12).

Ключевые слова: уравнение Пуассона, циклический многосеточный метод, параллельные вычисления, графические процессоры, технология CUDA.

1. Введение. В настоящее время развитие параллельных вычислительных технологий достигло значительного прогресса. Для выполнения сложных математических расчетов все чаще применяются графические адаптеры (GPU, Graphics Processing Unit — ГПУ, по аналогии с ЦПУ). Применение графических адаптеров позволяет заметно ускорить расчеты на обычных персональных компьютерах малой стоимости за счет использования общей памяти и значительного параллелизма. Исследования в данной области значительно интенсифицировались [7] после появления среды разработки высокого уровня CUDA (Compute Unified Device Architecture) на языке C компании NVIDIA [8]. Поскольку целью настоящего исследования является разработка оптимального алгоритма численного решения краевой задачи для уравнения Пуассона в $\mathbb{R}^3$ на графическом адаптере, более детальную информацию о CUDA можно найти в [8] и другой сопутствующей документации, доступной на сайте компании NVIDIA.

Для численного решения уравнения Пуассона используется каскадный многосеточный метод, позволяющий относительно просто реализовать его на CUDA-языке и, с другой стороны, имеющий вычислительные затраты порядка $O\left(N \ln \varepsilon^{-1}\right)$, где N — общее количество элементов и ε — точность вычислений.

2. Постановка задачи. Рассмотрим следующую краевую задачу со смешанными граничными условиями:

$$LU = F \quad \text{в} \quad \Omega, \quad \Omega \in \mathbb{R}^3, \quad \partial\Omega_0(U) = U_0, \quad \partial\Omega_1(U) = \frac{\partial U}{\partial n_{\partial\Omega}}. \tag{1}$$

Здесь L — эллиптический оператор вида $LU = \sum\limits_{i,j} \dfrac{\partial}{\partial x_i}\left(a_{ij}\dfrac{\partial U}{\partial x_j}\right)$; a_{ij} удовлетворяют условию эллиптичности $a_{ij} = a_{ji}$; существует такая постоянная $\alpha > 0$, что $\sum a_{ij}\xi_i\xi j \geqslant \alpha \sum\limits_i \xi_i^2$, где $\xi \in \mathbb{R}^3$ и $i,\ j \in [1,2,3]$; F — известная функция; неизвестная функция U принадлежит $\mathbb{C}^2$.

3. Численное решение. Рассмотрим численное решение задачи (1). Введем сеточное пространство выпуклых кубов $\Omega_h \in [1,\ldots,M;1,\ldots,N;1,\ldots,K]$ и запишем (1) в дискретной форме

$$L_hU_h = F_h \quad \text{в} \quad \Omega_h, \quad \Omega_h \in \mathbb{R}^3, \quad \partial\Omega_{h0}(U) = U_0, \quad \partial\Omega_{h1}(U) = \frac{\partial U}{\partial n_{\partial\Omega}}. \tag{2}$$

Индекс h означает принадлежность к сеточному пространству; M, N, K — размерность задачи в $\mathbb{R}^3$. Сторона куба вычисляется как $\Delta h_j = \dfrac{1}{X_j}$; $X_j = \{M,N,K\}$; $j \in [1,2,3]$.

[1] Институт системного анализа РАН, просп. 60-летия Октября, 9, 117312, Москва; ст. науч. сотр., e-mail: evstigneevNM@yandex.ru

Будем аппроксимировать дифференциальный эллиптический оператор трехточечным и пятиточечным конечно-разностными шаблонами вида

$$L_h^*(\lambda) = \sum_{j=1}^{3} \frac{\left(\lambda_{i+1} - 2\lambda_i + \lambda_{i-1}\right)_j}{(\Delta h)_j^2} + O\left(\Delta h^2\right), \tag{3}$$

$$L_h^{**}(\lambda) = \sum_{j=1}^{3} \frac{\left(-\lambda_{i+2} + 16\lambda_{i+1} - 30\lambda_i + 16\lambda_{i-1} - \lambda_{i-2}\right)_j}{(12\Delta h)_j^2} + O\left(\Delta h^4\right). \tag{4}$$

