ЧИСЛЕННЫЕ МЕТОДЫ И АНАЛИЗ ДАННЫХ

МОДЕЛИРОВАНИЕ ВЫЧИСЛЕНИЙ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ ПО РАЗНОСТНЫМ СХЕМАМ

Д.Г. Воротникова, А.В. Кочуров, Д.Л. Головашкин
Институт систем обработки изображений РАН, Самара, Россия,
Самарский государственный аэрокосмический университет имени академика С.П. Королёва (национальный исследовательский университет) (СГАУ), Самара, Россия

Аннотация

Предложены модели вычислений на GPU, рекомендуемые к использованию при организации вычислений по явным и неявным разностным схемам на графических процессорах. В частности, модель для выбора оптимальной длины вектора в алгоритмах с «длинновекторным» представлением и для отыскания высоты пирамиды, когда используется соответствующий метод построения параллельных алгоритмов.

Ключевые слова: моделирование вычислений, GPU, разностные схемы, CUDA.

Цитирование: **Воротникова, Д.Г.** Моделирование вычислений на GPU по разностным схемам/ Д.Г. Воротникова, А.В. Кочуров, Д.Л. Головашкин // Компьютерная оптика. – 2015. – Т. 39, № 5. – С. 801-807. – DOI: 10.18287/0134-2452-2015-39-5-801-807.

Введение

Развитие инженерных [1] и научных [2] приложений в новых областях (физика ядра, космические полёты), где применение аналитических подходов ограничено, обусловило всплеск интереса к вычислительной математике в середине прошлого века, сопровождавшийся также бурным развитием компьютерной техники. Появился новый инструмент познания – математическое моделирование, характеризующийся однонаправленной последовательностью «модель, численный метод, программная реализация» [3]. Усложнение в семидесятых годах прошлого века процессорных архитектур (конвейеризация, векторизация и др.) привело часть исследователей [4, 5] к осознанию необходимости учёта архитектурных особенностей уже на стадии построения численного метода.

Однако главенствующая долгое время (до середины двухтысячных) тенденция повышения быстродействия процессоров за счёт увеличения тактовой частоты позволяла научному сообществу игнорировать это соображение. И сейчас зачастую говорят о «распараллеливании» того или иного численного метода [6], понимая под этим исключительно написание параллельной программы или в лучшем случае алгоритма. Такой подход авторы настоящей работы полагают устаревшим.

Действительно, актуальная в последнее десятилетие проблема «кремниевого тупика» [7] вернула исследователей к ситуации, когда повышение быстродействия достигается за счёт усложнения архитектуры вычислительных систем. При этом естественно представить учёт архитектурных особенностей в качестве второго (первый – задачи физики и техники) источника развития численных методов, как это уже было сделано в [4] и [5], но не получило широкого распространения. По мнению авторов предлагаемой работы, для этого полезна запись моделей алгоритмов, моделей вычислительных систем и построение отображения первых на вторые.

Моделирование вычислительных архитектур, алгоритмов и процессов освещалось ранее [8], широко представлено в современной литературе [9,10], но в массовой вычислительной практике встречается редко (в том числе в силу своей сложности) и не признаётся обязательной частью процесса математического моделирования. В предлагаемой статье на типичном примере разностного решения уравнения теплопроводности с использованием GPU демонстрируется насущная необходимость указанного моделирования и предлагается соответствующая модель, отличающаяся от традиционных [9,10] практичностью.

1. Выбор модели алгоритма

Определяясь с объектом моделирования, укажем на существование для GPU достаточно разработанных моделей архитектуры и программных комплексов, неразрывно с такой архитектурой связанных [11]; моделей, весьма удобных программистам при составлении кода по готовым алгоритмам и не вполне полезных специалистам в области вычислительной математики в процессе синтеза алгоритмов в силу перегруженности техническими деталями с одной стороны и отсутствия математического аппарата с другой. Авторы настоящей работы намерены восполнить указанный пробел, записав модель алгоритмов, которые для GPU имеют свою специфику.

Выделим актуальные проблемы, на решение которых такая модель будет ориентирована в случае реализации вычислений на графических процессорах по разностным схемам. За последние 15 лет накоплен достаточно обширный опыт использования GPU в вычислительной практике, чтобы уверенно выделить две основные сложности такого использования: одновременное полное задействование большого количества исполнительных устройств (CUDA ядер) [12] и помещение обширных вычислительных областей в ограниченную (по сравнению с оперативной памятью узла суперкомпьютера) видеопамять [13]. Следова-

тельно, модель алгоритмов должна коррелировать с типом DM-SIMD (Distributed Memory Single Instruction Multiple Data) процессорной архитектуры [14], сочетая в себе векторное или матричное представление данных (операции со скалярами нежелательны) и инструменты для коммуникации между разными (распределённой и общей) видами оперативной памяти.

