Решение сеточных уравнений на графических вычислительных устройствах. Метод пирамид*

Д. Л. Головашкин[1], А. В. Кочуров[2]
[1] *Институт систем обработки изображений РАН, Самара, Россия*
[2] *Самарский государственный аэрокосмический университет, Россия*
e-mail: `dimitriy@smr.ru, ipris@inbox.ru`

Разработан метод решения сеточных уравнений явных разностных схем на графическом вычислительном устройстве с учётом ограничений на объём доступной видеопамяти. В основе предлагаемого подхода — адаптированный метод пирамид, ранее используемый для автоматического распараллеливания циклических фрагментов последовательных программ. Показано преимущество разработанного метода в производительности (на порядок) перед тривиальным. Теоретические оценки быстродействия подтверждены результатами вычислительных экспериментов.

Ключевые слова: метод пирамид, сеточные уравнения, параллельные вычисления, вычисления на графических устройствах.

Введение

Вопросам вычислений на графических процессорах (GPU) и, в частности, решению сеточных уравнений на видеокартах посвящено много работ (см., например, [1, 2]). Одним из основных ограничений методов, приведённых в этих работах, является требование на размещение сеточной области в графической памяти целиком. Вместе с тем на современных GPU не предусматривается установка дополнительной памяти. Кроме того, объёмы видеопамяти, установленной на видеокартах, намного меньше объёма ОЗУ. Так, GeForce 8800 GTX с 768 Мб видеопамяти позволяет применять метод FDTD к сеточным областям размером до $220 \times 220 \times 220$ узлов [2], а NVidia Tesla C2050 с 6 Гб памяти — до $550 \times 550 \times 550$ узлов.

На практике, например, при проектировании оптических элементов с линейными размерами в несколько микрон и нанометровыми неоднородностями требуемый размер сеточной области составляет тысячи узлов по одному направлению [3]. Это определяет потребность в методах решения сеточных уравнений, не требующих размещения всей сеточной области в видеопамяти.

Традиционный подход, состоящий в разбиении сеточной области на подобласти и в последовательной обработке каждой подобласти с копированием её из хранилища (ОЗУ или ПЗУ) перед обработкой и сохранением в хранилище после обработки на каждой итерации, является крайне неэффективным. Скорость передачи между ОЗУ

*Работа выполнена при финансовой поддержке РФФИ (гранты № 10-07-00553-а и 10-07-00453-а) и гранта Президента РФ МД-6809.2012.9.

и видеопамятью, а также между ОЗУ и ПЗУ сравнительно невысока по отношению к скорости чтения/записи из видеопамяти и к скорости чтения/записи из ОЗУ CPU.

В настоящей работе предлагается иной подход к решению этой задачи, основанный на применении метода пирамид [4], ранее используемого для автоматического распараллеливания циклических фрагментов последовательных программ [5]. Ключевой особенностью метода является сокращение числа пересылок за счёт дублирования вычислений.

1. Автоматическое распараллеливание циклических фрагментов последовательных программ методом пирамид

По Лэмпорту [6], значительная часть времени выполнения большинства программ тратится на один или несколько одно- или многомерных циклов, поэтому параллельное исполнение итераций таких циклов позволяет значительно ускорить вычисления.

Векторы, координаты которых представляют собой параметры итераций цикла, образуют пространство итераций. Два вектора будем называть зависимыми, если между соответствующими им итерациями существует зависимость (информационная или иная).

Применение метода пирамид включает следующие действия:

- на пространстве итераций A строится отношение частичного порядка как транзитивное замыкание ψ^+ введённого ранее отношения: $a_1 \psi a_2 \Leftrightarrow a_1$ зависит от a_2;
- из пространства итераций A выбирается множество результирующих векторов $R = \{a \in A |\ \nexists b \in A,\ b\psi a\}$ (от которых не зависят другие векторы);
- строится некоторое разбиение R на подмножества $R_i : R_1 \bigcup R_2 \bigcup \cdots \bigcup R_\mu = R$, $i \neq j \Leftrightarrow R_i \bigcap R_j = \emptyset$;
- каждое из этих подмножеств дополняется множеством итераций, от которых транзитивно зависят его результирующие векторы $A_i = R_i \bigcup \{a | a \in A,\ \exists r \in R,\ r\psi^+ a\}$;
- задаче i (в терминах модели канал/задача [7]) назначаются к исполнению векторы из множества A_i, упорядоченные в соответствии с транзитивным замыканием отношения непосредственного следования на множестве операторов последовательной программы.

Существенной особенностью метода пирамид применительно к сеточным уравнениям является возможность выбора распределения результирующих итераций по задачам, обеспечивающего локализацию данных, требуемых для выполнения каждой из задач. При этом любая из подобластей сеточной области, относящаяся к ведению одной задачи, может быть помещена в видеопамять целиком.

Предлагаемая модификация метода пирамид подразумевает отличный от традиционного подход: поскольку видеопамять каждого устройства ограничена и на одном устройстве невозможно одновременное исполнение нескольких задач, то задачи выполняются квазипараллельно, по очереди монопольно занимая устройство на время, необходимое для выполнения одного прохода.