Введем конечную систему сеток в следующей последовательности при условии, что сторона Ω_h равна 1:

$$S_1(M,N,K) \leftarrow S_2\left(\frac{M}{2},\frac{N}{2},\frac{K}{2}\right) \leftarrow S_4\left(\frac{M}{4},\frac{N}{4},\frac{K}{4}\right) \leftarrow \ldots \leftarrow S_q\left(\frac{M}{q},\frac{N}{q},\frac{K}{q}\right). \tag{5}$$

Таким образом, сторона куба на каждой из сеток имеет следующую последовательность: $\Delta h_i \leftarrow 2\Delta h_i \leftarrow 4\Delta h_i \leftarrow \ldots \leftarrow q\Delta h_i$. Здесь q — уровень сетки, определяемый как $q = 1, 2, 4, \ldots, 2n$, где $n \in N$ — натуральные числа. Введем энергетическую норму для сеточного пространства Ω_h на сетке уровня q:

$$\left\|\{U\}\right\|^q = \sqrt{q^3 \Delta h_x \Delta h_y \Delta h_z \sum_{i=1;\, j=1;\, k=1}^{i=M/q;\, j=N/q;\, k=K/q} \left(LU_{ijk}, U_{ijk}\right)}. \tag{6}$$

К полученной системе сеток будем применять каскадный многосеточный метод, впервые предложенный в [1] и проанализированный в [2], где доказано, что скорость расчета составляет $O(CN)$, где $C > 1$ — произвольная константа, пропорциональная скорости сходимости, и N — общее количество расчетных ячеек. Кроме того, каскадный многосеточный метод, в применении к эллиптическим и параболическим дифференциальным операторам, значительно развит в работах [3, 4] где доказана сходимость и устойчивость метода и оценена скорость расчета.

Каскадный многосеточный метод (КМ) в основном состоит из двух основных стадий для каждой из сеток (5).

1. Решение задачи (2) с использованием аппроксимации L вида (3) или (4) на сетке S_i до тех пор, пока ошибка решения не станет меньше энергетической нормы (6) для данной i-й сетки.

2. Проекция полученного решения задачи (2) для функции U в S_i на функцию U в S_{i-1}, т.е. проекция решения с более грубой сетки на более точную.

Таким образом, схематически КМ можно показать как (рис. 1) последовательное движение от сетки порядка q к сетке порядка 1, т.е. к самой точной, исходной сетке, на которой изначально и рассматривается задача (2).

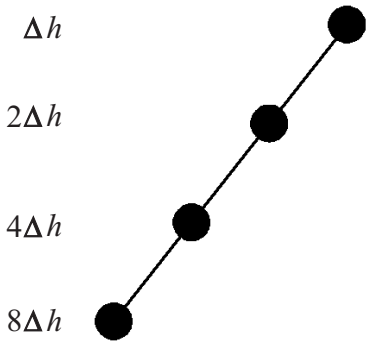

Рис. 1. Четырехступенчатый каскадный многосеточный метод

Для стадии 1 применяются итерационные методы, обладающие свойством сглаживать высокочастотные компоненты функции U, в том числе метод простой итерации Якоби, метод Гаусса–Зейделя, метод релаксации и метод сопряженных градиентов. Кроме того, для начальных шагов на грубой сетке применяются прямые методы. Для стадии 2 используется линейная интерполяция или интерполяция более высокого порядка.

В работах [5, 6] для эллиптических операторов в $\mathbb{R}^3$ было определено, что выбор методов нахождения решения на каждом из шагов, а также выбор типа проекции с одной сетки на другую значительно влияют на скорость расчета, причем теоретического обоснования оптимальности использования того или иного метода на стадиях 1 и 2, к сожалению, не было найдено.

