УДК 533.6

РЕАЛИЗАЦИЯ МЕТОДА РЕШЕТОЧНЫХ УРАВНЕНИЙ БОЛЬЦМАНА ДЛЯ РАСЧЕТОВ НА GPU-КЛАСТЕРЕ

Д. А. Бикулов[1], Д. С. Сенин[1], Д. С. Демин[1], А. В. Дмитриев[1], Н. Е. Грачев[1]

Рассмотрена реализация метода решеточных уравнений Больцмана D3Q19 с использованием технологии NVIDIA CUDA и GPU-кластера. Проведена оценка масштабируемости на суперкомпьютерном комплексе “Ломоносов”, установленном в НИВЦ МГУ. Модель прошла верификацию на двух тестах: расчет стационарного течении Пуазейля и расчет коэффициента лобового сопротивления шара. Результаты моделирования хорошо согласуются с теоретическими результатами в обоих случаях.

Ключевые слова: решеточный метод Больцмана, CUDA, GPU, высокопроизводительные вычисления.

Введение. Видеокарты изначально создавались как устройства, способные обрабатывать большие объемы данных для рендеринга изображений. Почти сразу же стало понятно, что использовать такие мощности можно и для других задач, например, в вычислительной физике или биологии. Раньше это требовало знания специального языка шейдеров. Теперь, благодаря технологии NVIDIA CUDA, это сделать значительно проще.

Эффективное использование графических ускорителей возможно для особого класса высокопараллельных задач, т.е. задач, в которых расчет каждой части происходит независимо или требует минимум информации о других ее частях. Решеточный метод Больцмана как раз относится к такому типу: для одной ячейки нужна информация лишь о непосредственно примыкающих соседях независимо от общего их числа, достигающего 10^9 и более.

Решеточный метод Больцмана (LBM, Lattice Boltzmann Method) используется для моделирования множества процессов: развития стеноза артерий [4], течения неньютоновских жидкостей [5], течения в микроканалах [13], роста кристаллов в перенасыщенном растворе [10], течения жидкости в поровом пространстве [9], поведения двухфазных систем типа жидкость–пар [2] и др. Опубликовано множество работ (например [2, 11, 15, 16]), в том числе и с использованием MPI (Message Passing Interface) [21], в которых метод LBM реализован с использованием возможностей графических ускорителей (GPU, Graphics Processing Unit). В результате использования GPU наблюдается рост скорости вычислений на два порядка [23].

В настоящей статье рассмотрен решеточный метод Больцмана, особенности его реализации на CUDA (Compute Unified Device Architecture) и изменения, необходимые для использования нескольких видеокарт одновременно.

1. Описание метода LBM. Метод решеточных уравнений Больцмана [18] — сравнительно новый способ решения кинетического уравнения Больцмана

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial \mathbf{x}} \frac{\mathbf{p}}{m} + \frac{\partial f}{\partial \mathbf{p}} \mathbf{F} = \left(\frac{\partial f}{\partial t}\right)_{\text{coll}}, \tag{1}$$

где m — масса частиц; $\mathbf{F}$ — поле сил, действующих на частицы; $\left(\frac{\partial f}{\partial t}\right)_{\text{coll}}$ — интеграл столкновений, характеризующий скорость изменения функции распределения вследствие столкновений частиц; $\mathbf{x}$ и $\mathbf{p}$ — координата и импульс соответственно; t — время, f — функция распределения частиц. В настоящее время этот метод приобретает все бóльшую популярность на фоне развития технологий массивно-параллельных вычислений.

