РЕАЛИЗАЦИЯ МЕТОДА ЯКОБИ НА GPU ДЛЯ МАССИВОВ С РАЗМЕРОМ, ПРЕВЫШАЮЩИМ ОБЪЕМ GPU-ПАМЯТИ

А.В.Кочуров[1,2], Д.Г.Воротникова[1,2], Д.Л.Головашкин[1]

[1]Институт Систем Обработки Изображений РАН (ИСОИ РАН)
[2]Самарский государственный аэрокосмический университет им. академика С.П. Королёва (национально исследовательский университет)

В статье предлагается метод для расширения размерности сетки, которая может быть обработана на GPU неявным конечно-разностным методом. Подход основан на методе пирамид. Предлагается математическая модель для предсказания длительности вычислений. Эта модель позволяет находить оптимальные параметры алгоритма. В статье представляются результаты вычислительного эксперимента, подтверждающего, что модель обладает достаточной точностью для предсказания оптимальных параметров.

Введение

Вопросам вычисления на графических вычислительных устройствах (GPU) и, в частности, решения сеточных уравнений на GPU, посвящено множество статей [1, 2].

Основной недостаток методов, приведенных в этих статьях, состоит в требовании наличия всей сеточной области в видеопамяти для вычисления. Известные библиотеки для GPU с открытым исходным кодом, которые решают эти задачи[3], обладают таким же ограничением. Следует заметить, что современные GPU не подразумевают расширения памяти. В то же время типичный компьютер оснащен ОЗУ, в разы большим, чем видеопамяти. В таблице 1 приведены оценки размера сетки, которые могут обрабатываться CPU и GPU на некоторых системах.

Таблица 1. Максимальный размер сеточной области, которые могут обработать узлы некоторых кластеров

Кластер	ОЗУ	Размер сетки для ОЗУ*	Видеопамять	Размер сетки для GPU*
[9]	22 Гб	$1400 \times 1400 \times 1400$	6 Гб	$930 \times 930 \times 930$
K-100**	96 Гб	$2300 \times 2300 \times 2300$	3×2.5 Гб	$1000 \times 1000 \times 1000$
[10]	до 256 Гб	$3200 \times 3200 \times 3200$	До 4×8 Гб	$1600 \times 1600 \times 1600$
* На основе оценок из [12]			** K-100 это кластер [11]	

На практике широкомасштабные задачи требуют сетки размером в тысячи узлов по всем измерениям. Подобные задачи возникают во многих областях, где применяются разностные схемы, таких как разработка микронных оптических элементов с нанометровыми неоднородностями[4], симуляция распространения света методом FDTD[5], моделирование сейсмических волн [6]. Существует определенная потребность в методах решения разностных уравнений на GPU для сеточных областей больших, чем может вместить видеопамять.

Для явных разностных схем известно несколько подходов, позволяющих применять
GPU для решения подобных задач, как то: использование нескольких GPU или вычислительного кластера, оснащенного GPU [6] или использование алгоритмических уловок, позволяющих за счёт дублирования вычислений сократить требования к памяти путем хранения основного массива данных в ОЗУ и оперативной пересылки необходимых для вычислений блоков данных на GPU непосредственно перед вычислениями [7]. Обобщенная версия последнего подхода была исследована авторами в прошлом году [8] и продемонстрировала впечатляющие характеристики.

И хотя явные разностные схемы во многих случаях могут обеспечить требуемую точность, неявные разностные схемы как правило обеспечивают более быструю сходимость и накладывают меньше ограничений на решение, характеризуясь безусловной устойчивостью в большинстве случаев.

Основной операцией в решении неявных разностных схем является решение СЛАУ с векторами неизвестных, размер которых соответствует размеру сеточной области, что влечет за собой столь же существенные требования к объему ОЗУ, как и для явных схем.

Цель данной статьи состоит в том, чтобы показать, что масштабные задачи с использованием неявных разностных схем могут быть решены на GPU без существенных накладных расходов на существующем оборудовании ценой некоторого ухудшения производительности. Для достижения этого мы продемонстрируем, каким образом метод Якоби может быть эффективно применен для решения СЛАУ для векторов неизвестных, размер которых превышает размер доступной видеопамяти.

Хотя метод Якоби и считается простым и критикуется за медленную сходимость в части случаев, но благодаря его простоте и хорошо изученной сходимости он по прежнему активно изучается [13], в частности в применении к вычислениям на GPU [14].