2. Декомпозиция одномерной сеточной области

Применение метода пирамид к решению сеточных уравнений на GPU иллюстрируем на примере линейного одномерного однородного нестационарного уравнения теплопроводности $\frac{\partial}{\partial t}U = \alpha^2 \frac{\partial^2}{\partial x^2}U$, где $U(x,t)$ — распределение температур в пространстве и времени, $x \in [0, X]$, $t \in [0, T]$, α — коэффициент температуропроводности, с краевыми условиями первого рода и начальным условием $U(x, 0) = \phi(x)$. Простейшая явная разностная схема для этой задачи имеет вид [8]

$$U_i^{(k+1)} = \frac{\alpha^2 h_t}{h_x^2}\left(U_{i+1}^{(k)} + U_{i-1}^{(k)} - 2U_i^{(k)}\right) + U_i^{(k)}, \quad i \in \overline{0, I}, \quad k \in \overline{0, K-1}. \tag{1}$$

При разностном решении (1) традиционно требуется расположение сеточной области в видеопамяти целиком, что не всегда возможно. Чтобы обойти это ограничение, обычно сеточная область разбивается на подобласти, каждая из которых пересылается в видеопамять, вычисляются значения сеточных функций на следующем слое по времени во всех внутренних точках подобласти, после чего вычисленные значения пересылаются обратно. Тогда число пересылаемых в видеопамять значений сеточных функций определяется формой подобластей и в наиболее благоприятном случае декомпозиции на равные отрезки его можно оценить как $w_d = K(I-1)\frac{R}{R-2}$, где R — количество значений сеточных функций, которые могут быть размещены в видеопамяти. Число пересылаемых из видеопамяти значений составляет $w_h = K(I-1)$, а количество значений, вычисленных по дифференциальному шаблону, $w_a = K(I-1)$. Общее число пересылаемых между ОЗУ и видеопамятью значений составляет $w_c = 2K(I-1)\frac{R-1}{R-2}$, т. е. на одно вычисляемое значение приходится более двух пересылаемых. Такое решение порождает высокую нагрузку на шину между ЦП и видеокартой, что значительно увеличивает время работы программы.

Авторами предлагается следующая модификация метода пирамид: для вычисления K слоев сеточной области по времени необходимо последовательно выполнить s проходов методом пирамид, вычисляя каждый раз n слоев ($ns = K$). Взаимодействие между задачами осуществляется через ОЗУ и происходит только после завершения одного прохода и перед началом следующего, т. е. каждый проход задачи выполняют независимо от других. Высота пирамид n выбирается из соображений минимизации общей длительности работы программы.

Результирующим векторам для m-го прохода ($m \in \overline{1, s}$) соответствуют все итерации, вычисляющие $m \cdot n$ слой по времени и только они. При декомпозиции сеточной области на μ задач каждая из них вычисляет за один проход значения сеточных функций на r результирующих узлах сеточной области ($r\mu = I-1$). В силу формы дифференциального шаблона для вычисления r узлов на $m \cdot n$ слое требуются значения сеточных функций в $r+2$ узлах на $m \cdot n - 1$ слое, в $r+4$ узлах на $m \cdot n - 2$ слое и т. д. Таким образом, для работы программы необходима область видеопамяти для размещения $R = r + 2n$ значений сеточных функций. Условимся при этом хранить в видеопамяти значения только на одном слое по времени.

Рисунок 1 иллюстрирует данную декомпозицию на примере $\mu = 3,\ R = 9,\ n = 3$ для задачи 2 (здесь столбцы означают пространственные узлы сетки, строки — временные

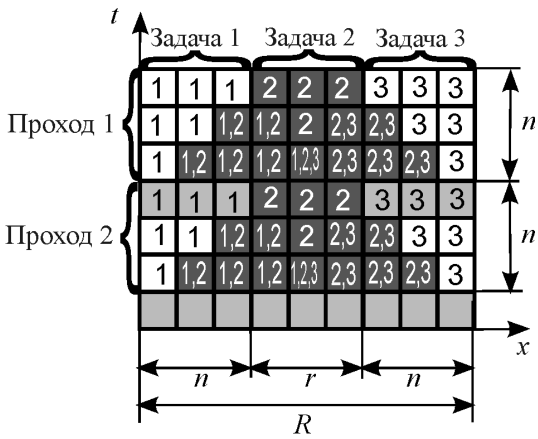

Рис. 1. Декомпозиция одномерной сеточной области для задачи 2 при $\mu = 3$, $R = 9$, $n = 3$

слои, квадраты — значения сеточных функций, цифры на квадратах — задачи, вычисляющие соответствующие значения). Все значения, вычисляемые задачей 2, выделены темно-серым, а значения, которые необходимо передать на устройство перед каждым проходом этой задачи, — светло-серым.

За каждый проход должны быть вычислены значения сеточных функций на результирующем слое и на всех промежуточных слоях. При этом каждой задаче на $(m-1)\cdot n+a$ слое ($a \in \overline{1,n}$) следует вычислить $R-2a$ значений. Всего одной задаче за один проход необходимо вычислить $\sum\limits_{a=1}^{n} R-2a$, т. е. $n(R-n-1)$, значений.