Для решения задачи (2) проведем численный анализ получаемых данных при использовании аппроксимаций (3) и (4), различных методов нахождения решения для U и различных методов проекции и выберем наиболее быстрое сочетание из полученных результатов.

4. Оптимизация цикличного многосеточного метода. В качестве методов решения СЛАУ, полученных в результате применения аппроксимаций (3) и (4), сравним следующие методы: метод Гаусса–Зейделя (ГЗ), метод сопряженных градиентов (СГ) и метод верхней релаксации (ВР). Эти методы решения СЛАУ общеизвестны [10] и в данной работе не рассматриваются.

Проектирование сеточных функций с уровня q на уровень $q-1$ определяется билинейной аппроксимацией и аппроксимацией четвертого порядка. Пусть имеются две сетки уровня $q-1$ и уровня q в соответствии с (5) и пусть $(i,j,k) \in S_q$. Введем оператор I интерполяции, определенным образом отображающий сеточные функции с уровня q на уровень $q-1$. Тогда оператор I билинейной интерполяции $U_h^q \xrightarrow{I} U_h^{q-1}$, $U_h^q \in S_q$, $U_h^{q-1} \in S_{q-1}$ запишется как

$$\begin{aligned}
&U^{q-1}_{ijk} = U^q_{ijk},\\
&U^{q-1}_{i\pm1/2,jk} = \frac{1}{2}\big(U^q_{ijk} + U^q_{i\pm1,jk}\big),\\
&U^{q-1}_{i\pm1/2,j\pm1/2,k} = \frac{1}{4}\big(U^q_{ijk} + U^q_{i\pm1,jk} + U^q_{i,j\pm1,k} + U^q_{i\pm1,j\pm1,k}\big),\\
&U^{q-1}_{i\pm1/2,j\pm1/2,k\pm1/2} = \frac{1}{8}\big(U^q_{ijk} + U^q_{i\pm1,jk} + U^q_{i,j\pm1,k} + U^q_{i\pm1,j\pm1,k} + U^q_{ij,k\pm1} + U^q_{i\pm1,j,k\pm1} +\\
&\qquad + U^q_{i,j\pm1,k\pm1} + U^q_{i\pm1,j\pm1,k\pm1}\big),
\end{aligned} \tag{7}$$

а при четвертом порядке запишется как [9] (приведено только для одной точки):

$$\begin{aligned}
U^{q-1}_{ijk} = \frac{1}{24}\big(&12U^q_{ijk} + U^q_{i+1,j+1,k} + U^q_{i+1,j-1,k} + U^q_{i-1,j-1,k} + U^q_{i-1,j+1,k} + U^q_{i+1,j,k+1} +\\
&+ U^q_{i+1,j,k-1} + U^q_{i-1,j,k-1} + U^q_{i-1,j,k+1} + U^q_{i,j+1,k+1} + U^q_{i,j+1,k-1} +\\
&+ U^q_{i,j-1,k-1} + U^q_{i,j-1,k+1}\big).
\end{aligned} \tag{8}$$

Алгоритм расчета может быть разбит на следующие этапы.

1) Задается система сеток (5).

2) Ставится краевая задача (2) на всем семействе сеток (5).

3) С помощью прямого метода находится решение задачи (2) на самой грубой сетке уровня q.

4) С помощью оператора интерполяции (7) или (8) значение сеточной функции переносится на уровень сетки $q-1$.

5) С помощью одного из итерационных методов (ГЗ, СГ, ВР) находится решение задачи (2), тем самым убираются самые высокочастотные невязки, полученные на сетке уровня q. Итерации продолжаются до тех пор, пока невязка не станет равна энергетической норме (6) на данной сетке.

6) Переход, если не достигнут уровень сетки 1, на этап 4 алгоритма, при этом $q = q-1$.