[1] ООО “Интровижн”, Ленинские горы, Научный парк МГУ, вл. 1, стр. 77, 119992, Москва; Д. А. Бикулов, программист, e-mail: bikulov@introvision.ru; Д. С. Сенин, технический директор, e-mail: senin@introvision.ru; Д. С. Демин, ст. аналитик, e-mail: demin@introvision.ru; А. В. Дмитриев, директор по развитию, e-mail: dmitriev@introvision.ru; Н. Е. Грачев, ген. директор, e-mail: grachev@introvision.ru

© Научно-исследовательский вычислительный центр МГУ им. М. В. Ломоносова

Кинетическое уравнение Больцмана (1) описывает эволюцию одночастичной функции распределения f во времени в фазовом пространстве. Дискретизация по времени, направлениям скоростей и пространству приводит к следующему уравнению, которое используется в методе LBM:

$$f_i(\mathbf{x} + \mathbf{c}_i\delta_t, t + \delta_t) - f_i(\mathbf{x}, t) = \Omega(f_i), \tag{2}$$

где $\mathbf{c}_i$ — вектор i-й скорости (см рис. 1) и $\delta_t = 1$ — шаг по времени.

Пространство представляется в виде решетки (кубической, гексагональной и т.д.), в ячейках которой располагаются частицы. Заранее заданы набор допустимых направлений скоростей c_i и соответствующие им функции распределения f_i. Мы используем модель D3Q19 [12, 20] — трехмерную кубическую пространственную решетку с 19 направлениями скоростей:

$$||\mathbf{c}_i|| = \begin{pmatrix} 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 \end{pmatrix}.$$

Интеграл столкновений $\Omega(f_i)$ задает изменение функции распределения во времени. Мы выбрали классическое приближение этого интеграла в форме BGK (Bhatnagar, Gross, Krook [3]) в случае отсутствия массовых сил. Для компактной записи потребуется равновесная функция распределения f_i^{eq}. Она аппроксимируется [6, 24] уравнением

$$f_i^{\text{eq}}(\rho, \mathbf{v}) = \omega_i \rho \left[1 + \frac{(\mathbf{c}_i\mathbf{v})}{c_s^2} + \frac{(\mathbf{c}_i\mathbf{v})^2}{2c_s^4} - \frac{(\mathbf{v}, \mathbf{v})}{2c_s^2}\right] \tag{3}$$

и характеризует состояние равновесия, к которому со временем стремятся частицы внутри ячейки.

Скорость звука c_s вычисляется как $c_s = 1/\sqrt{3}$, а весовые коэффициенты w_i имеют следующий вид:

$$w_i = \begin{cases} \dfrac{12}{36}, & \text{если } i = 0, \\ \dfrac{2}{36}, & \text{если } i = 1, \ldots, 6, \\ \dfrac{1}{36}, & \text{если } i = 7, \ldots, 18. \end{cases}$$

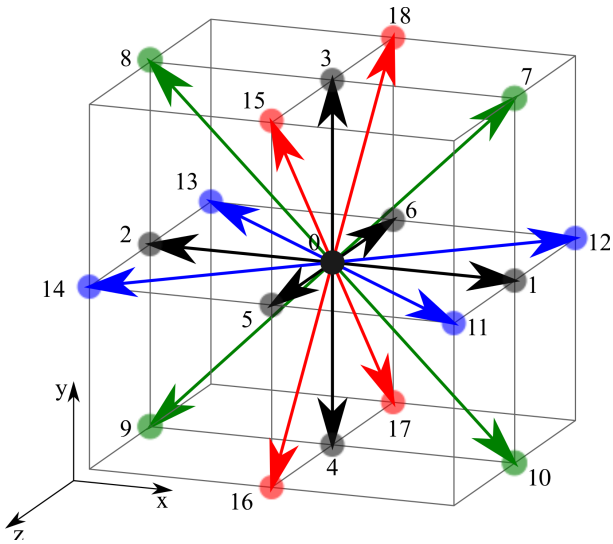

Рис. 1. Сетка модели D3Q19 с обозначенными направлениями скоростей c_i: $c_0 = (0, 0, 0)$, $c_1 = (1, 0, 0)$, … , $c_7 = (1, 1, 0)$ и т.д.

