УЧЕНЫЕ ЗАПИСКИ КАЗАНСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
Том 152, кн. 1 Физико-математические науки *2010*

УДК 519.684:532.546

РЕШЕНИЕ ЗАДАЧ ВЫЧИСЛИТЕЛЬНОЙ ГИДРОДИНАМИКИ С ПРИМЕНЕНИЕМ ТЕХНОЛОГИИ NVIDIA CUDA

Д.Е. Демидов, А.Г. Егоров, А.Н. Нуриев

Аннотация

На примере модельной задачи о развитии неустойчивости Кельвина – Гельмгольца рассмотрены преимущества технологии NVIDIA CUDA при решении задач гидродинамики численными методами. Рассмотрены как сеточные методы, так и бессеточный. Приводится краткое описание технологии CUDA.

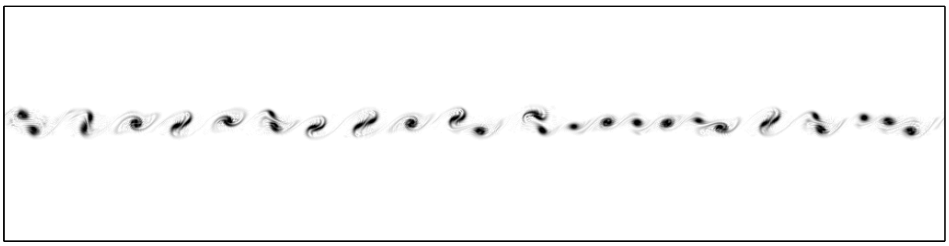
Ключевые слова: вычислительная гидродинамика, параллельные вычисления, технология CUDA.

Введение

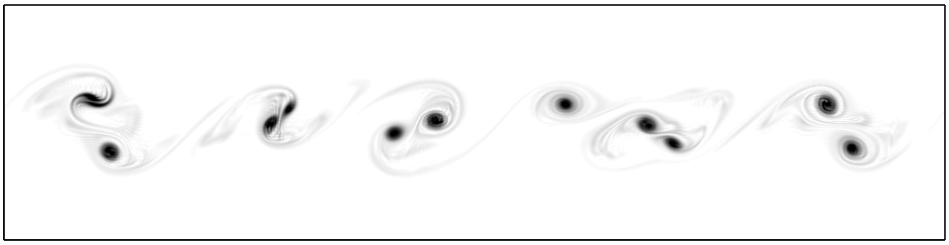
В настоящее время графические процессоры (GPU) являются оптимальной по соотношению цена – производительность параллельной архитектурой с общей памятью. Действительно, видеокарта NVIDIA GTX 285, доступная на рынке по цене около 15 000 руб., имеет пиковую производительность 950 GFLOPS, тогда как один из последних четырехядерных центральных процессоров (CPU) Intel Core2 Extreme QX9775 – всего 102 GFLOPS и продается по цене около 48 000 руб. Причина такой производительности видеокарт заключается в том, что они изначально были сконструированы для одновременного применения одной и той же шейдерной функции к большому числу пикселей, или, другими словами, для высокопроизводительных параллельных вычислений.

До недавнего времени использовать вычислительные мощности видеокарт было крайне неудобно, так как программисту приходилось выражать свою задачу в графических терминах. Ситуация изменилась в 2007 г., когда компания NVIDIA выпустила технологию CUDA – программно-аппаратную архитектуру для вычислений на графических процессорах. Несмотря на свою молодость, CUDA уже приобрела широкую популярность в научных кругах. Вычислительные задачи, реализованные на CUDA, получают ускорение в десятки и сотни раз в таких областях, как молекулярная динамика [1, 2], астрофизика [3, 4], медицинская диагностика [5, 6] и др.

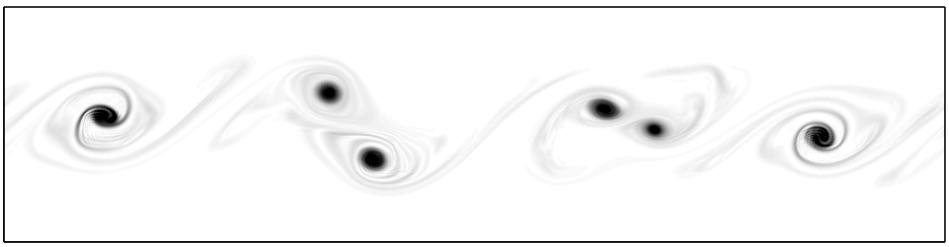
В настоящей работе рассматривается возможность применения технологии CUDA к задачам вычислительной гидродинамики. В качестве тестовой выбрана классическая двумерная задача о развитии неустойчивости в свободном сдвиговом слое между противоположно направленными потоками вязкой жидкости. Изначально бесконечно тонкий вихревой слой на границе потоков вследствие неустойчивости Кельвина – Гельмгольца распадается на систему вихревых сгустков (рис. 1). С течением времени ширина вихревого слоя растет за счет спаривания вихревых сгустков [7, 8]. Качественный статистический анализ данного процесса требует выполнения большого объема вычислительных экспериментов, что оказывается возможным с использованием технологии CUDA.