Для проведения расчета выбраны две краевые задачи вида (2) — задача Дирихле во всей области (все границы имеют индекс 0 в (2)) и задача Неймана во всей области за исключением одной из сторон, на которой поставлено условие Дирихле.

Первая краевая задача для (2) рассматривается для верификации алгоритма, так как при $F = 0$ в (2) (трехмерное уравнение Лапласа) она имеет аналитическое решение для случая определения интенсивности магнитного поля в прямоугольном куске ферромагнетика, намагниченного с двух противоположных торцов магнитным полем B_0 интенсивностью 1 Тесла, относительно центральной плоскости $y0$. Геометрия, представленная прямоугольником, определяется размерами $x1 \leqslant x \leqslant x2$, $|y| \geqslant y0$ и $z1 \leqslant z \leqslant z2$. На плоскостях ставится граничное условие Дирихле ($U = 1$ на плоскостях, параллельных $y0$, и 0 на всех остальных). Тогда вектор-функция магнитного поля $\boldsymbol{B}(x,y,z)$ определяется как

$$\begin{aligned}
B_y &= \frac{B_0}{4\pi}\sum_{i,j=1}^{2}(-1)^{i+j}\left(\operatorname{arctg}\left(\frac{X_i Z_j}{Y_+ R^+_{ij}}\right) + \operatorname{arctg}\left(\frac{X_i Z_j}{Y_- R^-_{ij}}\right)\right);\\
B_a &= \frac{B_0}{4\pi}\sum_{i,j=1}^{2}(-1)^{i+j}\ln\left(\frac{a_j + R^-_{ij}}{a_j + R^+_{ij}}\right); \quad a = \{X, Z\},
\end{aligned} \tag{9}$$

где $X_i = x - x_i$, $Z_j = z - z_j$, $Y_\pm = y_0 \pm y$ и $R^\pm_{ij} = \sqrt{X_i^2 + Z_j^2 + Y_\pm^2}$.

Вторая краевая задача рассматривается как наиболее затратная по времени и представляет собой нахождение решения задачи (2) для случая, когда на одной из граничных плоскостей задано условие Дирихле, а на всех остальных — условие Неймана. Такая ситуация очень часто встречается при решении

уравнений Навье–Стокса для несжимаемой жидкости методами, основанными на проекции скорости при нахождении поля давления, представляющего себой уравнение Пуассона.

С целью нахождения оптимального метода расчета и интерполяции выполнена серия расчетов для второй задачи с помощью рассмотренного метода, реализованного на языке C++ в серийном режиме с одним ядром двухъядерного процессора AMD Athlon 64X2 4800+.

Результаты показаны на рис. 2. Как можно видеть, наиболее выигрышным по времени оказалось использование комбинации методов (4), (8) и (3), (4). Для всех протестированных сеток с разным количеством кубов оптимальным оказалось применение (4), (8) на сетках с уровня q до уровня 4 и (3), (7) с уровня 4 до уровня 1 в последовательности сеток (5). Результаты для максимального уровня q подтвердили оценки, сделанные в [3], в соответствии с которыми $q \leqslant 10$.