Тогда приближенный интеграл столкновений в виде BKG записывается для каждой функции распределения следующим образом:

$$\Omega(f_i) = \frac{1}{\tau_f}(f_i^{\text{eq}} - f_i). \tag{4}$$

Здесь τ_f — параметр релаксации, связанный [1, 2, 17] с кинематической вязкостью:

$$\nu = c_s^2\left(\tau_f - \frac{1}{2}\right)\delta t.$$

Значения плотности и скорости среды в некоторой ячейке вычисляются в методе LBM на основе функций распределения [1]:

$$\rho = \sum_0^{18} f_i, \quad \mathbf{v} = \frac{\sum_0^{18} \mathbf{c}_i f_i}{\rho}. \tag{5}$$

Решение уравнения (2) принято выполнять в два этапа: этап столкновения (6a) и этап распространения (6b):

$$f_i(\mathbf{x}, t + \delta_t) = f_i(\mathbf{x}, t) + \frac{1}{\tau_f}(f_i^{\text{eq}} - f_i), \tag{6a}$$

$$f_i(\mathbf{x} + \mathbf{c}_i\delta_t, t + \delta_t) = f_i(\mathbf{x}, t + \delta_t). \tag{6b}$$

В реализованной модели LBM плотность пересчитывается в давление как $p = c_s^2\rho$. Область моделирования представляет собой куб со сторонами $N_x \times N_y \times N_z$ ячеек. Последние бывают двух видов: непротекаемые, если течение через ячейку невозможно, и протекаемые, если возможно.

После этапа распространения в ячейках, примыкающих к непротекаемым ячейкам и расположенных на гранях области моделирования, неизвестны некоторые функции распределения f_i. Значения в них задаются с помощью краевых и граничных условий соответственно. В качестве краевых условий мы используем условие on-grid bounceback [24], имеющее первый порядок точности.

На гранях, параллельных плоскости OYZ, мы используем два типа граничных условий: заданное давление [14] и заданная скорость [7]. Граничные условия на остальных гранях периодические.

2. Некоторые сведения о технологии CUDA. CUDA — технология, позволяющая осуществлять высокопроизводительные расчеты на видеокартах. Она была анонсирована в ноябре 2006 г. GeForce 8800 была первой видеокартой, основанной на новом GPU G80, который поддерживал язык C, новую SIMT-модель исполнения и барьерную синхронизацию между нитями. Спустя пять лет она является лидирующей в области программирования на GPU. Технология CUDA включает в себя аппаратную и программную стороны.

2.1. Аппаратная сторона. В качестве примера рассмотрим видеокарту GeForce GTX580, работающую на архитектуре Fermi. Она содержит 512 ядер CUDA, объединенных в 16 потоковых мультипроцессоров (SM, Streaming Multiprocessor) и работающих на частоте 1544 МГц. Один мультипроцессор SM составляют 32 ядра, 4 сопроцессора для операций со специальными функциями, два планировщика, два обработчика инструкций и 16 блоков чтения/записи. Каждое ядро включает в себя сопроцессоры для операций с целыми числами и числами с плавающей точкой.

На каждом мультипроцессоре SM расположено 64 Кб памяти, которая может быть разделена между кэшем первого уровня и доступной разделяемой памятью. К ней имеет доступ только конкретный мультипроцессор SM. Кэш второго уровня имеет размер 768 Кб и доступен всем мультипроцессорам SM одновременно. Самой медленной и самой большой является глобальная память. Ее размер составляет 1.5 Гб. Следует учитывать особенность чтения последней: она читается в быстрый кэш строками по 32, 64 или 128 байтов. Поэтому при обращении к соседним ячейкам происходит объединение запросов, и чтение происходит существенно быстрее.

2.2. Программная сторона. Разработчику доступны: глобальная память в DRAM-устройстве (Dynamic Random Access Memory); разделяемая память непосредственно на каждом мультипроцессоре SM, к которой имеют доступ только нити, выполняющиеся на одном SM; текстурная и константная память. Последние две так же расположены в DRAM-устройстве, но благодаря особенностям работы устройства обращение к ним выполняется быстрее, чем к глобальной памяти напрямую. В регистры помещаются локальные переменные внутри функций, выполняемых на видеокарте.