В таксономии Яна Фостера [8] отсутствуют модели алгоритмов, одновременно удовлетворяющие обоим требованиям. Джеймс Ортега [4] детализирует лишь нотацию векторных алгоритмов, к тому же не вполне удобную, по мнению авторов настоящей работы, которые остановили выбор на представлении Джина Голуба из [5]. Последнее использовалось его автором для записи векторных, матричных и параллельных алгоритмов для распределённой и общей памяти. К сожалению, нотация Голуба не вошла тогда в широкий научный оборот, возможно по причине вытеснения языка Linda [15] (частично её использовавшего) библиотекой OpenMP [16]. Так, в более позднем издании [17] Голуб, записывая алгоритмы для общей памяти, переходит к обозначениям, свойственным OpenMP, однако мало пригодным для работы с GPU, при которой коммуникации между оперативной и видеопамятью (а также между различными типами видеопамяти) прописываются в явном виде. Для записи алгоритмов разностного решения дифференциальных уравнений ни Голуб, ни Ортега выбранную нотацию никогда не использовали (авторы предлагаемой статьи делают это впервые), хотя и опубликовали совместную монографию по указанной тематике [17].

Признавая сходство ранней нотации Голуба с синтаксисом языка MatLab (отмечаемое самим Голубом в [5]), предназначенного в настоящее время в том числе и для вычислений на GPU, укажем на различия в области применения этих инструментов. Как будет показано далее, использование выбранной модели в качестве программы хоть и возможно, но не эффективно по сравнению с технологией CUDA.

Проиллюстрируем теперь применение модели Голуба для решения обеих упомянутых проблем реализации вычислений на графических процессорах по разностным схемам.

2. Учёт зависимости длительности вычислений от длины вектора

Ранее авторами [12] для разностного решения двумерного нестационарного уравнения теплопроводности были представлены несколько алгоритмов с «длинными» векторами для GPU. Наиболее эффективный из них основан на разворачивании двумерной сеточной функции в вектор и предварительном вычислении по частям дифференциального шаблона. В предлагаемой здесь нотации он выглядит следующим образом:

1. get(U, V);
2. for timwstep=1:N_{time}
3. T(2:(N+1)(N+2))=V(2:(N+1)(N+2))+V(N+3:$(N+2)^2$-1);
4. *V(N+4:(N+1)(N+2)-1)= α(T(2:N(N+2)-1) + +T(N+5:N(N+2)-1))+V(N+4:(N+1)(N+2)-1);*
5. ZeroBorder(V);
6. endfor;
7. put(V, U)

В алгоритме пересылка значений сеточных функций между GPU и CPU происходит при помощи операторов get (из CPU в GPU в строке 1 передаются значения функции в массиве *U*, принимаются в *V*) и put (из GPU в CPU в строке 7 передаются значения функции в массиве *V*, принимаются в *U*). Общее число узлов сеточной области, включая граничные, выбиралось равным $n=(N+2)^2$, где N+2 — число узлов по одному пространственному измерению [12].

На каждом слое timwstep по времени (за перебор N_{time} слоёв отвечает циклическая конструкция – строки от 2-й до 6-й) в соответствии с дифференциальным шаблоном «крест» производятся [12]: вычисление вспомогательного вектора *T* (строка 3), расчёт значений сеточной функций из *V* (строка 4) и восстановление граничных значений (строка 5). Все операции векторные и различаются длиной используемых векторов. При указании элементов в круглых скобках слева от двоеточия указывается первый задействованный элемент вектора, справа – последний. Величина α – константа.

Предваряя моделирование вычислений по алгоритму, укажем на модель процессорной архитектуры, наиболее соответствующую GPU. Согласно [19] графические процессоры классифицируются как SIMT (Single Instruction, Multiple Threads), в которых один набор команд выполняется над множеством потоков, формирующих в каждый момент обработки значения (один поток — одно или несколько значений) результирующего вектора (или нескольких векторов).

Отображая модель алгоритма на модель архитектуры [20], оценим длительность вычислений по векторному алгоритму, принимая это за цель моделирования. Будем оперировать числом потоков $num_{streams}$, исполняемых в доступных CUDA-ядрах (один или несколько потоков на ядро). Любая векторная операция в алгоритме при моделировании должна быть разбита на насколько в случае, если размер векторов в ней (пусть m) превышает $num_{streams}$. Тогда длительность одной операции, например сложения двух векторов, можно оценить величиной $t_{add} \times \lceil m / num_{streams} \rceil$, где t_{add} – время производства скалярной операции сложения, $m/num_{streams}$ – количество таких сложений, выполненных последовательно, $\lceil .. \rceil$ – операция округления до ближайшего большего целого.