(a) $t = 1$



(b) $t = 3$



(c) $t = 5$

Рис. 1. Развитие неустойчивости Кельвина-Гельмгольца в свободном сдвиговом слое

Для решения соответствующей задачи вычислительной гидродинамики широко используются как сеточные, так и бессеточные методы. Наиболее популярные из них рассмотрены в настоящей работе. Это, с одной стороны, геометрический и алгебраический многосеточные методы, а также метод редукции с использованием быстрого преобразования Фурье при решении задачи на сетке. Бессеточные методы представлены простейшим вариантом метода точечных вихрей. Во всех случаях при решении задачи на GPU получено значительное ускорение (десятки и сотни раз) по сравнению с CPU.

1. Программная модель CUDA

CUDA – это среда разработки приложений, которая предоставляет в распоряжение программиста вычислительные мощности современных графических карт NVIDIA. В основе программной модели CUDA лежат три ключевые абстракции: иерархия групп потоков, разделяемая память и барьерная синхронизация, – реализуемые в виде минимального набора расширений языка C/C++.

Приложения CUDA используют как центральный, так и графический процессоры. Фактически видеокарта является высокопроизводительным сопроцессором. Параллельные участки кода выполняются на видеокарте в виде вычислительных ядер. *Ядро* – это функция, которая вызывается из основной программы на CPU

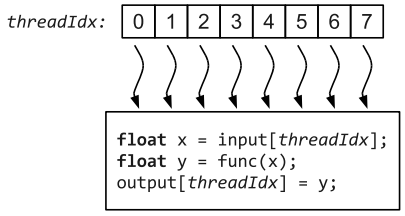


Рис. 2. Ядро выполняется массивом потоков, каждый из которых имеет уникальный идентификатор

и выполняется на видеокарте. Для выполнения каждого ядра запускается большое число одновременных потоков. Укажем два ключевых, на наш взгляд, различия между программными потоками на центральном процессоре и на видеокарте. Во-первых, десятки тысяч потоков CUDA могут быть созданы всего за несколько процессорных циклов. Благодаря этому расходы на управление потоками значительно ниже, чем на CPU, и даже легковесные ядра, состоящие всего из нескольких строчек кода, могут быть чрезвычайно эффективными. Во-вторых, переключение между контекстами потоков происходит мгновенно и используется для маскировки задержек при обращении к памяти. При этом если один из потоков простаивает в ожидании операции чтения/записи или синхронизации, он заменяется другим потоком, выполняющим арифметические операции. Это означает, что для эффективной загрузки видеокарты число потоков должно многократно превышать число процессоров.

Итак, каждое ядро CUDA выполняется массивом параллельных потоков. Все потоки выполняют один и тот же программный код, и для того чтобы они могли обрабатывать свой участок данных, каждому из них назначается уникальный идентификатор. Этот идентификатор используется для вычисления адресов памяти и организации ветвлений. Например, на рис. 2 представлено ядро, выполняемое восемью потоками. Сначала каждый из них использует свой идентификатор для считывания элемента из входного массива. Затем каждый поток передает считанное значение в функцию и записывает результат в выходной массив.

Это пример так называемой *неограниченно параллельной* (embarassingly parallel) задачи, когда каждый поток может выполнять свою подзадачу независимо от соседей. В реальных задачах требуется организация межпоточного взаимодействия: потоки должны, во-первых, совместно использовать результаты, чтобы избежать лишних вычислений; во-вторых, разделять операции доступа к памяти, чтобы снизить требования программы к пропускной способности канала данных. В программной модели CUDA взаимодействие допускается только между потоками внутри небольших групп, или *блоков*. Каждый блок – это одно-, двух- или трехмерный массив потоков фиксированного размера. При запуске ядра потоки организуются в одно- или двухмерную *решетку* (grid) блоков. Потоки внутри блока могут взаимодействовать посредством разделяемой памяти и выполнять барьерную синхронизацию. Организация потоков в блоки позволяет программам прозрачно масштабироваться при переходе на видеокарту с бóльшим числом потоковых процессоров.