Ниже приведены оценки сверху числа значений сеточных функций, обрабатываемых за один проход одной задачей, полученные при использовании метода пирамид с декомпозицией одномерной сеточной области (количество операций будет несколько меньше для задач, обрабатывающих подобласти, прилегающие к краям сетки; то же относится к случаям одномерной и двумерной декомпозиции двумерной сеточной области — см. ниже):

Общее число пересылаемых значений ($w_c = w_d + w_h$)... $2(R-n)$, в том числе
  пересылаемых в видеопамять (w_d).................. R
  пересылаемых из видеопамяти (w_h) .... $R-2n$
Число значений, вычисляемых по шаблону (w_a)........ $n(R-n-1)$

Вычислим оценку длительности работы программы:

$$\tau = s \cdot \mu(w_c\tau_c + w_a\tau_a) = \frac{K}{n} \cdot \frac{I-1}{R-2n}\left(2(R-n)\tau_c + n(R-n-1)\tau_a\right).$$

Здесь τ_c и τ_a — средняя длительность пересылки значений сеточных функций в одном узле между ОЗУ и видеопамятью и средняя длительность вычисления значений сеточных функций в одном узле по шаблону соответственно. Эти величины, как правило, мало зависят от количества одновременно пересылаемых и обрабатываемых значений при достаточно больших объёмах данных, так как определяются в первую очередь пропускной способностью шины и производительностью GPU:

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

Для нахождения оптимальной высоты пирамиды n необходимо решить задачу оптимизации $\tau \to \min$.

Как видно, на общее время работы программы влияют два фактора: $f_d = \dfrac{R-n}{R-2n}$ и $f_t = \dfrac{2}{n}\tau_c + \tau_a$. Фактор f_d определяет отношение числа дублирующих операций, выполняемых при использовании предполагаемого подхода, к числу таковых, требуемых для вычисления по тривиальному алгоритму, и возрастает вплоть до $\dfrac{R}{2}$ при $n = \dfrac{R}{2}$. Фактор f_t показывает соотношение количества пересылок и вычислений, а также времени пересылки и вычисления одного значения без учёта дублирующихся вычислений. В тривиальном алгоритме на одно вычисленное значение всегда приходится два пересылаемых (одно — на устройство, другое — с устройства). Метод пирамид позволяет варьировать это соотношение: на все вычисленные за проход значения приходятся два пересылаемых, т. е. на одно вычисленное значение приходится в среднем $\dfrac{2}{n}$ пересылаемых.

3. Декомпозиция двумерной сеточной области

Рассмотрим дальнейшее приложение метода пирамид к решению сеточных уравнений на GPU, иллюстрируя его примером разностного решения двумерного однородного нестационарного уравнения теплопроводности

$$\frac{\partial}{\partial t}U = \alpha^2 \left(\frac{\partial^2}{\partial x^2}U + \frac{\partial^2}{\partial y^2}U \right),$$

где $U(x,y,t)$ — распределение температур в пространстве и времени, $x \in [0,X]$, $y \in [0,I]$, $t \in [0,T]$, α — коэффициент температуропроводности, с краевыми условиями первого рода и начальным условием $U(x,y,0) = \phi(x,y)$. Простейшая явная разностная схема в данном случае имеет вид [8]

$$U_{i,j}^{(k+1)} = \frac{\alpha^2 h_t}{h_x^2}\left(U_{i+1,j}^{(k)} + U_{i-1,j}^{(k)} + U_{i,j-1}^{(k)} + U_{i,j+1}^{(k)} - 4U_{i,j}^{(k)} \right) + U_{i,j}^{(k)}, \tag{3}$$

где $i \in \overline{0,I}, j \in \overline{0,I}, k \in \overline{0,K-1}$. Для этой задачи существует тривиальный алгоритм, аналогичный представленному в разделе 2, позволяющий работать с сеточными областями, превышающими объём доступной видеопамяти. При его использовании число пересылаемых в видеопамять значений сеточных функций можно оценить как $w_d = K(I-1)^2\dfrac{R}{R-2}$. Число пересылаемых из видеопамяти значений составит $w_h = K(I-1)^2$, а число значений сеточных функций, вычисленных по дифференциальному шаблону, $w_a = K(I-1)^2$. Общее количество пересылаемых между ОЗУ и видеопамятью значений составляет $w_c = 2K(I-1)^2\dfrac{R-1}{R-2}$.

Описанный выше метод пирамид может быть адаптирован и к двумерной сеточной области.

Ниже будут рассмотрены два варианта декомпозиции сеточной области по задачам.

3.1. Одномерная декомпозиция

При одномерной декомпозиции сеточной области на μ задач каждая из них вычисляет значения сеточных функций на r строках сеточной области на n слоев вперёд ($r\mu = I - 1$). Результирующим векторам для m-го прохода ($m \in \overline{1, s}$) соответствуют все итерации, вычисляющие $m{\cdot}n$ слой по времени и только они. Для работы программы необходима область видеопамяти для размещения $R = r + 2n$ строк сеточной области, или $R(I - 1)$ значений сеточных функций (на практике для выравнивания может потребоваться несколько больший объём видеопамяти). Условимся при этом хранить в видеопамяти значения только на одном слое по времени.

Рисунок 2 иллюстрирует одномерную декомпозицию на примере $\mu = 3, R = 9, n = 3$ для задачи 2. Здесь кубы — значения сеточных функций, столбцы — пространственные узлы сетки, строки — временные слои, цифры на кубах обозначают задачи, вычисляющие значения сеточных функций в направлении y. Все значения, вычисляемые задачей 2, выделены тёмно-серым, а значения, которые необходимо передать на устройство перед каждым проходом этой задачи, — светло-серым.