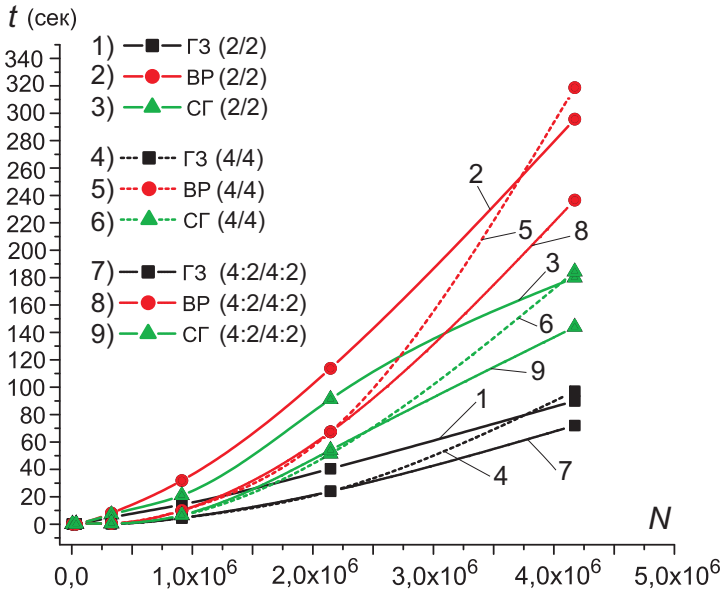

Рис. 2. Сравнение временны́х затрат в зависимости от методов нахождения решения и интерполяции на каждом шаге: 2/2 — используется схема (3) для расчета и схема (7) для интерполяции; 4/4 — схема (4) для расчета и схема (8) для интерполяции; 4:2/4:2 — комбинация для сеток уровней $q, \ldots, k$ (схема 4/4) и для сеток уровней $k, \ldots, 1$ (схема 2/2)

5. Реализация на графическом процессоре с помощью CUDA. В соответствии с документацией [8] была составлена программа, в которой ядрами решения на ГПУ являлись процедуры нахождения решения и интерполяции, при этом использовалась глобальная память устройства. В связи с тем, что ГПУ представляет собой параллельную вычислительную систему, для нахождения решения задачи (2) на каждом из этапов циклического многосеточного метода применялась процедура цветовой раскраски кубов для метода Гаусса–Зейделя.

Сама по себе процедура нахождения значений функции U в узлах сетки строится таким образом, чтобы максимально повысить скорость расчета. Для этой цели внутри процедуры вычисления, обозначенной как `GAUS_SEIDEL`, вводится массив разделяемой памяти. Скорость доступа к разделяемой памяти превосходит скорость доступа к глобальной на два порядка [8, с. 55]; таким образом, применение разделяемой памяти значительно ускоряет расчет. С другой стороны, размер разделяемой памяти не может быть больше отведенного; следовательно, необходимо выполнять блочную загрузку данных в разделяемую память, поскольку размеры задачи значительно превышают размеры разделяемой памяти. При этом [8, с. 59] наиболее эффективно использовать двумерную индексацию (по параллельным потокам индексы i и j). Это сокращает время запроса на получения индекса по сравнению с трехмерной индексацией примерно в 1,5–1,8 раза. В результате на сетку потоков отображаются только два цикла, в примере — по индексам i и j. Индекс k для трехмерной задачи вычислялся из известных индексов i и j и размера задачи. В отличие от версии метода Гаусса–Зейделя для ЦПУ, ядро для ГПУ можно записать в виде следующего фрагмента:

```
// часть алгоритма Гаусса Зейделя, отвечающая за четные кубы
__global__ void kernel_poisson_red(int _M, int _N, int _K, float* x, float* x0)
// Определение ядра обработки в GPU
{       int i = blockIdx.x * blockDim.x + threadIdx.x;
         int j = blockIdx.y * blockDim.y + threadIdx.y; // y увеличен на размер
задачи в направлении индекса K
         int k=trs(_M,_N,_K,i,j); // определение недостающего индекса макросом

        if((i + j + k) % 2 == 0 ) { // если куб четный
                //
                GAUS_SEIDEL // выполняется вычисление по всем нечетным кубам
         }
```

```
        GAUS_SEIDEL_BC // определение граничных условий
        __syncthreads(); // синхронизация потоков для исключения конфликтов
}
```

Таким образом, все три цикла отображаются на графический процессор. Кроме того, в соответствии с описанием [8, с. 34], все вычисления должны проводиться с данными в видеопамяти и количество перегрузок данных из оперативной памяти компьютера в видеопамять должно быть минимальным. В данном случае алгоритм реализован таким образом, что все вычисления производятся с данными в видеопамяти и выгрузка данных в оперативную память компьютера производится только после расчета задачи на сетке уровня 1 при достижении заданной точности.