Метод

В качестве приложения для метода Якоби [15] воспользуемся разностной схемой для двумерного стационарного уравнения теплопроводности (диффузии) вида

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = \frac{-f(x,y)}{D},$$

где f(x,y) – функция источника тепла, D - эффициент температуропроводности, U(x, y) – распределение температур в пространстве и времени, заданное на квадратной области $x \in (0,X), y \in (0,X)$, с краевыми условиями вида $U(0,y) = U(X,y) = U(x,0) = U(x,X) = 0$.

Неявная разностная схема на равномерной сетке для этого уравнения имеет вид

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} = \frac{-h^2}{D} f_{i,j}.$$

Как видно, для нахождения $u_{i,j}$ необходимо решить СЛАУ вида $Ax = b$ где x представляет собой вектор, состоящий их построчной развертки элементов матрицы u, т.е. $x_k = u_{1,k}$ для $k = 1,...,N, x_k = u_{2,k-N}$ для $k = N+1,...,2N$, и т.д., вектор b - представляет собой такую же построчную развертку матрицы $\frac{-h^2}{D} f$, а матрица A имеет вид

$$A = \begin{pmatrix} T & -I & & & \\ -I & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & T \end{pmatrix}, \quad T = \begin{pmatrix} 4 & -1 & & & \\ -1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 4 \end{pmatrix}.$$

Очевидно, что диагональ матрицы A не содержит нулевых элементов. Тогда итерация метода Якоби для этого СЛАУ имеет следующий вид:

$$x_i^{k+1} = \frac{1}{a_{i,i}} \left(-\sum_{j \neq i} a_{i,j} x_j^k + b_i \right), \qquad (1)$$

где $i = 1,..,N, \quad k = 0,1,..,N, \quad x_i^k$ - значения приближения, полученные на k-ой итерации. x_i^0 представляет собой начальное приближение решения.

Принимая во внимание, что матрица A является пятидиагональной матрицей, можно заметить, что под знаком суммы в формуле (1) остается всего 4 слагаемых.

Выполнив обратный переход от векторного представления x_i к матричному представлению $u_{i,j}$ получим следующую формулу для итерации методом Якоби:

$$u_{i,j}^{k+1} = \frac{1}{4} \left(u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k + \frac{h^2}{D} f_{i,j} \right).$$

Аналогичным образом может быть получена схема для трехмерного уравнения теплопроводности $\frac{\partial^2 U}{\partial x^2}+\frac{\partial^2 U}{\partial y^2}+\frac{\partial^2 U}{\partial z^2}=\frac{-f(x,y,z)}{D}$. Разностная схема для него имеет вид:

$$u_{i,j}^{k+1}=\frac{1}{6}\left(u_{i+1,j,k}^{t}+u_{i-1,j,k}^{t}+u_{i,j+1,k}^{t}+u_{i,j-1,k}^{t}+u_{i,j,k-1}^{t}+u_{i,j,k+1}^{t}+\frac{h^2}{D}f_{i,j,k}\right). \quad (2)$$

Как видно, для произведения вычислений на один шаг необходимо держать в памяти матрицы u^t, u^{t+1} и f. Размер видеопамяти накладывает очевидное ограничение на размер решаемых задач. Однако, поскольку по форме формула 2 похожа на итерацию обычной явной разностной схемы для нестационарного уравнения теплопроводности, для неё применим метод пирамид, ранее используемых для явных разностных схем [8].

Этот метод подразумевает использование ОЗУ в качестве основного хранилища значений сеточной функции, в то время как вся сеточная область разбивается на пересекающиеся блоки, каждый из которых последовательно копируется в видеопамять из ОЗУ, на GPU выполняется обработка нескольких временных шагов разностной схемы, и затем результирующие данные копируются обратно в ОЗУ. Число шагов, выполняемых за раз на GPU называется высотой пирамиды n. Пересечение между соседними блоками имеет 2n узлов в ширину. Очевидно, что значения, попадающие в область пересечения будут вычислены дважды для двух блоков.

Высота пирамид является параметром декомпозиции. Слишком малые значения его приводят к избыточным пересылкам, слишком большие же ведут к чрезмерному дублированию вычислений. Представляется разумным выбрать высоту пирамид таким образом, чтобы общая длительность вычислений была минимальна.

Как было показано [8], для длительности работы алгоритма в случае одномерной декомпозиции сеточной области может быть применена следующая оценка:

$$\tau \approx K(I-1)^3\,\frac{R-n}{R-2n}\left(\frac{2}{n}\tau_c+\tau_a\right),$$

где τ - длительность вычислений, K - общее число временных шагов, на которые необходимо произвести вычисления, n - высота пирамиды, I - размер сетки по одному измерению (подразумевается, что сетка имеет одинаковый размер по всем измерениям), R - ширина обрабатываемого блока, τ_c - длительность пересылки одного значения из ОЗУ в видеопамять или обратно, τ_a – длительность вычисления одного значения по шаблону.