Ниже приведены оценки сверху числа значений сеточных функций, обрабатываемых за один проход одной задачей, полученные при использовании метода пирамид с одномерной декомпозицией двумерной сеточной области:

Общее число пересылаемых значений ($w_c = w_d + w_h$)... $2(R - n)(I - 1)$, в том числе
пересылаемых в видеопамять (w_d).................. $R(I - 1)$
пересылаемых из видеопамяти (w_h) .... $(R - 2n)(I - 1)$
Число значений, вычисляемых по шаблону (w_a)........ $n(R - n - 1)(I - 1)$

Как видно, от одномерного случая количество операций отличается только множителем $(I - 1)$. Поэтому оценка общей длительности вычислений может быть получена из (2) умножением на $(I - 1)$:

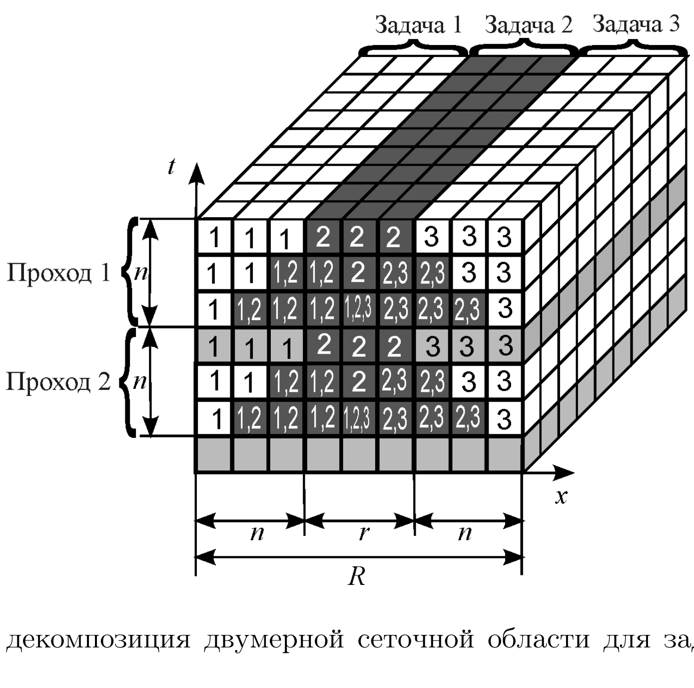

Рис. 2. Одномерная декомпозиция двумерной сеточной области для задачи 2 при $\mu = 3$, $R = 9$, $n = 3$

$$\tau \approx K(I-1)^2 \frac{R-n}{R-2n} \left(\frac{2}{n}\tau_c + \tau_a \right). \tag{4}$$

Для нахождения оптимальной высоты пирамиды n необходимо решить задачу оптимизации $\tau \to \min$.

3.2. Двумерная декомпозиция

При двумерной декомпозиции сеточной области на μ^2 задач каждая из них вычисляет значения сеточных функций на блоке $b \times b$ узлов на n слоев вперёд ($b\mu = I-1$). Результирующими итерациями для m-го прохода ($m \in \overline{1,s}$) будут все итерации, вычисляющие $m \cdot n$ слой по времени и только они. Для работы программы требуется область памяти для размещения квадратной подобласти сеточной области со стороной $B = b + 2n$, т. е. B^2 значений сеточных функций (на практике для выравнивания может потребоваться несколько больший объём видеопамяти). Условимся при этом хранить в видеопамяти значения только на одном слое по времени. Рисунок 3 иллюстрирует такую декомпозицию. Кубические ячейки означают значения сеточных функций в различных пространственных и временных узлах. Для одной из задач тёмно-серым выделены все вычисляемые ею значения, а светло-серым — все значения, которые необходимо передать в видеопамять перед каждым проходом этой задачи. Некоторые значения вычисляются многократно разными задачами.

Ниже приведены оценки сверху числа значений сеточных функций, обрабатываемых за один проход одной задачей, полученные при использовании метода пирамид с двумерной декомпозицией двумерной сеточной области:

Общее число пересылаемых значений ($w_c = w_d + w_h$)... $2((B-n)^2 + n^2)$, в том числе
  пересылаемых в видеопамять (w_d) .... B^2
  пересылаемых из видеопамяти (w_h) .... $(B-2n)^2$
Число значений, вычисляемых по шаблону (w_a) .... $n\left((B-n)^2 - 2(B-n) + \frac{n^2+2}{3}\right)$

По аналогии со случаем одномерной декомпозиции оценка длительности работы программы в зависимости от высоты пирамиды n имеет вид

$$\tau = s \cdot \mu^2 (w_c \tau_c + w_a \tau_a),$$

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

$$\tau \approx \frac{K(I-1)^2}{(B-2n)^2} \left(2((B-n)^2 + n^2)\tau_c \frac{1}{n} + \left((B-n)^2 + \frac{n^2}{3} \right) \tau_a \right). \tag{5}$$

Для нахождения оптимальной высоты пирамиды n необходимо решить задачу оптимизации $\tau \to \min$.