В нашей работе использовался графический адаптер NVIDIA GeForce 8800 GTX с 768Mb оперативной видеопамяти.

С целью сопоставления корректности расчета на ГПУ проведено сравнение данных с (9) для краевой задачи Дирихле и с рядом других задач для (2), полученных при расчете на ЦПУ. Результат решения задачи (9) на сетке $50 \times 50 \times 50$ показан на рис. 3. Максимальное отличие от аналитического решения составляет 2.5×10^{-6}, что показывает достаточно высокую точность вычислений. Отличие результата между ГПУ и ЦПУ имеет порядок 10^{-8}.

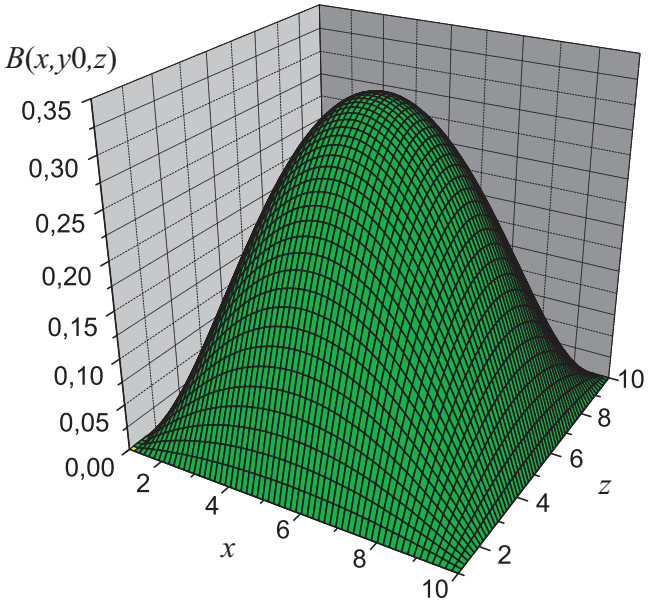

Рис. 3. Результаты расчета модельной задачи распределения составляющей магнитного поля в ферромагнетике (задача 1). Сетка — численное решение на сетке $50 \times 50 \times 50$, заливка — аналитическое решение (9)

Для определения ускорения по сравнению с ЦПУ было проведено решение (2) для краевой задачи Неймана во всей области, кроме одной граничной плоскости с условием Дирихле. Вычисления на ЦПУ велись с одинарной точностью, и там, где было возможно, использовались функции одинарной точности, такие как `absf()` или `maxf()`, а компилятор выполнялся с ключом `/O2`, оптимизирующим код на максимальную скорость. Это позволило исключить влияние использования двойной точности при определении ускорения. Результаты экспериментов показаны на рис. 4. Ускорение на 10^6 ячеек составляет 200 раз относительно ЦПУ с использованием одного ядра. Расчет задачи, состоящий из 19×10^6 ячеек, занимает 11.2 секунды на ГПУ. Определение ускорения на ГПУ по сравнению с восьмиядерным ЦПУ для рассматриваемой задачи выполнено на рабочей станции в ИСА РАН, лаборатория № 11-3 "Хаотические динамические системы". Рабочая станция имеет два четырехъядерных процессора Intel(R) Xeon(R) с тактовой частотой 2.33 ГГц. Распараллеливание версии программы для ЦПУ выполнялось средствами OpenMP. Версия программы для ГПУ запускалась на той же системе, что и в предыдущих тестах. Результаты экспериментов приведены на рис. 5.

Как можно видеть, прирост скорости расчета падает при увеличении задействованных ядер ЦПУ, доходя до 41 раза по сравнению с восемью ядрами. Эта оценка может показать, насколько эффективнее выполнять расчеты на ГПУ по сравнению с многоядерными вычислительными станциями в смысле их стоимости и затрат времени.