При использовании библиотеки CUBLAS [21] для работы с векторами (в [12] демонстрируется эффективность её применения) необходимо учитывать длительность инициализации ядер и потоков t_{kernal} сопровождающей каждый вызов функции (например, сложения векторов) библиотеки, не зависящей от длины вектора. Тогда, продолжая пример, оценим время сло-

жения двух векторов как $t_{kernal} + t_{add} \times \lceil m / num_{streams} \rceil$. Завершая построение отображения, учтём затраты на перемещение данных из CPU в GPU и обратно, приняв их вместе за $\tau_{CPU/GPU}$ и положив общую длительность одной изолированной векторной операции за t_{kernal}.

Возвращаясь к приведённому выше алгоритму, оценим общее время T его исполнения на GPU как

$$T = \tau_{CPU/GPU} + N_{time} \times \left(\begin{array}{l} 4 \times t_{kernal} + t_{add} \times \left\lceil \left((N+2)^2 - 1 \right) / num_{streams} \right\rceil + \\ + t_{add} \times \left\lceil \left(N(N+2) - 2 \right) / num_{streams} \right\rceil + \\ + t_0 + t_{saxpy} \times \left\lceil \left(N(N+2) - 2 \right) / num_{streams} \right\rceil \end{array} \right), \quad (1)$$

где первое слагаемое $\tau_{CPU/GPU}$ соответствует строкам 1 и 7 алгоритма, множитель N_{time} – количеству итераций циклической конструкции (строки 2 – 6), в которой 4 раза ($4 \times t_{kernal}$) производятся векторные операции: сложение двух векторов размера $(N+2)^2$-1 (строка 3), сложение векторов размера $N(N+2)$-2 (строка 4, правая часть, первое выражение в скобках), saxpy [4] над векторами того же размера (оставшиеся действия в строке 4) и восстановление граничных значений (строка 5) длительностью t_0. Значения величин $\tau_{CPU/GPU}$, t_{kernal}, t_{add}, t_{saxpy} и t_0 определяются экспериментальным путём запуском соответствующих функций библиотеки CUBLAS и могут отличаться для разных видеокарт.

Адекватность представленных моделей алгоритмов, архитектуры и их отображения проверим сравнением с экспериментальными данными из работы [12], относящимися к первому алгоритму с «длинными» векторами для уравнения теплопроводности (алгоритм 1, рис. 2, 3). Здесь на рис. 1 сопоставлены зависимости длительности вычислений по указанному алгоритму, полученные в ходе натурных измерений (экспериментально) и методом математического моделирования (теоретически) по (1).

Рис. 1. Сравнение модельных и экспериментальных данных

Величина n варьировалась от небольших значений ($n = 100$), соответствующих малой сеточной области 10×10 узлов, до величины ($n = 49 \times 10^4$), определённой размером видеопамяти (2 Гб) карты NVIDIA GeForce GTX 960Ti.

Для оценки отклонения теоретических значений от экспериментальных введём понятие нормированного СКО как $\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\frac{T_i^{эксп} - T_i^{мод}}{T_i^{эксп}}\right)^2} \cdot 100\%$, где m – число проведённых экспериментов с $n = N \times N$, меняющейся N от 10 до 7000 с шагом 2 для $10 \le N \le 100$ и с шагом 20 для $N > 100$. В рассматриваемом случае (рис. 1) при $m = 450$ данная величина будет равна 5 %, что свидетельствует о хорошем соответствии выбранных моделей объектам исследования (алгоритму и архитектуре GPU).

Наблюдая зависимости длительности вычислений от размера векторных величин, отметим «ступенчатость» обоих графиков, наиболее ярко выраженную для результатов моделирования. В нашем случае первая ступенька на графике приходится на значение в районе n = 3600, что хорошо согласуется с архитектурой видеокарты NVIDIA GeForce GTX 980, на которой возможно запускать до 4000 CUDA тредов (для 2048 CUDA-ядер). Далее, с увеличением n, длительность вычислений опять медленно растёт до второй полной загрузки CUDA-ядер.

Объяснение более плавного вида ступенек для экспериментальных данных на рис. 1 авторы связывают, во-первых, с особенностями производства векторных операций на конвейере видеопроцессора (длительность загрузки/выгрузки конвейера в модели не учитывается), про который, в силу закрытости архитектуры CUDA [23], известно не более того, что он существует [19]. Во-вторых, отсутствует инструментарий для контроля загруженности CUDA-ядер, что не позволяет точно оценить аппаратные ресурсы, задействованные при выполнении программы и, как следствие, сделать сравнение теоретических и экспериментальных данных более точным. Таким образом, обсуждая применимость разработанного аппарата, авторы рекомендуют с осторожностью использовать (1) при моделировании для n кратных $num_{streams}$.