Поясним последнее утверждение. Благодаря тому, что блоки потоков независимы, они могут выполняться в любом порядке на любом из доступных процессоров. Допустим, ядро состоит из 8 блоков потоков. На видеокарте с двумя мультипроцессорами каждый мультипроцессор должен будет выполнить по 4 блока. А на видеокарте с 4 мультипроцессорами каждый из них должен будет выполнить

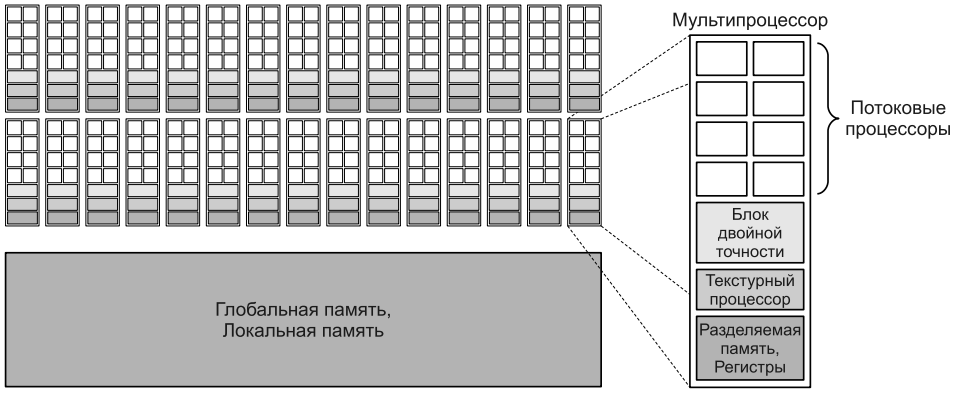


Рис. 3. Архитектура GPU 10-й серии

Табл. 1

Различные типы памяти, доступные программам на CUDA

Вид	Область видимости	Скорость доступа	Объем	Использование
Регистровая	Поток	Высокая	16384 регистра на МП	Локальные переменные потока
Локальная	Поток	Низкая	Ограничен объемом глобальной памяти	Локальные переменные, не уметившиеся в регистрах
Разделяемая	Блок	Высокая	16 КБ	Организация межпоточного взаимодействия
Глобальная	Программа	Низкая	До 4 ГБ	Хранение данных, обмен с CPU

по 2 блока, то есть задача будет выполнена в два раза быстрее. Такое масштабирование происходит автоматически без каких-либо изменений в программе. Поэтому однажды написанная программа на CUDA может выполняться на самых различных по производительности устройствах – от ноутбуков до высокопроизводительных серверов.

На рис. 3 представлена архитектура видеокарт NVIDIA 10-й серии (например, GTX 280). Видеокарта содержит 240 *потоковых процессоров*, каждый из которых может одновременно выполнять до 96 потоков. Потоковые процессоры сгруппированы в 30 *мультипроцессоров* (МП), каждый из которых содержит 8 потоковых процессоров, блок двойной точности, текстурный процессор и блок регистровой и разделяемой памяти.

Потоки CUDA имеют доступ к нескольким областям памяти, которые различаются областью видимости, скоростью доступа и объемом (табл. 1). *Регистровая память* используется для хранения локальных переменных потока. Переменные, которые не уместились в регистрах, хранятся в так называемой *локальной памяти*, которая, несмотря на название, находится вне чипа и потому имеет низкую скорость доступа. *Разделяемая память* используется для организации взаимодействия между потоками в блоке. Все потоки ядра имеют доступ чтения/записи к *глобальной памяти*, которая располагается в памяти видеокарты. Область видимости глобальной памяти – все приложение; ее содержимое не изменяется между запус-

ками различных ядер. Кроме того, центральный процессор также имеет доступ к глобальной памяти, поэтому глобальная память используется для обмена данными между CPU и GPU.

Рассмотрим модель исполнения кода в CUDA. Каждый поток выполняется потоковым процессором; блок потоков выполняется одним мультипроцессором; каждый мультипроцессор может выполнять несколько блоков. Число блоков, которое может выполняться на одном мультипроцессоре, ограничено разделяемыми ресурсами, то есть объемом регистровой и разделяемой памяти.

2. Сеточные методы

Перейдем к решению тестовой задачи о развитии неустойчивости Кельвина–Гельмгольца. При использовании подхода Эйлера течение вязкой жидкости в свободном сдвиговом слое описывается уравнениями Навье–Стокса. В терминах функции тока (ψ) – завихренности (ω) они имеют вид:

$$\Delta\psi = \omega, \tag{1a}$$

$$\frac{\partial\omega}{\partial t} + \mathbf{u}\cdot\nabla\omega - \varepsilon\Delta\omega = 0, \quad \mathbf{u} = \left(\frac{\partial\psi}{\partial y}, -\frac{\partial\psi}{\partial x}\right). \tag{1b}$$

Областью течения является полоса $-H < y < H$. На границах $y = \pm H$, моделирующих бесконечность, ставятся условия $\psi = \omega = 0$. В продольном направлении ψ и ω считаются периодическими функциями с периодом L. Начальное условие соответствует бесконечно тонкому сдвиговому слою:

$$t = 0: \quad \omega = 2(1 + \zeta(x))\delta(y).$$

Здесь δ – функция Дирака, ζ – малое случайное возмущение.