Функция, выполняемая на GPU и запускаемая с CPU, называется ядром. Разработчик указывает, сколько нитей будут выполнять одно и то же ядро, и запускает его. Здесь следует описать иерархию нитей на видеокарте. Нити объединяются в блоки, размерность которых может быть одномерной, двумерной или трехмерной. Блоки объединяются в сетку, являющуюся высшим уровнем объединения. Размерность у нее так же одномерная, двумерная или трехмерная — на выбор разработчика.

Блок нитей всегда выполняется только на одном SM. С другой стороны, на одном SM может выполняться несколько блоков. За распределением блоков по мультипроцессорам следит компьютер. Он принимает решение на основе данных об объеме требуемых блоку ресурсов и количестве доступных регистров и памяти на SM. Одновременно выполняются не все нити одного блока, а пачки по 32 нити, называемые варпами. Только нити одного варпа выполняются физически одновременно.

3. Реализация метода LBM в рамках технологии CUDA. Для достижения максимальной производительности в реализациях LBM на графических процессорах следует учитывать особенности доступа к памяти и особенности архитектуры видеокарт. Например, в [19] для объединения запросов в памяти и максимального использования разделяемой памяти алгоритм усложнен и разбит на несколько этапов. Авторы делят область на равные части, каждой части в соответствие ставится блок нитей. Каждой нити блока соответствует одна пространственная ячейка (т.е. число нитей в блоке должно быть равно числу ячеек в части области).

1. Нить читает значения функций распределения по всем направлениям скоростей в своей ячейке и записывает их в разделяемую память.

2. В каждом блоке запускается расчет этапов столкновения и распространения с периодическими граничными условиями, результат записывается обратно в глобальную память.

3. Специальное ядро правильным образом меняет местами значения f_i на границах частей.

4. Применяются граничные условия. Подробнее исходный алгоритм описан в [19].

В трехмерном случае помимо скорости работы также важен объем расходуемой памяти. Обычно на этапе распространения используется два одинаковых массива памяти: "старый" и "новый". Значения функций распределения читаются из первого и записываются во второй. Такой подход очень прост и быстр, но требует двойного объема памяти. Мы предлагаем технику, позволяющую сократить объем дополнительных массивов.

Функции распределения для каждого направления скорости хранятся в виде одномерных массивов размера N, где N — число ячеек разбиения, $N = N_x \times N_y \times N_z$. Этот подход известен как SoA (Structure of Arrays, структура массивов) и используется для обеспечения объединенного доступа в глобальную память на видеокарте. Будем выполнять этап распространения для каждого направления скорости отдельно. Выделим дополнительный массив $f_{\text{add}}[N]$ длины N. Фиксируем i, читаем значения $f_i[k]$ и записываем их в $f_{\text{add}}[dest(k)]$ с учетом смещения. Как только $k = N - 1$, для всех f_i этап распространения завершен. После обмена указателей $\text{swap}(f_i, f_{\text{add}})$ то же самое производится для следующего массива f_{i+1} и т.д.

Использование такой техники сокращает объем необходимой для вычислений памяти с $2 \times QNC$ до $(Q+1)NC$, где Q — число дискретных направлений скорости (в модели D3Q19 это 19), $N = N_x \times N_y \times N_z$ и C — некоторый объем памяти, который занимает описание одной пространственной ячейки. При этом распространение происходит сразу во всем объеме, поэтому нет необходимости в специальном ядре.

Одна итерация модели, таким образом, состоит из следующих шагов: распространение, применение граничных условий и учет столкновений.