Учитывая, что для произведения одной итерации метода Якоби в отличии от используемой в [8] явной разностной схемы требуется скопировать в видеопамять не только значения самой сеточной функции, но и значения функции источников тепла f. Поэтому, оценка должна быть уточнена следующим образом:

$$\tau \approx K(I-1)^3\,\frac{R-n}{R-2n}\left(\frac{3}{n}\tau_c+\tau_a\right).$$

Таким образом для нахождения оптимальной высоты пирамид n необходимо решить задачу минимизации τ. Поскольку число возможных значений n не превышает R/2, то эта задача может быть решена достаточно быстро перебором ценой несоизмеримых с длительностью вычислений на GPU временных затрат.

Следует отметить существенное отличие метода Якоби от обычной явной разностной схемы: если при решении нестационарного уравнения делалось заданное число шагов по времени (итераций) K, то условие остановки итераций метода Якоби имеет вид $\left\|u^{k+1}-u^{k}\right\|<\varepsilon$. Т.е. для метода Якоби число итераций не может быть определено заранее. Однако опыт показывает, что оптимальная высота пирамид является сравнительно небольшой величиной (порядка сотен), в то время число итераций метода Якоби, необходимое для достижения разумной погрешности ε, на порядок выше. Таким образом авторами был использован алгоритм 1.

Алгоритм 1. Методы пирамид для схемы Якоби

Вход: u_0 – начальное распределение температур, массив размера $N \times N \times N$.

Вход: f – плотность источника тепла, массив размера $N \times N \times N$. Данные должны быть предварительно умножены на коэффициент теплопроводности D.

Вход: *optimalPyramidHeight* – оптимальная высота пирамид.

Вход: *maxPyramidHeight* – максимальная высота пирамид.

Вход: ε – максимальная погрешность.

Вход: N – размер сетки.

Вход: h – пространственный шаг сетки.

Вход: R – число слоёв размера $N \times N$, которые может обработать GPU.

Выход: u_n – результирующее распределение температур, массив размера $N \times N \times N$.

1: $u := u_0$

2: $n := min(maxPyramidHeight; optimalPyramidHeight)$

3: $blocks := buildPyramids(n; N)$

4: $repeat$

5: $u_n := zeroMatrix(N)$

6: $for i = 1 : size(blocks) do$

7: $u_{gpu} = copyRamToGpu(u(addGhostZone(blocks(i); n)))$

8: $f_{gpu} = copyRamToGpu(f(addGhostZone(blocks(i); n)))$

9: $for\ i = 1 : n\ do$

10: $tmp_{gpu} := JacobiKernel(u_{gpu}; f_{gpu})$

11: $swap(tmp_{gpu}; u_{gpu})$

12: $endfor$

13: $u_n(blocks(i)) := copyGpuToRam(u_{gpu})$

14: $endfor$

15: $swap(u; u_n)$

16: $until\ u_n - u_\infty < \varepsilon$

Предложенный алгоритм 1 использует три временных массива в видеопамяти ($u_{gpu}, tmp_{gpu}, f_{gpu}$ размером $R \times N \times N$. *JacobiKernel* – это ядро на GPU, которое выполняет итерацию Якоби над входными данными. Все прочие операции выполняются на центральность процессоре.

Основное ядро JacobiKernel представляет собой типичную шаблонную (оконную) обработку, алгоритм ядра приведен в алгоритме 2. Каждый поток GPU обрабатывает $R - 2$ значений. Это ядро обеспечивает высокую теоретическую загрузку GPU на аппаратном обеспечении, которое использовали авторы (см. таб. 1) и, согласно данным NVidia Visual Profiler, является ограниченным в первую очередь производительностью АЛУ, нежели производительностью памяти.

Результаты

Была разработана программа для описанного выше метода пирамид. Результаты вычислительного эксперимента представлены в таблице 2. Эксперимент проводился на компьютере, оснащенном GPU Nvidia Tesla C2050 (448 ядер CUDA, 3 ГБ видеопамяти), 96 ГБ ОЗУ и 2 CPU Xeon X5670 с 6 ядрами. Точность вычислений задавалась как $\varepsilon = 10^{-5}$. Распределение источников тепла f было задано в виде 8-октавного шума Перлина с диапазоном значений от -1 до 1. Начальное распределение u_0 заполнялось случайными числами, равномерно распределенными в диапазоне от -1 до 1. Все вычисления производились с одинарной точностью (FP32).