Предложенное описание алгоритмов во многом сходно с синтаксисом языка MatLab, то есть разработанная модель конструктивна. Однако реализовав на MatLab R2012b при помощи пакета Matlab Parallel Computing Toolbox обсуждаемый алгоритм, авторы наблюдали десятикратный рост длительности вычислений по сравнению с реализацией на CUDA, связанный с несовершенством упомянутого пакета, которое отмечалось и ранее [22]. Выбранная в первом параграфе настоящей работы модель предназначена для разработчиков векторных алгоритмов и позволяет математику до программной реализации оценить эффективность вычислений по ним. Создание программного комплекса традиционно производится программистом по готовому алгоритму в соответствии с моделью программирования в CUDA [23] (или другой) и выходит за рамки данной публикации.

3. Расчёт длительности вычислений при интенсивном обмене данными между CPU и GPU

Как отмечалось выше, кроме проблемы полной загрузки многочисленных ядер видеопроцессора, при ор-

ганизации вычислений по разностным схемам большое внимание принято уделять преодолению высоких коммуникационных издержек, обусловленных ограниченным объёмом видеопамяти. Традиционно для этого применяются различные варианты метода пирамид [24] («трапеций» в иностранной литературе [25]). Моделирование длительности вычислений по ним, предлагаемое здесь, позволяет заранее выбрать оптимальную высоту пирамиды и является в силу этого важной частью алгоритма. Стремясь проиллюстрировать применение представляемых моделей широким классом примеров (как для явных, так и для неявных разностных схем), авторы начинают изложение с алгоритма из [24], основанного на методе пирамид для решения сеточных уравнений неявной разностной схемы для трёхмерного стационарного уравнения теплопроводности с использованием метода Якоби. В нотации Голуба обсуждаемый алгоритм выглядит следующим образом:

```
1. (numBlocks, h)=buildPyramids();
2. repeat
3.   u=0;
4.   for i=1:numBlocks do
5.     get (U(0:N-1, 0:N-1, (i-1)×R+1-h:(i-1)×R+R+h), V);
6.     get(f( 0:N-1, 0:N-1, (i-1)×R+1-h:(i-1)×R+R+h), fgpu);
7.     for j=1:h do
8.       V(1:N-2,1:N-2,j:R-j+1)=(1/6)×(V(1:N-2,1:N-2,j-1:R-j)+
           +V(1:N-2,1:N-2,j+1:R-j+2)+V(0:N-3,1:N-2,j:R-j+1)+
           +V(2:N-1,1:N-2,j:R-j+1)+V(1:N-2,0:N-3,j:R-j+1)+
           +V(1:N-2,2:N-1,j:R-j+1)+α×fgpu(1:N-2,1:N-2,j:R-
           j+1));
9.     endfor
10.    u=U
11.          put(V( 0:N-1, 0:N-1, h:R-h), U(0:N-1, 0:N-1, (i-
1)×R+h:(i-1)×R+R-h));
12. endfor;
13. until ||u − U||∞ < ε .
```

Оперативная память ЭВМ здесь используется для хранения всех значений U сеточной функции, которые, однако, не помещаются целиком в видеопамять и разбиваются поэтому на блоки V (перекрывающиеся). Последние один за другим копируются на GPU (и обратно в строчках 5 и 11), где и выполняется их обработка.

В отличие от алгоритма из пункта 1 сеточные функции здесь трёхмерны, соответственно, трёхмерны и массивы U, V и f, f_{gpu}. Последние два хранят значения правой части решаемого в [24] уравнения в общей (f) и видеопамяти (f_{gpu}), копирование данных из первой во вторую производится в строчке 6 алгоритма.

Во внешнем цикле repeat until (строчки 2 – 13) производится h итераций по методу Якоби, после чего выполняется проверка условия остановки итерационного процесса (строка 13). При этом за U принимается текущее итерационное приближение, u – предыдущее приближение (строки 3, 10).

Второй цикл (строки 4 – 12) предназначен для перебора блоков (общим числом numBlocks, каждый шириной R (где R линейно зависит от высоты пирамиды и максимального объёма доступной видеопамяти) при одномерной декомпозиции и высотой h), на которые разбита сеточная область. Третий цикл (строки 7 – 9) связан с вычислениями h временных слоёв (высота пирамиды) по дифференциальному шаблону внутри блока (строка 8).