Рассмотрим объем памяти, необходимый для полного описания одной пространственной ячейки, чуть подробнее. В шаблоне D3Q19 скорость дискретизирована на 19 направлений. Это 19 чисел с плавающей запятой, занимающих в сумме $19 \times 4 = 76$ байт, плюс одна дополнительная переменная, необходимая для реализации приема, описанного выше. В итоге получается более 80 байт на ячейку. Максимальный размер DRAM-памяти на GPU сейчас составляет 6 Гб, что эквивалентно $(6 \times 1024^3/80)^{1/3} \approx 430$), т.е. сетке 430^3. Понятно, что для реальных задач этой размерности недостаточно. Можно либо подгружать область из оперативной памяти на видеокарту частями, либо использовать несколько GPU. Второй способ обещает гораздо бóльшую производительность, поэтому мы выбрали его.

4. Использование нескольких видеокарт. Технология CUDA не предоставляет встроенных механизмов обмена данными между видеокартами, расположенными на физически разных узлах. Эту задачу принято возлагать на интерфейс MPI. Интерфейс MPI и технология CUDA выполняют разные задачи, не мешают работе, а дополняют ее. Поэтому сначала задача разбивается на подзадачи, а затем посредством MPI информация об области рассылается различным процессам. Каждый процесс уже использует CUDA и работает с собственной видеокартой. Подобный подход используется в [8, 11, 15, 16, 21].

Мы разбиваем расчетную область на слои по оси Z так, чтобы получившиеся подобласти (юниты) полностью помещались в память видеокарты: $N = N_x \times N_y \times N_z < 6 \times 1024^3/80$ (рис. 2). Информация о структуре объема рассылается на узлы, и запускается расчет. Как было отмечено выше, одна итерация состоит из трех шагов. Этап столкновения и применение граничных условий могут рассчитываться на каждом узле совершенно независимо, а вот распространение требует информации от соседних узлов. Поэтому он выполняется в два подэтапа. На первом выполняется распространение внутри области с периодическими граничными условиями. На втором данные из граничных ячеек считываются на хост и рассылаются средствами MPI соответствующим соседним областям. Обновленные значения загружаются обратно на видеокарту. Одна итерация модели теперь состоит из следующих шагов: распространение, сшивка по MPI, применение граничных условий и учет столкновений.

В качестве примера мы провели расчет области $256 \times 512 \times 510$ для теста Пуазейля (радиус трубы $r = 50$) на 1, 2, 5, 10, 15 и 30 видеокартах. Использовались NVIDIA Tesla X2070, стоящие на суперкомпьютере "Ломоносов". Результаты производительности представлены на графике (рис. 3). Виден линейный рост при увеличении числа видеокарт: объем данных на каждой из них уменьшается — расчет происходит быстрее, а обмен данными по MPI происходит параллельно и занимает фиксированное время, не зависящее от числа задействованных узлов. Производительность 370 MLUPS, полученная на 30 GPU, превосходит почти в два раза результаты, полученные для 64 CPU Intel Xeon в [22].

5. Верификация. Для проверки реализованной модели нами были проведены два тестовых расчета: расчет ламинарного течения Пуазейля в цилиндрической трубе и расчет коэффициента лобового сопротивления шара в зависимости от числа Рейнольдса. Результаты моделирования хорошо согласуются с аналитическими решениями. Во обоих случаях полный цикл расчета составил около 2×10^4 итераций. Критерием остановки было изменение интересующей величины менее, чем на 0.1% за 10^3 итераций.

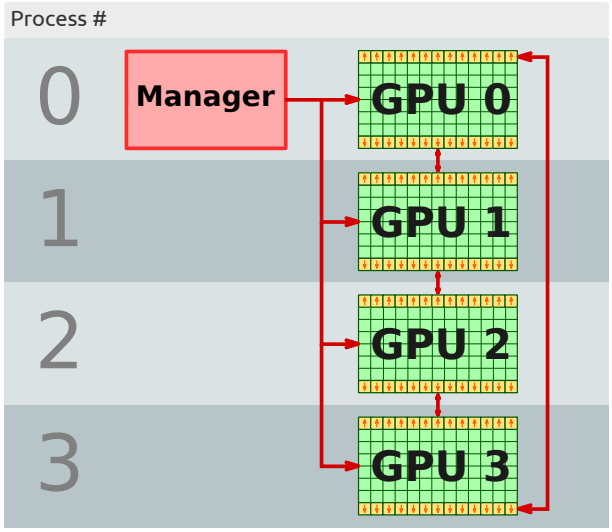

Рис. 2. Одномерное разбиение области на слои и распределение по нескольким процессам