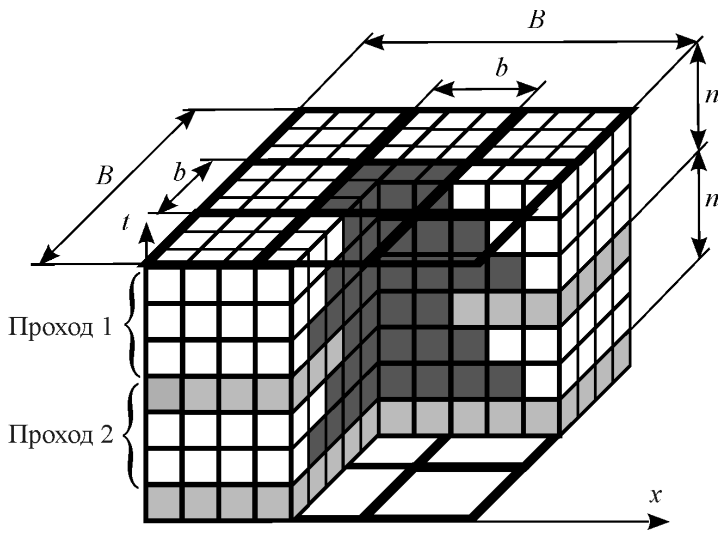

Рис. 3. Двумерная декомпозиция двумерной сеточной области

3.3. Сравнение способов декомпозиции

В таблице приведены теоретические оценки ускорения (отношения длительности вычислений по традиционному и авторскому алгоритмам), полученные с помощью формул (4) и (5) для одномерной и двумерной декомпозиций.

При двумерной декомпозиции не всегда можно обеспечить такую же скорость передачи, как при одномерной, поскольку во втором случае пересылка осуществляется наиболее оптимальным для аппаратного обеспечения образом — непрерывным блоком. В строчках таблицы "с замедленными пересылками" приведены теоретические оценки ускорения с учётом указанного эффекта (длительность пересылок увеличена на 20 %). Как видно, с ростом отношения τ_c/τ_a ускорение увеличивается. Применение метода

Теоретическая оценка ускорения для различных объёмов видеопамяти при $I = 32\,768$

Способ декомпозиции	τ_c/τ_a			
	1	5	10	15
128 Мб видеопамяти (1/32 от сеточной области)				
Одномерная	2.74	8.95	15.63	21.53
Двумерная	2.84	9.77	17.75	25.22
Двумерная с замедленными пересылками	2.77	9.22	16.36	22.84
512 Мб видеопамяти (1/8 от сеточной области)				
Одномерная	2.87	9.94	18.18	25.95
Двумерная	2.89	10.12	18.65	26.80
Двумерная с замедленными пересылками	2.83	9.70	17.60	24.97
1024 Мб видеопамяти (1/4 от сеточной области)				
Одномерная	2.93	10.46	19.56	28.41
Двумерная	2.92	10.36	19.31	27.97
Двумерная с замедленными пересылками	2.88	10.07	18.54	26.60

пирамид становится эффективнее при доминировании коммуникационных издержек. С увеличением объёма видеопамяти (уменьшением числа подобластей) ускорение возрастает и одномерная декомпозиция становится предпочтительнее двумерной.

На практике оптимальное быстродействие можно обеспечить, выбирая во время выполнения программы тот или иной способ декомпозиции в зависимости от параметров сеточной области и аппаратного обеспечения.

3.4. Программная реализация

В качестве среды для реализации была выбрана технология OpenCL [9]. Основное преимущество OpenCL перед другими подобными технологиями (NVidia Cuda, ATI FireStream) — отсутствие привязки к определённому аппаратному обеспечению. OpenCL позволяет запускать одну и ту же программу на различных устройствах от мобильных телефонов и CPU до вычислительных станций NVidia Tesla.

Отдельные подпрограммы, выполняемые на GPU, называются ядрами (kernel). Все потоки одновременно выполняют одну и ту же подпрограмму. OpenCL оперирует понятием групп потоков — набора потоков исполнения, выполняемых на одном мультипроцессоре, имеющем одно управляющее и несколько арифметико-логических устройств.

Как известно [10], для эффективного использования шины между GPU и видеопамятью все потоки группы должны производить чтения из одного блока данных, выровненного по границе 128 или 256 бит (кроме того, для некоторых GPU требуется, чтобы потоки производили чтение слов из этого блока в порядке следования потоков).

Чтобы все чтения из исходного массива были выровнены, в случае одномерной декомпозиции достаточно выровнять первый элемент каждой строки. В случае двумерной декомпозиции при переходе от слоя к слою меняется также и первый обрабатываемый столбец. Для обеспечения выравнивания данных первая группа потоков на каждом слое должна обрабатывать $g - k$ столбцов, где g — число потоков в группах, k — остаток от деления порядкового номера вычисляемого слоя в текущем проходе на g. Если первый элемент каждой строки блока данных выровнен в памяти, то вторая и последующая группы потоков будут производить чтение по выровненному адресу, которое происходит значительно быстрее.

Разработанный для GPU алгоритм состоит из двух ядер: одно подготавливает данные, второе производит один проход одной задачи методом пирамид. Далее приведён код основного ядра на алгоритмическом языке для векторного процессора (из фундаментальной монографии Дж. Голуба, Ч. Ван Лоуна [11]), которым по сути является отдельный мультипроцессор GPU.

```
{ w - число столбцов в обрабатываемом блоке }
{ g - число потоков в группе }
{ f - число столбцов, обрабатываемых первой группой }
{ G - номер текущей группы потоков, начиная с 0 }
{ x - номер первого столбца, обрабатываемого группой потоков }
{ s - число столбцов, которые будет обрабатывать группа }

if G == 0 {проверяем номер группы - для первой группы число столбцов вычисляется
 иначе}
   x = 0
   s = f
```

```
else
   x = f + g (G - 1)
   s = min(w - x,  g)
endif

{ M - обрабатываемый блок данных, представляет собой матрицу, хранимую по строкам
 (в стиле C)}
{ перед исполнением ядра содержит значения сеточной функции на предыдущем слое по
 времени, после исполнения ядра - значения сеточной функции на следующем слое
 по времени }

{ копируем из глобальной памяти в локальную участки текущей и предыдущей строк,
 обрабатываемые группой потоков }
p = M(1, x:x+s-1)
c = M(2, x:x+s-1)

{ h - число строк в обрабатываемом блоке }
for y = 1:h {цикл по строкам, обрабатываемым группой}
   n = M(y+2, x:x+s-1) {участок следующей строки тоже копируем в локальную память}

   { L = M(1:h, x-1), R = M(1:h, x+s) - заполняется ядром подготовки данных }
   { A, B - коэффициенты шаблона разностной схемы }
   M (y+1, x:x+s-1) = B c + A ([L(y) c(1:s-1)] + [c(2:s) R(y)] + p + n)

   { копируем данные внутри локальной памяти, чтобы не производить
    повторное чтение из глобальной }
   p = c {при переходе к следующей строке текущая становится предыдущей}
   c = n {а следующая - текущей}
end {for}
```