Строя отображение данного алгоритма на DM-SIMD архитектуру, примем за t_a длительность расчёта одного значения сеточной функции по дифференциальному шаблону, за $t_{CPU/GPU}$ – время пересылки в одном направлении массива из значений сеточных функций размером N^2.

В отличие от алгоритма из предыдущего пункта, здесь не производится переход к «длинным» векторам. Для векторов небольших размеров использование функций библиотеки CUBLAS неэффективно, в силу чего величина t_{kernal} более не учитывается. Хранение сеточной функции в многомерном массиве, вместо разворачивания её в один вектор, и непосредственная работа по дифференциальному шаблону делает удобным объединение длительности операций сложения и умножения в одну величину. Переход к трёхмерному случаю обуславливает новую нотацию для обозначения коммуникационных издержек. Как и ранее в пункте 2, все арифметические операции производятся над векторами, которые, однако, все одного размера и длина каждого $h(R\text{-}h\text{-}1)\times(N\text{-}2)^2$ не превышает значения $num_{streams}$.

Тогда время исполнения одной итерации внешнего цикла можно оценить как

$$T = numBlocks\times\left(n_c t_{CPU/GPU} / N^2 + n_a t_a / num_{treads}\right)\approx$$
$$\approx numBlocks\times \qquad (2)$$
$$\times\left(\left(3R-2h\right)t_{CPU/GPU} + \left\lceil\frac{h\left(R-h-1\right)\left(N-2\right)^2}{num_{treads}}t_a\right\rceil\right).$$

В табл. 1 приведены названия и значения используемых в (2) и ранее не упомянутых величин.

Табл. 1. Характеристики действий во втором цикле алгоритма

Величина	Значение
Количество отсылаемых (строки 5,6 алгоритма) в видеопамять значений сеточных функций $n_{CPU/GPU}$	$2\times R\times N^2$
Количество принимаемых (строка 11) из видеопамяти значений $n_{GPU/CPU}$	$(R\text{-}2h)\times N^2$
Общее количество пересылаемых элементов $n_c = n_{CPU/GPU} + n_{GPU/CPU}$	$(3R\text{-}2h)\times N^2$
Количество значений сеточных функций, вычисляемых по шаблону (в строке 8) n_a	$h(R\text{-}h\text{-}1)\times(N\text{-}2)^2$

Второй операнд, представленный в виде скобки с двумя слагаемыми, из (2) оценивает длительность одной итерации второго цикла, притом первое слагаемое этого выражения соответствует коммуникационным издержкам, второе – арифметическим.

Моделирование длительности вычислений используется в алгоритме для выбора высоты пирамиды. При вызове функции (первая строка алгоритма) buildPyramids() методом перебора из небольшого числа (10 – 15) значений решается соответствующая задача оптимизации. Слишком малые значения h приводят к большому количеству пересылок: например, при h = 1

коммуникации сопровождают переход в разностной схеме к каждому следующему слою по времени, при $h=2$ коммуникации производятся вдвое реже (хотя данных пересылается вдвое больше, но длительность коммуникаций уменьшается, как видно из рис. 2). Слишком большие значения h приводят к чрезмерному дублированию арифметических операций (на дублировании основан метод пирамид [24, 25]), что видно из (2) и рис. 2.

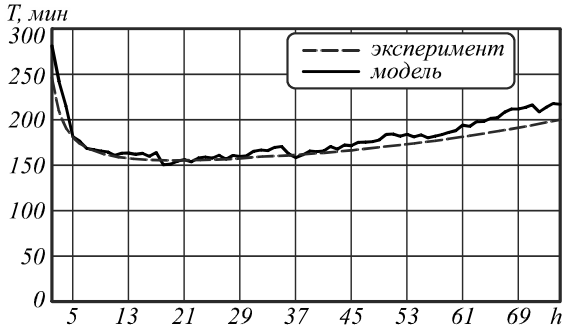

Рис. 2. Сравнение экспериментальных данных с моделью (N = 800)

Для выбранных параметров (рис. 2) расхождение экспериментальных данных с результатами моделирования, рассчитанное в соответствии с (2), не превысило 5 %. При дальнейшем росте высоты пирамиды ($h>80$) упомянутое расхождение увеличивается, в силу чего авторы рекомендуют использовать модель для оценки высоты пирамиды на интервале $2<h<N/10$, который во всех проведённых экспериментах (включая представленный на рис. 2, где оптимальная высота пирамиды $h=20$) достаточен для решения поставленной задачи оптимизации.