Дискретизация уравнения (1) проводится на равномерной прямоугольной сетке методом конечных разностей со вторым порядком точности по пространственной переменной. Для решения дискретной задачи используется полунеявная схема с поэтапным решением уравнений для завихренности и функции тока, предложенная в [9]. При этом уравнение переноса завихренности (1b) решается явно методом Рунге–Кутты четвертого порядка, так что для вычисления значения завихренности на новом временном слое требуется 4 шага. После каждого шага необходимо обращать оператор Лапласа для определения функции тока. Схему решения можно представить следующим образом:

$$\frac{\omega_1 - \omega^n}{\tau/2} + (\mathbf{u}^n\cdot\nabla_h)\omega^n = \varepsilon\Delta_h\omega^n, \quad \psi_1 = \Delta_{h,0}^{-1}\omega_1, \tag{2a}$$

$$\frac{\omega_2 - \omega^n}{\tau/2} + (\mathbf{u}_1\cdot\nabla_h)\omega_1 = \varepsilon\Delta_h\omega_1, \quad \psi_2 = \Delta_{h,0}^{-1}\omega_2, \tag{2b}$$

$$\frac{\omega_3 - \omega^n}{\tau} + (\mathbf{u}_2\cdot\nabla_h)\omega_2 = \varepsilon\Delta_h\omega_2, \quad \psi_3 = \Delta_{h,0}^{-1}\omega_3, \tag{2c}$$

$$k_4 = -\tau\left[(\mathbf{u}_3\cdot\nabla_h)\,\omega_3 - \varepsilon\Delta_h\omega_3\right], \tag{2d}$$

$$\omega^{n+1} = \frac{1}{3}\left(-\omega^n + \omega_1 + 2\omega_2 + \omega_3\right) + \frac{1}{6}k_4, \quad \psi^{n+1} = \Delta_{h,0}^{-1}\omega^{n+1}. \tag{2e}$$

Верхними индексами здесь обозначены значения переменных на соответствующих временных итерациях; нижними индексами обозначены промежуточные значения. Нижний индекс h обозначает оператор, дискретизированный на сетке с шагом h. Запись $\Delta_{h,0}^{-1}(\cdot)$ обозначает решение задачи Пуассона с нулевым граничным условием Дирихле для заданной правой части.

Табл. 2

Видеокарты, использовавшиеся в вычислительных экспериментах

Модель	Стоимость (руб.)	Число МП	Пиковая производительность (GFLOPS)	Пропускная способность памяти (ГБ/с)
8500 GT	1500	2	43	13
8600 GTS	3000	4	139	32
GTX 260	7000	27	804	112

Использование метода четвертого порядка позволяет обеспечить устойчивость данной схемы при разумном выборе шага по времени $\tau \sim h$. Наиболее дорогостоящей операцией, очевидно, является решение уравнения Пуассона для функции тока. Здесь необходимы алгоритмы, обладающие высокой степенью параллельности, например, редукция с использованием быстрого преобразования Фурье [10] или более общий многосеточный метод [11] в его геометрическом либо алгебраическом варианте. Производительность этих методов оценивалась с использованием трех видеокарт NVIDIA – 8500 GT, 8600 GTS и GTX 260 (табл. 2).

2.1. Геометрический многосеточный метод. Многосеточные методы (ММ) считаются оптимальными для решения эллиптических дифференциальных уравнений [11] и основываются на использовании сеточной иерархии и операторов перехода от одной сетки к другой. Основная идея заключается в ускорении сходимости итерационного метода, лежащего в основе ММ, за счет коррекции решения, полученного на точной сетке, с помощью решения уравнения на грубой сетке.

Геометрический многосеточный метод (ГММ) основывается на предопределенной сеточной иерархии. При решении сеточного уравнения $Au = f$ на каждом уровне иерархии задается сетка Ω_l, оператор задачи A_l, оператор продолжения $P_l : u_{l+1} \to u_l$ и оператор сужения $R_l : u_l \to u_{l+1}$.

На каждом уровне l иерархии один шаг ГММ выполняет следующие действия:

- текущее решение сглаживается несколькими шагами релаксации;
- вычисляется невязка $r_l = f_l - A_l u_l$, которая затем сужается в правую часть более грубого уровня: $f_{l+1} = R_l r_l$;
- если уровень $l+1$ – последний в иерархии, уравнение $A_{l+1}u_{l+1} = f_{l+1}$ решается прямым методом; иначе на уровне $l+1$ рекурсивно выполняется один шаг многосеточного метода с нулевым начальным приближением $u_{l+1} = 0$;
- решение на уровне l корректируется: $u_l \to u_l + P_l u_{l+1}$;
- выполняются несколько шагов релаксации.

Наиболее дорогостоящей операцией здесь является релаксация. Она занимает до 60% от общего времени счета, поэтому реализация этой процедуры особенно важна. В настоящей работе используется красно-черная релаксация Гаусса–Зейделя, так как она естественным образом распараллеливается.