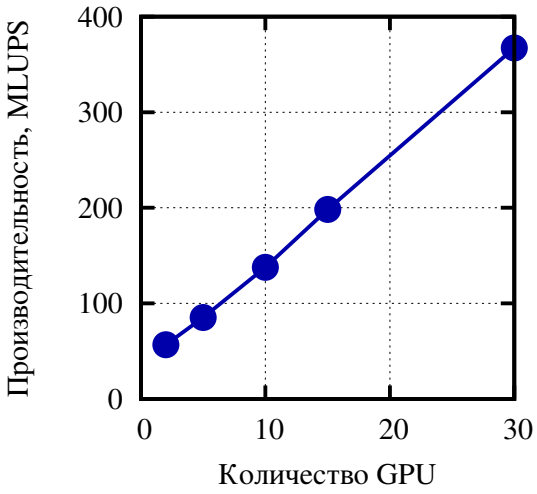

Рис. 3. Зависимость производительности от числа видеокарт

В начальный момент времени во всех протекаемых и примыкающих к ним непротекаемых (для реализации on-grid bounceback краевого условия) ячейках были выбраны значения $f_i(\mathbf{x}, t=0) = f_i^{\mathrm{eq}}$.

5.1. Течение Пуазейля. Размер расчетной области составил $N_x \times N_y \times N_z = 192 \times 192 \times 256$. Цилиндрическая труба задавалась в соответствии с правилом: если

$$\left(y_i - \frac{N_y}{2}\right)^2 + \left(z_i - \frac{N_z}{2}\right)^2 < R^2,$$

то через ячейку возможно течение. Иначе ячейка считается непротекаемой. Здесь y_i и z_i — координаты центра ячейки, $R = 90$ — радиус трубы.

На гранях, параллельных плоскости OYZ, было выбрано граничное условие в виде заданного давления, причем при $x = 0$ и $x = N_x$ давление соответствовало плотности $\rho_1 = 1.001$ и $\rho_2 = 0.999$. На остальных гранях были заданы периодические граничные условия. Длина трубы составила $L = N_x = 192$.

Аналитическая зависимость модуля скорости в поперечном сечении трубы от расстояния r до ее центра имеет вид

$$v(r) = \frac{p_1 - p_2}{4\eta L}\left(R^2 - r^2\right),$$

где η — динамическая вязкость, $\eta = \rho\nu$.

В случае прямоугольной сетки, использованной в модели, эта зависимость принимает форму

$$v(r) = \frac{p(x=0) - p(x=N_x)}{4\eta L}\left(R^2 - ((y_i - N_y/2)^2 + (z_i - N_z/2)^2)\right).$$

Полученный в численном эксперименте профиль скорости показал хорошее соответствие аналитическому решению (см. рис. 4).

5.2. Лобовое сопротивление шара. Размер расчетной области составил $N_x \times N_y \times N_z = 64 \times 64 \times 64$. Шар задавался в соответствии с правилом: если

$$(x_i - N_x/2)^2 + (y_i - N_y/2)^2 + (z_i - N_z/2)^2 > R^2,$$

то через ячейку возможно течение. Иначе ячейка считается непротекаемой. Аналогично тесту с течением Пуазейля, здесь x_i, y_i и z_i — координаты центра ячейки, а $R = 10$ — радиус шара.

На гранях, параллельных плоскости OYZ, было выбрано граничное условие в виде заданной скорости течения, причем для различных чисел Рейнольдса $\mathrm{Re} = 2Ru_x/\nu$ при $x = 0$ и $x = N_x$ были выбраны следующие значения скорости: $u_x = 0.125 \times 10^{-3}, 0.25 \times 10^{-3}, 0.5 \times 10^{-3}, 1 \times 10^{-3}, 2 \times 10^{-3}, \ldots, 8 \times 10^{-3}$ и $u_y = u_z = 0$. На остальных гранях заданы периодические граничные условия.