Соответствующий этому коду код основного ядра на языке C для OpenCL приведён ниже.

```
// группа потоков (warp) обрабатывает последовательный блок столбцов, каждый
 поток - свой столбец
// A, B - коэффициенты шаблона разностной схемы
// M - обрабатываемый блок данных, представляет собой двумерный массив, хранимый
 по строкам, перед исполнением ядра (kernel) содержит значения сеточной функции
 на предыдущем слое по времени, после исполнения - значения сеточной функции
 на следующем слое по времени
// chunk - длина строки в массиве in
// g - число потоков в группе
// w - число столбцов в обрабатываемом блоке
// h - число строк в обрабатываемом блоке
// gid - номер текущей группы потоков (warp)
// row - участок локальной памяти, через который потоки обмениваются данными
 внутри одной группы потоков
// (заполняется на этапе подготовки)
// id - номер текущего потока в группе потоков (warp)
// w0 - число столбцов, обрабатываемых первой группой (для выравнивания)
#define AT(i, j) ((i) + (j) * chunk)
```

```
int x; // номер столбца текущего потока (каждый поток обрабатывает свой столбец)
bool first = !id;
bool last;
if (gid) {
   x = w0 + (gid - 1) * g + id;
   last = (id == g - 1) || (x == w-1);
} else {
   x = id;
   last = (id == w0 - 1) || (x == w-1);
}

// в первой группе работает только w0 потоков, в последней - столько, сколько
 необходимо для обработки всех столбцов
if ((gid || id < w0) && (x < w)) {
   float prev = M[AT(x + 1, 0)];
   float curr = M[AT(x + 1, 1)];
   // каждый поток проходит по всем строкам своего столбца.
   for (int y = 0; y < h; y++) {
      // внутри группы потоков происходит обмен значениями с соседними потоками
       через локальную память
      barrier(CLK_LOCAL_MEM_FENCE);
      row[id] = curr;
      barrier(CLK_LOCAL_MEM_FENCE);
      float next = M[AT(x + 1, y + 2)];
      // крайние потоки группы получают значения соседних столбцов, заранее
       сохранённые на этапе подготовки
      float left = first ? L[y] : row[id-1];
      float right = last ? R[y] : row[i+1];
      // вычисления по шаблону
      M[AT(x + 1, y + 1)] = curr * B + (left + right + prev + next) * A;
      prev = curr;
      curr = next;
   }
}
```

Общий цикл работы программы для CPU, осуществляющей вызовы этих ядер, выглядит для каждого блока следующим образом:

- скопировать значения сеточных функций из ОЗУ в видеопамять;
- выполнить ядро подготовки данных;
- инициализировать $y_0 = 0, x_0 = 0, w_0 = g$;
- при одномерной декомпозиции $w = I - 1, h = R$;
- при двумерной декомпозиции $w = B, h = B$;
- для каждого из n слоев прохода
  - — выполнить ядро обработки;
  - — если текущий блок не прилегает к верхней границе сеточной области, увеличить номер первой обрабатываемой строки блока y_0 на 1 и уменьшить счётчик строк h на 1;

— если текущий блок не прилегает к нижней границе сеточной области, уменьшить счётчик строк h на 1;

— если используется двумерная декомпозиция и текущий блок не прилегает к левой границе сеточной области, то увеличить номер первого обрабатываемого столбца блока x_0 на 1, уменьшить счётчик столбцов первой группы потоков w_0 на 1 и счётчик столбцов w на 1;

— при $w_0 = 0$ уменьшить число групп потоков на 1 и $w_0 = g$;

— если используется двумерная декомпозиция и текущий блок не прилегает к правой границе сеточной области, то уменьшить счётчик столбцов w на 1;

- скопировать из видеопамяти в ОЗУ полученные значения сеточных функций.

4. Вычислительный эксперимент

Тестирование предложенных программ производилось на ноутбуке, оснащенном процессором Intel Core i5 M430 (два ядра, тактовая частота 2.27 ГГц, до 2.8 Ггц в режиме Turbo Boost), 3 Гб ОЗУ DDR3 1066 и видеокартой GeForce GT330M (шесть мультипроцессоров, 48 ядер, 512 Мб видеопамяти GDDR3, шина PCIe x16 Gen1).

Параметры теоретических оценок длительности работы (время пересылки и время вычисления одного значения) были определены на основе данных измерений, проведенных с помощью NVIDIA Compute Visual Profiler.