2.2. Алгебраический многосеточный метод. Алгебраический многосеточный метод (АММ) [11] является одним из наиболее эффективных методов решения разреженных *неструктурированных* систем линейных уравнений большой размерности. В отличие от геометрического многосеточного метода (ГММ),

```
1  /* Умножение разреженной матрицы A на вектор x. */
2  __global__ void spmv(int m , int k, float *Adat, int *Aidx, float *x, fl oat *y) {
3      /* Каждый поток определяет номер своей строки матрицы */
4      int row = blockIdx.x * blockDim.x + threadIdx.x;
5      if (row < m) {
6          int  j, col;
7          float val;
8          float buf = 0.0;
9          /* Цикл по ненулевым элементам строки */
10         for(j = 0; j < k; j++) {
11             col = Aidx[row + j * m];
12             val = Adat[row + j * m];
13             /* Проверка, что ненулевые элементы в строке не кончились * /
14             if (col < 0) break;
15             buf += val * x[col];
16         }
17         y[row] = buf;
18     }
19 }
```

Рис. 4. Умножение разреженной матрицы в формате ELL на вектор

АММ не требует постановки задачи на сетке, а работает непосредственно с разреженной системой линейных алгебраических уравнений $Au = f$.

Алгоритм АММ состоит из двух основных этапов, примерно равных по числу операций: этап настройки метода и этап решения. Этап настройки заключается в конструировании проблемно-зависимой иерархии и включает в себя выбор вложенных подмножеств $\Omega_{l+1} \subset \Omega_l$ исходных переменных Ω_0, построение операторов продолжения $P_l : \Omega_{l+1} \to \Omega_l$ и операторов на грубых сетках $A_{l+1} = P_l^T A_l P_l$. Эта часть алгоритма АММ основана только на алгебраической информации относительно матрицы A. Второй этап алгоритма заключается в решении уравнения $Au = f$ для заданной правой части. Если это уравнение нужно решить для нескольких правых частей, то этап настройки достаточно выполнить один раз. Алгоритм этапа решения АММ практически совпадает с алгоритмом ГММ.

Нами АММ применяется для решения задачи Пуассона в (2). Этап настройки в классическом варианте АММ практически невозможно распараллелить, поэтому он полностью выполняется на CPU. Однако это не сказывается на скорости решения задачи (1), так как этап настройки достаточно выполнить один раз.

Этап решения был полностью реализован на GPU. Основным алгоритмическим элементом здесь является умножение разреженной матрицы на вектор. К этой операции сводятся практически все компоненты метода. Операции умножения разреженной матрицы на вектор с использованием CUDA посвящено несколько статей, например [12, 13]. Более всего на производительность влияет выбранный формат хранения разреженной матрицы. Одним из наиболее эффективных форматов является ELL [12], обеспечивающий *когерентный доступ к памяти* (coalesced memory access) соседними потоками [14]. В этом формате разреженная матрица размерности $M \times N$ с максимальным числом K ненулевых элементов в строке хранится в плотном массиве `Adat` размера $M \times K$, где строки с числом ненулевых элементов, меньшим K, дополняются нулями. Номера столбцов ненулевых элементов хранятся в массиве `Aidx` размерности $M \times K$. Оба массива размещаются в памяти по столбцам. На рис. 4 приведен пример ядра, выполняющего умножение разреженной матрицы на вектор. Каждый поток в ядре выполняет умножение одной строки матрицы на вектор и заполняет соответствующий элемент выходного массива.

Табл. 3

Решение задачи с применением сеточных методов

Устройство	Время решения	Ускорение
Геометрический многосеточный метод		
CPU AMD Athlon64 3200+	9 120 с	–
GPU 8500 GT	1 824 с	5×
GPU 8600 GTS	608 с	15×
GPU GTX 260	190 с	48×
Алгебраический многосеточный метод		
CPU AMD Athlon64 3200+	14 009 с	–
GPU 8500 GT	2 838 с	5×
GPU 8600 GTS	1 044 с	14×
GPU GTX 260	311 с	45×
Метод редукции		
CPU AMD Athlon64 3200+	4 560 с	–
GPU 8500 GT	798 с	6×
GPU 8600 GTS	264 с	17×
GPU GTX 260	87 с	52×

Другими часто выполняемыми операциями являются вычисление скалярного произведения векторов, определение максимального элемента в массиве, суммирование элементов массива. Библиотека с открытым исходным кодом CUDPP [15] позволяет эффективно выполнить эти операции на видеокарте.