Согласно теории Стокса, для шара при малых значениях числа Рейнольдса коэффициент сопротивления выражается формулой

$$C_d = \frac{24}{\mathrm{Re}}.$$

Полученные в численном эксперименте значения коэффициента сопротивления показали хорошее согласие с этой аналитической зависимостью (см. рис. 5).

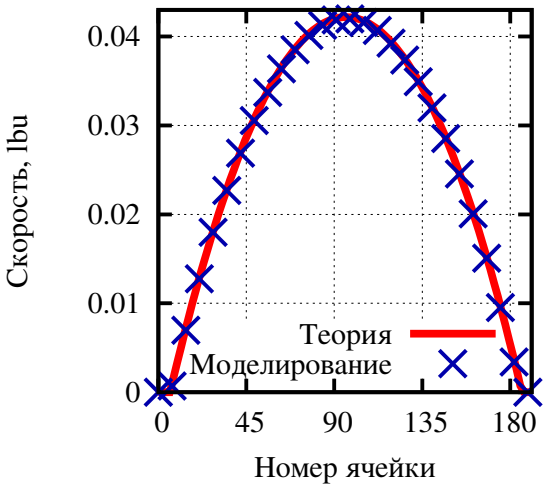

Рис. 4. Сравнение аналитического и численного значений скорости течения Пуазейля

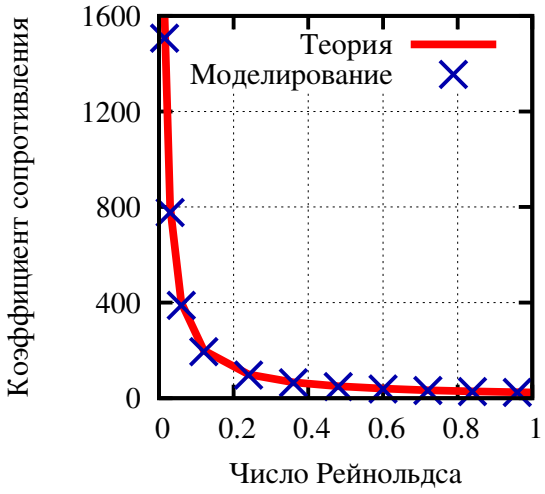

Рис. 5. Сравнение аналитического и численного значений коэффициента сопротивления

6. Результаты и дальнейшее развитие. В настоящей работе реализована модель однокомпонентного течения с использованием технологии CUDA вычислений на графических ускорителях. Проведена оценка масштабируемости для реального моделирования на суперкомпьютерном комплексе “Ломоносов”, установленном в Научно-исследовательском вычислительном центре МГУ им. М. В. Ломоносова. Линейный рост производительности от числа видеокарт подтверждает целесообразность использования многомасштабного параллелизма: между несколькими видеокартами на верхнем уровне и внутри каждой из них на нижнем. Подход весьма прост, применим не только в случае метода LBM, но и к другим схожим по структуре задачам.

В рамках текущей реализации предложен прием, позволяющий уменьшить необходимый объем дополнительной памяти для работы алгоритма при сохранении простоты реализации. Следует отметить, что проблема нехватки памяти в задаче LBM — одна из ключевых (ведь в D3Q19 на каждую ячейку приходится хранить информацию о ее типе и 19 плотностях по направлениям).

Выполнена верификация реализованного алгоритма с помощью двух тестовых расчетов: расчет стационарного течения Пуазейля и расчет коэффициента лобового сопротивления шара. Результаты численного моделирования хорошо согласуются с теоретическими значениями в обоих случаях.

В дальнейших планах предусматривается дополнительная оптимизация использования памяти внутри каждой видеокарты: хранение значений плотностей только в ячейках, в которых возможно течение. Кроме того, представляется важным переход к граничным условиям повышенной точности и проведение дополнительных расчетов на пористых объемах с известными характеристиками.