Заключение

Представленные в работе модели алгоритмов, вычислительных систем и их отображения авторы рекомендуют использовать при организации вычислений по явным и неявным разностным схемам на графических процессорах. В частности, для выбора оптимальной длины вектора в алгоритмах с «длинновекторным» представлением (что позволяет обеспечить рациональную загрузку CUDA-ядер) и отыскания высоты пирамиды, когда используется соответствующий метод построения параллельных алгоритмов (с целью минимизации коммуникационных издержек).

Благодарности

Работа выполнена при поддержке Министерства образования и науки РФ, а также грантов РФФИ (№14-07-31178мол_а, №14-01-31305мол_а и №14-07-00291А).

Литература

1. **Крылов, А.Н.** Лекции о приближенных вычислениях / А.Н. Крылов. – М.: Государственное издательство технико-теоретической литературы, 1954. – 401 с.
2. **Самарский, А.А.** О работах по теории разностных схем / А.А. Самарский // Международный конгресс математиков в Ницце, 1970: доклады советских математиков. – М.: Наука, Главная редакция физико-математической литературы, 1972. – С. 276-289.
3. Математическое моделирование: Идеи. Методы. Примеры / А.А. Самарский, А.П. Михайлов. – 2-е изд., испр. – М.: Физматлит, 2001. – 320 с. – ISBN 5-9221-0120-X.
4. **Ортега, Дж.** Введение в параллельные и векторные методы решения линейных систем / Дж. Ортега; пер. с англ. – М.: Мир, 1991. – 368 с.
5. **Golub, G.H.** Matrix Computations / G.H. Golub, Ch.F. Van Loan. – Baltimore: Johns Hopkins University Press, 1989. – 747 p.
6. **Карпов, В.Е.** Введение в распараллеливание алгоритмов и программ / В.Е. Карпов // Компьютерные исследования и моделирование. – 2010. – Т. 2, № 3. – С. 231-272.
7. **Фролов, В.** Введение в технологию CUDA. Электронный журнал «Компьютерная графика и мультимедиа» [Электронный ресурс]. – 2012. – URL: http://cgm.computergraphics.ru/issues/issue16/cuda (дата обращения 11.09.2015).
8. **Foster, I.** Designing and Building Parallel Programs / I. Foster. – Boston: Addison-Wesley Longman Publishing, 1995. – 430 p.
9. **Воеводин, В.В.** Параллельные вычисления / В.В. Воеводин, Вл.В. Воеводин. – СПб.: БХВ-Петербург, 2002. – 602 с.
10. **Хорошевский, В.Г.** Архитектура вычислительных систем / В.Г. Хорошевский. – М.: Издательство МГТУ им. Н.Э. Баумана, 2008. – 520 с.
11. Основы работы с технологией CUDA / А.В. Боресков, А.А. Харламов. – М.: ДМК Пресс, 2010. – 232 с.
12. **Воротникова, Д.Г.** Алгоритмы с «длинными» векторами решения сеточных уравнений явных разностных схем / Д.Г. Воротникова, Д.Л. Головашкин // Компьютерная оптика. – 2015. – Т. 39, № 1. – С. 87-93.
13. **Головашкин, Д.Л.** Решение сеточных уравнений на графических вычислительных устройствах. Метод пирамид / Д.Л. Головашкин, А.В. Кочуров // Вычислительные технологии. – 2012. – Т. 17, № 3. – С. 55-69.
14. Overview of Recent Supercomputers [Electronical Resource] / Aad J. van der Steen, Jack J. Dongarra. – 2008. – URL: http://www.netlib.org/utk/papers/advanced-computers/overview.html (request data 11.09.2015).
15. **Воеводин, Вл.В.** Курс лекций «Параллельная обработка данных». Система параллельного программирования Linda [Электронный ресурс]. – 2009. – URL: http://parallel.ru/vvv/lec7.html (дата обращения 11.09.2015).
16. Параллельное программирование с использованием технологии OpenMP: уч. пособие / А.С. Антонов. – М.: Издательство Московского университета, 2009. – 77 с.
17. **Golub, G.H.** Matrix Computations / G.H. Golub, Ch.F. Van Loan. – 3rd edition. – Baltimor: Johns Hopkins University Press, 1996. – 726 p.
18. **Golub, G.H.** Scientific Computing and Differential Equations: An Introduction to Numerical Methods / G.H. Golub, J.M. Ortega – California: Academic Press, 1992. – 344 p.
19. NVIDIA Tesla: A Unified Graphics and Computing Architecture / E. Lindholm, J. Nickolls, S. Oberman, J. Montrym // Micro, IEEE. – 2008. – Vol. 28, Issue 2. – P. 39-55. – ISSN: 0272-1732.
20. **Лельчук, Т.И.** Язык описания функциональной архитектуры вычислительных систем (Модель и общие принципы) / Т.И. Лельчук, А.Г. Марчук // Новосибирск: Препринт ВЦ СО АН СССР, JS 258. – 1981. – 19 с.
21. **Chrzeszczyk, A.** Matrix computation on the GPU. CUBLAS and MAGMA by examples / A. Chrzeszczyk, J. Chrzeszczyk [Electronical Resource]. – 2013. – URL: https://developer.nvidia.com/sites/default/files/akamai/cuda/files/Misc/mygpu.pdf (request data 16.09.2015).
22. **Golovashkin, D.L.** Solving finite-difference equations for diffractive optics problems using graphics processing units /