Для определения производительности в MFLOPS необходимо просчитать количество тактов на одну команду алгоритма, встроенного в ядро ГПУ (`__global__`-подпрограммы), для процедур метода Гаусса–Зейделя и метода проекции с одного слоя сетки на другую и поделить на время выполнения каждой из процедур. Алгоритм расчета времени циклов взят из [12]. В соответствии с [8, с. 49], было выписано количество тактов на каждую из операций и проведены замеры времени. Для ГПУ (NVIDIA 8800 GTX) теоретическая максимальная производительность составляет 345,6 GFLOPS [8] ([128 процессоров]×[1,35 ГГц тактовая частота]×[2 арифметико-логических устройства, работающих параллельно]). При применении функции `clock_t clock()` [8, с. 26] для задачи, состоящей из 0.5×10^6 ячеек, производительность составила 117.13 GFLOPS. Дальнейшее увеличение размерности задачи приводит к снижению производительности до 51.28 GFLOPS для задачи в 19×10^6 ячеек.

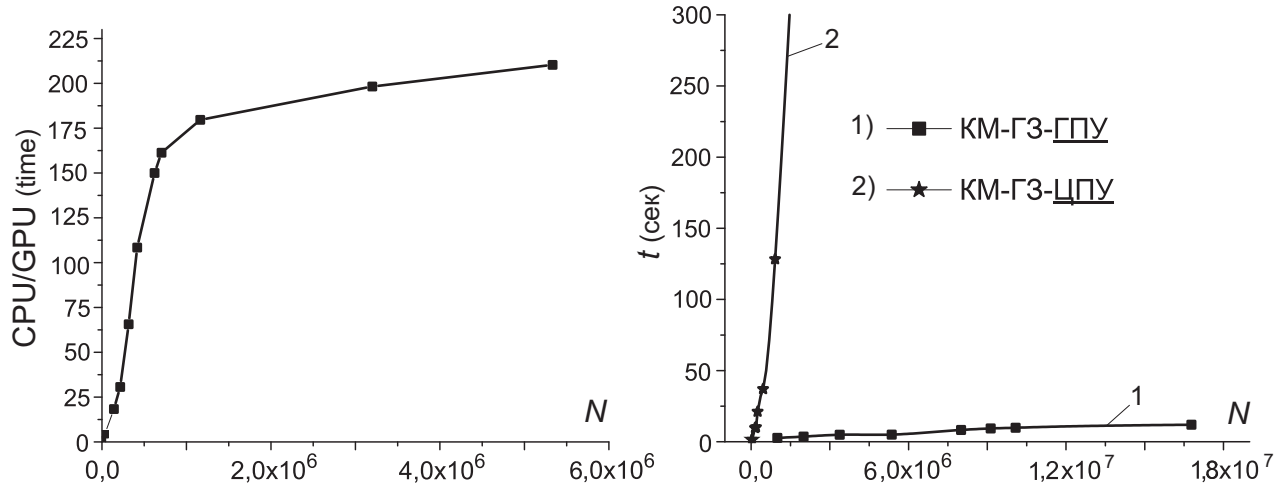

Рис. 4. Ускорение расчета на ГПУ во временны́х затратах по отношению к ЦПУ и сопоставление временных затрат на расчет задачи (2) на ЦПУ и ГПУ