2.3. Метод редукции. Метод редукции с применением быстрого преобразования Фурье [10] хорошо известен и применяется для нахождения решения простейших сеточных эллиптических уравнений в прямоугольнике. Основными алгоритмическими компонентами метода являются быстрое преобразование Фурье и обращение трехдиагональных матриц. Нами использовалось быстрое преобразование Фурье из библиотеки CUFFT [16], входящей в среду разработки CUDA. Библиотека позволяет выполнить прямое и обратное построчное преобразование Фурье вещественной матрицы. При обращении системы трехдиагональных матриц каждый поток решал одну систему линейных уравнений вида

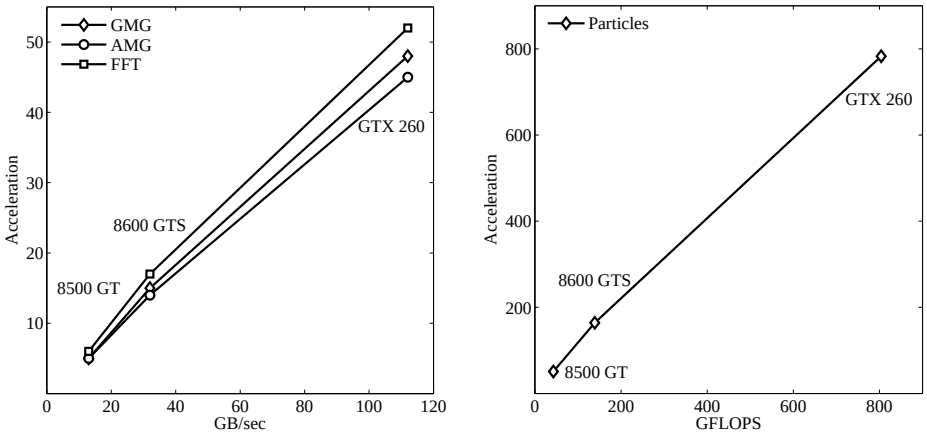
$$u_1 = u_n = 0, \quad u_{i-1} - (2 + 4\sin^2 k_i)u_i + u_{i+1} = h^2 f_i.$$

2.4. Оценка эффективности. В табл. 3 приведены результаты решения задачи (1) с применением схемы (2) описанными выше методами. Задача решалась на сетке 1024×1025 в области $(-\pi,\pi)\times(-\pi,\pi)$, было выполнено 4000 шагов по времени. Как видно, достигаемые ускорения для каждого из графических процессоров практически не зависят от выбранного метода. Причем даже наименее производительная видеокарта 8500 GT позволяет достичь по крайней мере пятикратного ускорения. Для данной задачи оптимальным оказался метод редукции. Однако он является узкоспециализированным методом, в то время как многосеточные методы применимы к гораздо более широкому классу задач.

Как видно из табл. 3, ускорение, получаемое сеточными методами, прямо пропорционально пропускной способности видеокарт. Это же подтверждает и рис. 5.

3. Метод частиц

Рассматриваемая задача может быть сформулирована и на основе подхода Лагранжа [17]. При этом поле завихренности представляется в виде конечного набора



(a) (b)

Рис. 5. Зависимость ускорения сеточных методов (a) и метода частиц (b) от пропускной способности и производительности видеокарт соответственно

точечных вихрей (частиц). Развитие поля завихренности во времени определяется движением этих частиц. В пренебрежении диффузией закон движения вихрей определяется системой обыкновенных дифференциальных уравнений

$$\frac{d}{dt}\mathbf{x}_i(t) = \sum_{j\neq i} \mathbf{K}\left(\mathbf{x}_i(t); \mathbf{x}_j(t)\right), \quad 0 \leq i < N, \tag{3a}$$

$$\mathbf{x}_i(0) = \frac{2\pi i}{N}\mathbf{e}_1, \tag{3b}$$

где $\mathbf{x}_i(t)$ – позиция i-й частицы в момент времени t, а функция $\mathbf{K}$ задает попарное гидродинамическое взаимодействие между двумя частицами. С учетом периодичности эта функция имеет следующий вид:

$$\mathbf{K}(\mathbf{x};\mathbf{y}) = \frac{1}{\cosh d_2 - \cos d_1}\begin{pmatrix}\sinh d_2 \\ -\sin d_1\end{pmatrix}, \quad \mathbf{d} = \mathbf{x} - \mathbf{y}.$$

Для учета диффузии в таком описании задачи может использоваться метод случайного блуждания (random walk) [18]. При этом на каждом временном шаге позиции частиц изменяются на случайную величину, распределенную по нормальному закону с дисперсией $\sigma^2 = 2\varepsilon\tau$, где ε – коэффициент диффузии, а τ – величина шага по времени. Для генерации последовательности псевдослучайных чисел был использован алгоритм Мерсенна Твистера (Mersenne Twister) [19]. Так как этот алгоритм носит чисто последовательный характер, каждый поток использовал собственную копию генератора.

Заметим, что задача (3) – это типичная задача о взаимодействии N тел. Такие задачи встречаются в астрономии, молекулярной динамике, гидродинамике и идеально подходят для реализации на GPU [20].