D.L. Golovashkin, D.G. Vorotnokova, A.V. Kochurov, S.A. Malysheva // Optical Engeneering. – 2013. – Vol. 52(9). – P. 091719. – DOI: 10.1117/1.OE.52.9.091719.

23. **Барилло, А.** NVIDIA CUDA – неграфические вычисления на графических процессорах / А. Барилло [Электронный ресурс]. – 2008. – URL: http://www.ixbt.com/video3/cuda-1.shtml (дата обращения 16.09.2015).
24. **Kochurov, A.** GPU implementation of Jacobi Method and Gauss-Seidel Method for Data Arrays that Exceed GPU-dedicated Memory Size / A. Kochurov, D. Golovashkin // Journal of Mathematical Modelling and Algorithms in Operations Research. – 2015. – DOI: 10.1007/s10852-015-9272-5.
25. **Jin, G.** A parallel optimization method for stencil computation on the domain that is bigger than memory capacity of GPUs. Cluster Computing (CLUSTER) / G. Jin, T. Endo, S. Matsuoka // 2013 IEEE International Conference. – 2013. – P. 1-8.

References

[1] Krilov AN. Lectures on approximate calculations [In Russian]. Moscow: State publishing house technical and theoretical literature; 1954.
[2] Samarskii AA. About the works in the theory of difference schemes [In Russian]. International Congress of Mathematicians (Nice); 1972.
[3] Samarskii AA, Mihajlov AP. Mathematical modeling: Ideas. Methods. Examples [In Russian]. Moscow: «Fizmatlit» Publisher; 2001.
[4] Ortega J. Introduction to Parallel and Vector Solution of Linear Systems. NY: Plenum Press; 1987.
[5] Golub GH, Van Loan ChF. Matrix Computations. JHU Press; 1989.
[6] Karpov VE. Introduction to the parallelization of algorithms and programs [In Russian]. Computer Research and Modeling 2010; 3: 231-72.
[7] Frolov V. Introduction to CUDA technology. Electronic magazine "Computer graphics and multimedia". Source: ⟨http://cgm.computergraphics.ru/issues/issue16/cuda⟩.
[8] Foster I. Designing and Building Parallel Programs. Addison Wesley; 1995.
[9] Voevodin VV, Voevodin VlV. Parallel computations [In Russian]. St.-Petersburg: "BHV-Petersburg" Publisher; 2002.
[10] Horoshevskii VG. Architecture of Computer Systems [In Russian]. Moscow: Publisher of Bauman Moscow State Technical University; 2008.
[11] Boreskov AV, Harlamov AA. Basics of CUDA technology [In Russian]. Moscow: "DMK Press" Publisher; 2010.
[12] Vorotnikova DG, Golovashkin DL. Long vector algorithms for solving grid equations of explicit difference scheme. Computer Optics 2015; 39(1): 87-93.
[13] Golovashkin DL, Kochurov AV. Solving finite-difference equations on GPU. The pyramid method [In Russian]. Computational Technologies 2012; 17(3): 55-69.
[14] van der Steen AJ, Dongarra JJ. Overview of Recent Supercomputers. Source ⟨http://www.netlib.org/utk/papers/advanced-computers/overview.html⟩.
[15] Voevodin VlV. Lectures "Parallel processing". Systems of parallel programming on Linda [In Russian]. Source: ⟨http://parallel.ru/vvv/lec7.html⟩.
[16] Antonov AS. Textbook: Parallel programming technology OpenMP [In Russian]. Moscow: MSU Press; 2009.
[17] Golub GH, Van Loan ChF. Matrix Computations. 3rd ed. Baltimor: JHU Press; 1996.
[18] Golub GH, Ortega JM. Scientific Computing and Differential Equations: An Introduction to Numerical Methods. California: Academic Press; 1992.
[19] Lindholm E, Nickolls J, Oberman S, Montrym J. NVIDIA Tesla: A Unified Graphics and Computing Architecture. Micro, IEEE 2008; 28(2): 39-55.
[20] Lelchuk TI, Marchuk AG. Description language of computing systems functional architecture (models and general principles) [In Russian]. Novosibirsk: Preprint of the Computing Center of the USSR Academy, JS 258; 1981.
[21] Chrzeszczyk A, Chrzeszczyk J. Matrix computation on the GPU. CUBLAS and MAGMA by examples. Source: ⟨https://developer.nvidia.com/sites/default/files/akamai/cuda/files/Misc/mygpu.pdf⟩.
[22] Golovashkin DL, Vorotnokova DG, Kochurov AV, Malysheva SA. Solving finite-difference equations for diffractive optics problems using graphics processing units. Optical Engeneering 2013; 52(9): 091719. DOI: 10.1117/1.OE.52.9.091719.
[23] Barillo A. NVIDIA CUDA – non-graphical computing on GPUs. Source: ⟨http://www.ixbt.com/video3/cuda-1.shtml⟩.
[24] Kochurov A, Golovashkin D. GPU implementation of Jacobi Method and Gauss-Seidel Method for Data Arrays that Exceed GPU-dedicated Memory Size. JMMA 2015; DOI: 10.1007/s10852-015-9272-5.
[25] Jin G, Endo T, Matsuoka T. 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 2013; 1-8.