При одномерной декомпозиции длительность пересылки одного значения τ_c составила 2.35 нс. При двумерной декомпозиции τ_c сильно варьируется в зависимости от версии OpenCL. В версии 1.0 отсутствовала возможность пересылки прямоугольных регионов больших буферов, и для преодоления данного ограничения осуществлялась пересылка блоков в несколько транзакций, что приводило к существенному снижению производительности. Средняя длительность пересылки значения τ_c для этого способа была более чем на порядок выше, чем при одномерной декомпозиции, и составляла 91 мс. В OpenCL версии 1.1 средняя длительность пересылки одного значения с применением новых функций пересылки регионов составила 2.5 нс, что весьма близко к длительности пересылки при одномерной декомпозиции.

Величина τ_a мало отличалась для различных способов декомпозиции и в среднем составляла 0.6 нс.

Была также разработана программа, осуществляющая вычисления по разностной схеме [8] для трёхмерного уравнения теплопроводности с применением тривиального подхода и метода пирамид с одномерной декомпозицией области. Величина τ_a в среднем составила 0.85 нс. Теоретическая оценка для данного случая может быть легко получена из (4) умножением на размер области.

Для исследования эффективности теоретической оценки использовались метрики $\varepsilon_{\max} = \max\left|\dfrac{t_n - \tau_n}{t_n}\right|$ и $\varepsilon = \dfrac{1}{N}\sqrt{\sum\limits_{n=1}^{N}\left(\dfrac{t_n - \tau_n}{t_n}\right)^2}$, где τ_n — оценка длительности выполнения программы для соответствующих способа декомпозиции и высоты пирамиды n, вычисляемая по формулам (4) и (5), t_n — реальное среднее время работы программы при этих способе декомпозиции и высоте пирамиды n, N — максимальная высота пирамид, для которой производились вычисления.

На рис. 4 приведены теоретический (серый) и экспериментальный (чёрный) графики длительности выполнения программы на тестовом наборе данных при различной высоте пирамиды с одномерной декомпозицией для двумерной и трёхмерной сеточных областей. На рис. 5 рассмотрен случай двумерной декомпозиции двумерной области с применением OpenCL 1.0 и OpenCL 1.1. Для простоты сравнения с сеточными областями других размеров на графиках длительность вычислений указана в пересчёте на один узел области: $\tilde{\tau} = \dfrac{\tau}{K(I+1)^2}$.

В вычислительных экспериментах использовались следующие параметры сеточной области и декомпозиций: размер двумерной сеточной области $I = 16\,384$, ширина полосы для одномерной декомпозиции $R = 1024$, сторона блока при двумерной декомпозиции $B = 4096$. Трёхмерная сеточная область задавалась $640 \times 640 \times 640$ узлами (общее число узлов близко к числу узлов двумерной сеточной области), размер обрабатываемой "полосы" составлял $640 \times 640 \times 64$ узлов.

Для одномерной декомпозиции максимальное расхождение между теоретической оценкой длительности работы и экспериментальными данными $\varepsilon_{\max} = 0.13$, среднее отклонение $\varepsilon = 0.04$, для двумерной декомпозиции — соответственно $\varepsilon_{\max} = 0.17$, $\varepsilon = 0.06$, что подтверждает практическую применимость оценок в обоих случаях.

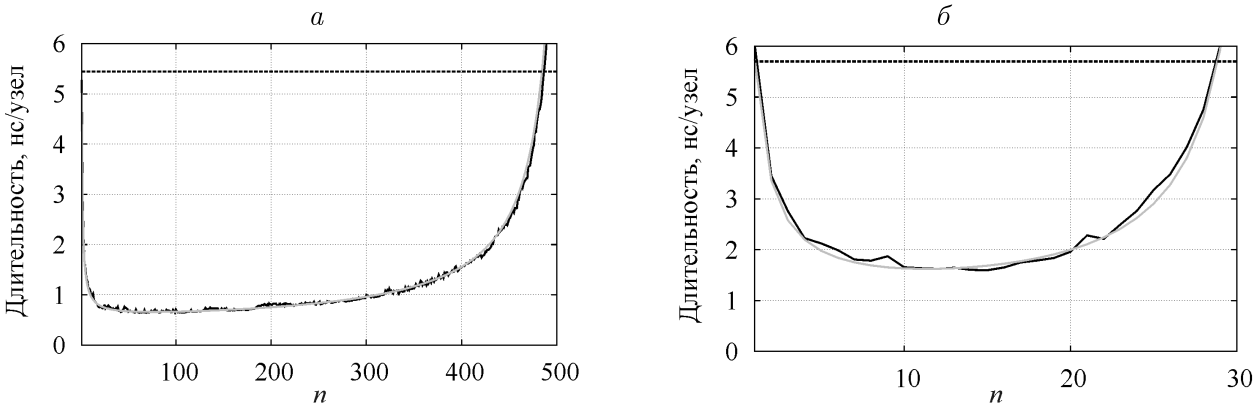

Рис. 4. Зависимость длительности вычислений по программе, реализующей одномерную декомпозицию, от высоты пирамиды n; *а* — двумерная, *б* — трехмерная сеточная область