В табл. 4 приведены результаты решения задачи (3) методом частиц для числа вихрей $N = 1024$, $N = 4096$ и $N = 8192$. Как видно, карты 8500 GT и 8600 GTS сразу же выходят на максимальную производительность. Карта GTX 260 в первых двух экспериментах используется не полностью, с чем и связано относительно низкое ускорение. Так, в первом эксперименте при размере блока в 256 потоков

Табл. 4

Время решения задачи (3). В каждом из экспериментов использовался шаг по времени $\tau = 10^{-3}$

Устройство	Время решения	Ускорение
$N = 1024, \quad T_{\max} = 20$		
CPU AMD Athlon64 3200+	5 353 с	–
GPU 8500 GT	118 с	45×
GPU 8600 GTS	36 с	149×
GPU GTX 260	30 с	179×
$N = 4096, \quad T_{\max} = 5$		
CPU AMD Athlon64 3200+	21 156 с	–
GPU 8500 GT	408 с	51×
GPU 8600 GTS	129 с	164×
GPU GTX 260	37 с	565×
$N = 8192, \quad T_{\max} = 1$		
CPU AMD Athlon64 3200+	16 025 с	–
GPU 8500 GT	313 с	51×
GPU 8600 GTS	105 с	152×
GPU GTX 260	20 с	783×

задействованы всего 4 мультипроцессора из 27 доступных. Поэтому неудивительно, что в этом эксперименте ускорения для карт 8600 GTS (4 мультипроцессора) и GTX 260 практически совпадают.

На порядок бóльшие ускорения, достигаемые методом частиц по сравнению с сеточными методами, объясняются тем, что скорость вычислений здесь лимитируется производительностью карты, а не ее пропускной способностью (рис. 5).

4. Статистическая обработка результатов

Полный анализ полученных результатов выходит за рамки настоящей работы. Приведем лишь график зависимости ширины сдвигового слоя l от времени (рис. 6). В каждом вычислительном эксперименте величина l определялась как

$$l(t) = \sqrt{\frac{1}{2}\int \omega(x, y, t) y^2 dx dy}.$$

Как видно на рис. 6, отдельные вычислительные эксперименты демонстрируют нерегулярное поведение, особенно при турбулизации сдвигового слоя при $t \gtrsim 1$. Физический интерес представляет среднее по реализациям значение величины l. Для ее нахождения было проведено несколько сотен вычислительных экспериментов. Результат осреднения l по 750 экспериментам представлен на рис. 6 сплошной линией. В начальный период времени соответствующая зависимость хорошо описывается известной формулой ламинарного режима $l = \sqrt{\pi \varepsilon t}$ (пунктирная линия). При переходе к турбулентному режиму эта зависимость становится линейной. Заметим, что для проведения такого количества вычислительных экспериментов без применения технологии CUDA вместо двух дней потребовалось бы несколько месяцев.

5. Обсуждение

Как видно, любой из рассмотренных нами методов получает значительный прирост в производительности при реализации на GPU. Ускорение сеточных методов

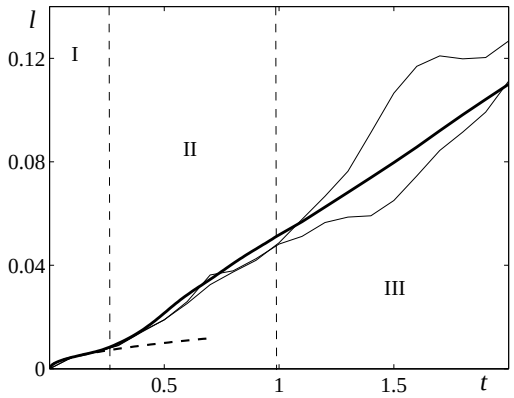


Рис. 6. Зависимость ширины l сдвигового слоя от времени t, осредненная по 750 реализациям (жирная сплошная линия). Тонкими линиями представлены зависимости $l(t)$ для двух отдельных экспериментов. Области I, II и III соответствуют ламинарному, переходному и турбулентному режимам

прямо пропорционально пропускной способности памяти видеокарты. Это объясняется тем, что отношение числа арифметических операций к числу обращений к глобальной памяти в этих методах невелико:

$$P = \frac{N_{op}}{N_{io}} \approx 1.$$

Таким образом, сеточные методы не используют полную вычислительную мощность видеокарты, так как процессоры в основном простаивают в ожидании очередной порции данных. В методе частиц отношение $P \gg 1$, благодаря чему ускорение при реализации на GPU прямо пропорционально пиковой производительности видеокарты.

Пожалуй, самым серьезным недостатком расчетов на GPU является то, что вычисления с одинарной и двойной точностью существенно различаются по производительности. Это объясняется тем, что в современных видеокартах на каждые восемь потоковых процессоров приходится один модуль двойной точности. Однако этот недостаток оказывается несущественным для сеточных методов, ограниченных пропускной способностью памяти. Опыт показывает, что при использовании двойной точности производительность сеточных методов снижается всего в 1.3–1.5 раз.