MODELING OF GPU COMPUTING USING DIFFERENCE SCHEMES

D.G.Vorotnikova, A.V.Kochurov, D.L.Golovashkin
Image Processing Systems Institute,
Russian Academy of Sciences, Samara, Russia,
Samara State Aerospace University, Samara, Russia

Abstract

We propose models of GPU (graphics processing unit) computing that can be recommended for implementation of explicit and implicit difference schemes on GPUs. In particular, these are models for selecting an optimal vector length in the 'long-vector' algorithms and for finding an optimal pyramid height in corresponding parallel algorithms.

Keywords: computational modeling, GPU, difference scheme, CUDA.

Citation: Vorotnikova DG, Kochurov AV, Golovashkin DL. Modeling of GPU using difference schemes. Computer Optics 2015; 39(5): 801-807. DOI: 10.18287/0134-2452-2015-39-5-801-807.

Acknowledgements: The work was partially funded by the Russian Federation Ministry of Education and Science and RFBR grants numbers №14-07-31178mol_a, №14-01-31305mol_a и №14-07-00291A.

Сведения об авторах

Воротникова Дарья Геннадьевна, 1989 года рождения, в 2012 году окончила Самарский государственный аэрокосмический университет по направлению «Прикладная математика и информатика». Работает научным сотрудником в Институте систем обработки изображений Российской академии наук и является ассистентом кафедры наноинженерии Самарского государственного аэрокосмического университета. Область научных интересов: векторные и параллельные вычисления, FDTD- и BPM-методы.

E-mail: *daryavorotnikova@gmail.com* .

Daria Gennadievna Vorotnikova, (b. 1989). Graduated (2012) from S.P. Korolyov Samara State Aerospace University (SSAU). She works as a researcher at the Image Processing Systems Institute of the RAS and as an assistant of Nanoengineering department, SSAU. Scientific interests: BPM- and FDTD-method, vector and parallel algorithms for matrix computation.

Кочуров Александр Валерьевич, 1989 года рождения, в 2012 году окончил Самарский государственный аэрокосмический университет, а в 2015 году успешно окончил аспирантуру СГАУ по специальности «Прикладная математика, численные методы, комплексы программ». Работает стажёром-исследователем в Институте систем обработки изображений Российской академии наук. Область научных интересов: векторные и параллельные вычисления, технология CUDA, разностные схемы.

E-mail: *ipris@inbox.ru* .

Alexander Valerevich Kochurov, (b. 1989). Graduated (2012) from S.P. Korolyov Samara State Aerospace University (SSAU), in 2015 successfully completed his postgraduate course at SSAU in Applied Mathematics. He works as a trainee-researcher at the Image Processing Systems Institute of the RAS. Research interests: vector and parallel computing, CUDA technology, finite-difference schemes.

Головашкин Димитрий Львович, доктор физико-математических наук, доцент, ведущий научный сотрудник Института систем обработки изображений РАН. Область научных интересов: разностное решение уравнений Максвелла (FDTD-метод), дифракционная оптика, векторные и параллельные матричные вычисления.

E-mail: *dimitriy@smr.ru* .

Dimitry Lvovich Golovashkin, Doctor of Physics and Mathematics. He is an Associate Professor at SSAU and a Lead Researcher at the Image Processing Systems Institute of the Russian Academy of Sciences. Scientific interests: FDTD-method, sub wave optics, vector and parallel algorithms of matrix computation.

Поступила в редакцию 24 сентября 2015 г.
Окончательный вариант – 29 октября 2015 г.