6. Заключение. Рассмотрено и численно подтверждено применение циклического многосеточного метода для нахождения численного решения трехмерного уравнения Пуассона на прямоугольной сетке. Определена оптимальная последовательность изменения порядка аппроксимации дифференциального оператора и изменения порядка интерполяции в зависимости от размера сетки. Полученный алгоритм адаптирован для расчета трехмерной краевой задачи для уравнения Пуассона на видеоадаптере с использованием системы CUDA компании NVIDIA. Путем специальной комбинации индексов массива и организации вычислений в видеопамяти удалось добиться 200-кратного ускорения по сравнению с аналогичной задачей, выполняемой на ЦПУ при незначительном снижении точности расчета. Кроме того, выполнено сравнение ГПУ версии программы с восьмиядерной рабочей станцией. Определено, что скорость расчета задачи на ГПУ в 41 раз выше при использовании всех восьми ядер. В настоящее время автором проводится работа по переносу среды решения задач пространственной нестационарной гидродинамики и газовой динамики на платформу графических адаптеров с целью ускорения проведения расчетов, а также применения каскадного многосеточного метода на неструктурированной сетке.

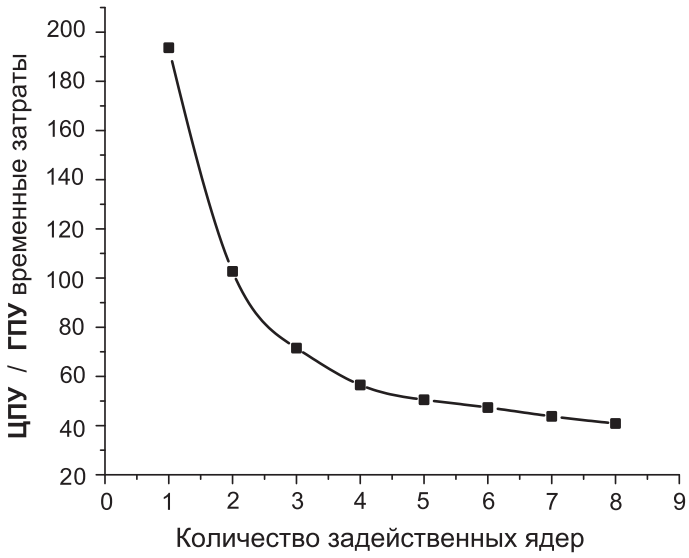

Рис. 5. Ускорение ГПУ по сравнению с многоядерным ЦПУ, в зависимости от количества ядер на ЦПУ

СПИСОК ЛИТЕРАТУРЫ

1. *Deuflhard P., Leinen P., Yserentant H.* Concepts of an adaptive hierarchical finite element code // Impact Comput. Sci. Eng. 1989. **1**. 3–35.
2. *Shaidurov V.V.* Some estimates of the rate of convergence for the cascadic conjugate gradient method // Comput. Math. Appl. 1996. **31**. 161–171.
3. *Shi Z., Xu X.* Cascadic multigrid method for elliptic problems // East-West J. Numer. Math. 1999. **3**. 199–209.
4. *Braess D., Dahmen W.* A cascadic multigrid algorithm for the Stokes equation // Numer. Math. 1999. **82**. 179–191.
5. *Shi Z., Xu X.* Cascadic multigrid method for parabolic problems. Preprint, 1999.
6. *Bornemann F.A., Krause R.* Classical and cascadic multigrid — a methodical comparison // Proc. of the Ninth International Conference on Domain Decomposition. Bergen: Domain Decomposition Press, 1998. 64–71.

7. *Боярченков А.С., Поташников С.И.* Использование графических процессоров и технологии CUDA для задач молекулярной динамики // Вычислительные методы и программирование. 2009. **10**, № 1. 13–27.
8. http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf
9. *Hyman J.M.* Mesh refinement and local inversion of elliptic differential equations // J. Comput. Phys. 1977. **23**. 124–134.
10. *Формалев В.Ф., Ревизников Д.Л.* Численные методы. М.: Физматлит, 2006.
11. *Degenhardt R., Berz M.* High accuracy description of the fringe field in particle spectrographs // Nuclear Instruments and Methods. 1999. **A427**, 151–156.
12. http://www.roylongbottom.org.uk/index.htm#anchorNew

Поступила в редакцию
16.03.2009