Работа выполнена при финансовой поддержке РФФИ (проект № 08-01-00548а).

Summary

D.E. Demidov, A.G. Egorov, A.N. Nuriev. Application of NVIDIA CUDA Technology for Numerical Solution of Hydrodinamic Problems.

Advantages of NVIDIA CUDA technology are presented on the example of classical problem of evolution of Calvin – Helmholz instability. Finite-difference as well as meshless methods are considered. Short observation of CUDA technology is given.

Key words: computational fluid dynamics, parallel computations, CUDA technology.

Литература

1. *Stone J.E.E., Phillips J.C.C., Freddolino P.L.L. et al.* Accelerating molecular modeling applications with graphics processors // J. Comput. Chem. – 2007. – V. 28, No 16. – P. 2618–2640.
2. *Van Meel J.A., Arnold A., Frenkel D. et al.* Harvesting graphics power for MD simulations // Mol. Simulat. – 2008. – V. 34, No 3. – P. 259–266.
3. *Zwart S.F.P. , Belleman R.G., Geldof P.M.*. High-performance direct gravitational N-body simulations on graphics processing units // New Astronomy. – 2007. – V. 12, No 8. – P. 641–650.
4. *Harris C., Haines K., Staveley-Smith L.* GPU accelerated radio astronomy signal convolution // Exp. Astron. – 2008. – V. 22, No 1. – P. 129–141.
5. *Muyan-Ozcelik P., Owens J.D., Xia J., Samant S.S.* Fast deformable registration on the GPU: A CUDA implementation of demons // Computational Science and its Applications: Intern. Conf. – 2008. – P. 223–233.
6. *Noël P.B., Walczak A., Hoffmann K.R. et al.* Clinical Evaluation of GPU-Based Cone Beam Computed Tomography // Proc. of High-Performance Medical Image Computing and Computer-Aided Intervention (HP-MICCAI). – 2008.
7. *Winant C.D., Browand F.K.* Vortex pairing: the mechanism of turbulent mixing-layer growth at moderate raynolds number // J. Fluid Mech. – 1974. – V. 63, No 2. – P. 237–255.
8. *Aref H., Siggia E.D.* Vortex dynamics of the two-dimensional turbulent shear layer // J. Fluid Mech. – 1980. – V. 100, No 4. – P. 705–737.
9. *Weinan E., Liu J.-G.* Vorticity boundary condition and related issues for finite difference schemes // J. Comput. Phys. – 1996. – V. 124. – P. 368–382.
10. *Hockney R.W.* A fast direct solution of Poisson's equation using Fourier analysis // J. ACM. – 1965. – V. 12, No 1. – P. 95–113.
11. *Trottenberg U., Oosterlee C., Schüller A.* Multigrid. – London: Acad. Press, 2001. – 631 p.
12. *Bell N.* Efficient sparse matrix-vector multiplication on CUDA: NVIDIA Technical Report NVR-2008-004 / N. Bell, M. Garland: NVIDIA Corporation, 2008.
13. *Baskaran M.M.* Optimizing sparse matrix-vector multiplication on GPUs: IBM Research Report RC24704 (W0812-047) / M. M. Baskaran, R. Bordawekar: IBM, 2009.
14. NVIDIA CUDA Programming guide. – NVIDIA Corporation, 2009. – Version 2.2.
15. *Harris M.* CUDA data parallel primitives library. – URL: http://gpgpu.org/developer/cudpp.
16. CUDA CUFFT Library. – NVIDIA Corporation, 2009. – Version 2.2.
17. *Хокни Р., Иствуд Д.* Численное моделирование методом частиц. – М.: Мир, 1987. – 639 с.
18. *Marsden J.E., Chorin A.J.* A Mathematical Introduction to Fluid Mechanics (Texts in Applied Mathematics). – Springer, 1993. – 169 p.
19. *Matsumoto M., Nishimura T.* Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator // ACM Trans. Model. Comput. Simul. – 1998. – V. 8, No 1. – P. 3–30.
20. *Nyland L., Harris M., Prins J.* Fast N-body simulation with CUDA // GPU Gems 3 / Ed. by H. Nguyen. – Addison Wesley Professional, 2007. – P. 677–695.

Поступила в редакцию
28.12.09

Демидов Денис Евгеньевич – кандидат физико-математических наук, старший научный сотрудник Казанского филиала Межведомственного суперкомпьютерного центра РАН.

E-mail: *ddemidov@ksu.ru*

Егоров Андрей Геннадьевич – доктор физико-математических наук, заведующий кафедрой аэрогидромеханики Казанского государственного университета.

E-mail: *Andrey.Egorov@ksu.ru*

Нуриев Артем Наилевич – бакалавр механики, магистрант кафедры аэрогидромеханики Казанского государственного университета.

E-mail: *artem501@list.ru*