Как видно из таблицы 2, применение метода пирамид позволяет получить значительный выигрыш в производительности перед вычислениями на CPU несмотря на то, что размер видеопамяти не позволяет вмещать матрицы *u* и *f* целиком.

Таблица 2. Результаты вычислительных экспериментов

Разм. сетки *I*	Разм. данных, Гб	# итераций	Длительность вычислений, сек.	
			CPU, 12 потоков	Метод пирамид
800	5.7	8300	9687	1353
900	8.1	11200	20003	2671
1000	11.2	12300	33543	4218

Заключение

Таким образом, метод пирамид позволяет применять GPU к решению неявных сеточных уравнений для стационарных задач, размер сетки которых превышает доступный объем видеопамяти. При этом достигается существенно лучшая производительность, чем без использования GPU (ускорение по сравнению с многопоточной программой для CPU составляет 7-8).

Характерно, что примененный метод Якоби может быть заменен более изощренным методом Гаусса-Зейделя лишь путем небольшой модификации.

Решенная стационарная задача теплопроводности с математической точки зрения представляет собой весьма близкую задачу к одному временному шагу неявной разностной схемы для нестационарной задачи, тем самым расширяя границы применимости метода пирамид и на неявные разностные схемы для нестационарных задач, которые устойчивее рассмотренных ранее явных разностных схем. Это позволяет надеяться, что неявные разностные схемы могут быть эффективны на GPU даже при условии, что размер данных превосходит объем доступной видеопамяти.

Литература

1. D.L. Golovashkin, D.G. Vorotnikova, A.V. Kochurov and S.A. Malisheva, Solving finite-difference equations for diffractive optics problems using graphics processing units// Opt. Eng. 52 (9), 091719 (May 03, 2013); doi: 10.1117/1.OE.52.9.091719.

2. П.Ю.Якимов. Предварительная обработка цифровых изображений в системах локализации и распознавания дорожных знаков// Компьютерная оптика, 2013, 37(3), стр. 401-405.

3. OpenCurrent is an open source C++ library for solving Partial Differential Equations (PDEs) over regular grids using the CUDA platform from NVIDIA https://code.google.com/p/opencurrent/

4. V.S. Pavelyev, S.V. Karpeev, P.N. Dyachenko and Y.V. Miklyaev, Fabrication of three-dimensional photonics crystals by interference lithography with low light absorption, J. of Modern Opt(2009) . 9(56) 1133–1136.

5. Д.А.Савельев, С.Н.Хонина, Максимизация продольной электрической компоненты при дифракции на бинарном аксиконе линейно-поляризованного излучения//Компьютерная оптика, 2012, 36(4), стр. 511-517

6. P. Micikevicius, Multi-GPU Programming for Finite Difference Codes on Regular Grids, Stanford AHPCRC/iCME Colloquium Series, 2012 http://www.stanford.edu/dept/ICME/docs/seminars/Micikevicius-2012-01-23.pdf

7. G. Jin, T.Endo and S. Matsuoka, A parallel optimization method for stencil computation on the domain that is bigger than memory capacity of GPUs, Cluster Computing (CLUSTER), 2013 IEEE International Conference on (2013) 1–8.

8. Головашкин Д.Л., Кочуров А.В. Решение сеточных уравнений на графических вычислительных устройствах. Метод пирамид // Вычислительные технологии, Т. 17, N 3, 2012, с 39-52

9. Amazon EC2 Instance Types http://aws.amazon.com/ec2/instance-types/

10. Nimbix Accelerated Compute Cloud http://www.nimbix.net/cluster-builder/

11. Applied Math Institution of Russian Academy of Science Hybrid computations cluster K-100 http://www.kiam.ru/MVS/resourses/k100.html

12. P. Micikevicius, 3D Finite Difference Computation on GPUs using CUDA, In Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units (2009) p.79–84

13. I. Bethune, J. M. Bull, N. J. Dingle and N. J. Higham, Investigating the Performance of Asynchronous Jacobi's Method for Solving Systems of Linear Equations, University of Manchester EPrint. (2011)

14. H. Anzt, S. Tomov, J. Dongarra and V. Heuveline, A Block-Asynchronous Relaxation Method for Graphics Processing Units, Journal of Parallel and Distributed Computing. Vol 73, Issue 12 (2013) 16131626.
15. J.M. Ortega, Introduction to Parallel & Vector Solution of Linear Systems, Plenum Press New York, NY, USA, 1988.