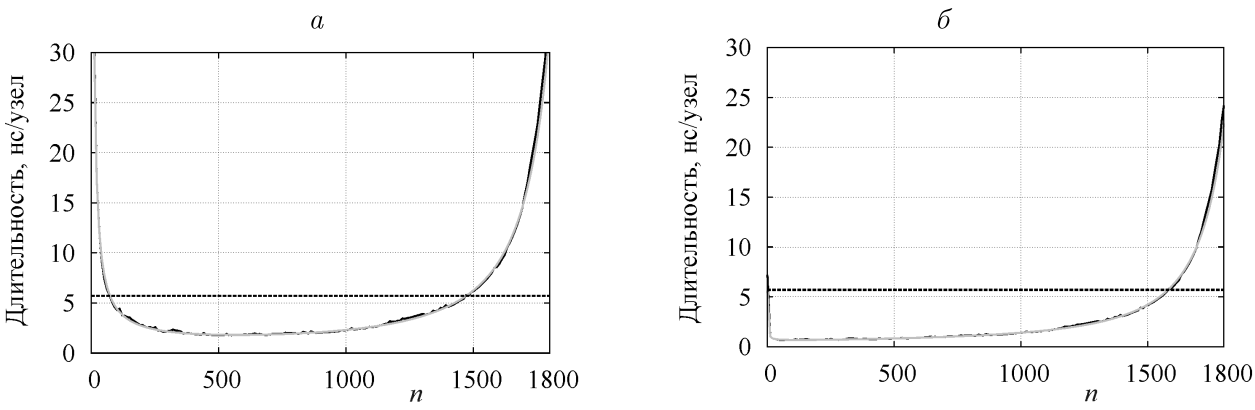

Рис. 5. Зависимость длительности вычислений по программе, реализующей двумерную декомпозицию двумерной сеточной области, от высоты пирамиды n; *а* — с применением OpenCL 1.0, *б* — с применением OpenCL 1.1

Средняя длительность работы по тривиальному алгоритму составила 5.44 нс/узел, по алгоритму метода пирамид с одномерной декомпозицией (при оптимальной высоте пирамид) — 0.64 нс/узел, с двумерной декомпозицией при использовании OpenCL 1.0 — 2.11 нс/узел, с применением OpenCL 1.1 — 0.71 нс/узел. Таким образом, максимальное достигнутое ускорение по сравнению с тривиальным алгоритмом составило 8.03 в случае одномерной декомпозиции и 7.66 в случае двумерной декомпозиции с применением OpenCL 1.1, т. е. для выбранных параметров сеточной области на использованном устройстве применение одномерной декомпозиции дало лучшие результаты.

Длительность вычислений тривиального алгоритма на трёхмерной сеточной области составила 5.74 нс/узел, программы по методу пирамид при оптимальной высоте пирамид — 1.62 нс/узел. Таким образом, ускорение составило 3.54.

Заключение

Разработанная модификация метода пирамид позволяет решать сеточные уравнения на областях, размер которых превышает объём доступной видеопамяти, достигая при этом значительно большей производительности, чем известные ранее методы. Лучший из предложенных алгоритмов в ходе вычислительных экспериментов продемонстрировал восьмикратное ускорение по отношению к тривиальному подходу.

Получены аналитические оценки длительности вычислений, позволяющие находить оптимальные параметры декомпозиции вычислительной области. Оценки подтверждены экспериментально с точностью 4–6 %.

В ходе вычислительных экспериментов на видеокарте GeForce GT330M для выбранных параметров сеточной области использование одномерной декомпозиции показало оптимальные результаты.

Развиваемый подход актуален при разностном решении других дифференциальных уравнений, например для явной схемы Yee метода FDTD.

Список литературы

[1] Евстигнеев Н.М. Интегрирование уравнения Пуассона с использованием графического процессора технологии CUDA // Вычисл. методы и программирование. 2009. Т. 10, № 2. С. 82–88.

[2] Adams S., Payne J., Boppana R. Finite difference time domain (FDTD simulations using graphics processors) // Proc. DoD High Performance Computing Modernization Program Users Group Conf. IEEE Conf. Publ., New York, 2007. P. 334–338.

[3] Pavelyev V.S., Karpeev S.V., Dyachenko P.N., Miklyaev Y.V. Fabrication of three-dimensional photonics crystals by interference lithography with low light absorption // J. of Modern Optics. 2009. Vol. 56, No. 9. P. 1133–1136.

[4] Головашкин Д.Л., Кочуров А.В. Электронная публикация тезисов работы “Решение сеточных уравнений на графических вычислительных устройствах. Метод пирамид” для конференции “Современные проблемы прикладной математики и механики: Теория, эксперимент и практика”. Новосибирск, 2011 // http://conf.nsc.ru/files/conferences/niknik-90/fulltext/37858/46076/kochurov_final.pdf

[5] Вальковский В.А. Параллельное выполнение циклов. Метод пирамид // Кибернетика. 1983. № 5. С. 51–55.

[6] Lamport L. The coordinate method for the parallel execution of DO-loops // Sagamore Computer Conf. on Parallel Proc. IEEE, New York, 1973. P. 1–12.

[7] Foster I. Designing and Building Parallel Programs. Addison-Wesley, 1995.

[8] Самарский А.А. Теория разностных схем. М.: Наука, 1977. 656 с.

[9] OpenCL — The open standard for parallel programming of heterogeneous systems (http://www.khronos.org/opencl/)

[10] Sanders J., Kandrot E. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley Professional, 2010.

[11] Голуб Дж., Ван Лоун Ч. Матричные вычисления. М.: Мир, 1999. 548 с.

Поступила в редакцию 7 июля 2011 г.,
с доработки — 2 апреля 2012 г.