СПИСОК ЛИТЕРАТУРЫ

1. *Кривовичев Г.В.* О применении интегро-интерполяционного метода к построению одношаговых решеточных кинетических схем Больцмана // Вычислительные методы и программирование. 2012. **13**, № 1. 19–27.
2. *Куперштох А.Л.* Трехмерное моделирование двухфазных систем типа жидкость–пар методом решеточных уравнений Больцмана на GPU // Вычислительные методы и программирование. 2012. **13**, № 1. 130–138.
3. *Bhatnagar P.L., Gross E.P., Krook M.* A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems // Phys. Rev. 1954. N 94. 511–525.

4. *Boyd J., Buick J., Cosgrove J.A., Stansell P.* Application of the lattice Boltzmann model to simulated stenosis growth in a two-dimensional carotid artery // Phys. in Medicine and Biology. 2005. N 50. 4783–4796.
5. *Boyd J., Buick J., Green S.* A second-order accurate lattice Boltzmann non-Newtonian flow model // J. Phys. A: Mathematical and General. 2006. N 39. 14241–14247.
6. *Chen S., Martinez D., Mei R.* On boundary conditions in lattice Boltzmann methods // Phys. of Fluids. 1996. **8**, N 9. 2527–2536.
7. *Hecht M., Harting J.J.* Implementation of on-site velocity boundary conditions for D3Q19 lattice Boltzmann simulations // J. of Statistical Mechanics: Theory and Experiment. 2009. N 1. P01018.
8. *Jacobsen D.A., Thibault J.C., Senocak I.* An MPI–CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters // Proc. 48th AIAA Aerospace Sciences Meeting. Reston: American Institute of Aeronautics and Astronautics. 2010. 1–15.
9. *Jiang Z., Wu K., Couples G.D., Ma J.* The impact of pore size and pore connectivity on single-phase fluid flow in porous media // Advanced Engineering Materials. 2011. N 13. 208–215.
10. *Kang Q., Zhang D., Lichtner P., Tsimpanogiannis I.* Lattice Boltzmann model for crystal growth from supersaturated solution // Geophysical Research Letters. 2004. N 31. L21604.
11. *Köstler H.* Numerical algorithms on multi-GPU architectures // Proc. 2nd Int. Workshops on Advances in Computational Mechanics. Yokohama, 2010. 1–47.
12. *Kutay M.E., Aydilek A.H., Masad E.* Laboratory validation of lattice Boltzmann method for modeling pore-scale flow in granular materials // Computers and Geotechnics. 2006. N 33. 381–395.
13. *Lim C.Y., Shu C., Niu X.D., Chew Y.T.* Application of lattice Boltzmann method to simulate microchannel flows // Physics of Fluids. 2002. N 7. 2299–2308.
14. *Narvaez A., Harting J.* Evaluation of pressure boundary conditions for permeability calculations using the lattice-Boltzmann method // Advances in Applied Mathematics and Mechanics. 2010. **2**, N 5. 685–700.
15. *Obrecht C., Kuznik F., Tourancheau B., Roux J.-J.* Multi-GPU implementation of the lattice Boltzmann method // Computers and Mathematics with Applications. 2011 (doi: 10.1016/j. canwa.2011.02.020).
16. *Obrecht C., Kuznik F., Roux J.-J.* The TheLMA project: multi-GPU implementation of the lattice Boltzmann method // Int. J. of High Performance Computing Applications. 2011. N 3. 295–303.
17. *Qian Y.H., D'Humieres D., Lalleman P.* Lattice BGK models for Navier–Stokes Equation // Europhysics Letters. 1992. **17**, N 6. 479–484.
18. *Sukop M.C., Thorne D.T.* Lattice Boltzmann modeling: an introduction for geoscientists and engineers. Berlin: Springer, 2007.
19. *Tolke J.* Implementation of a lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA // Computing and Visualization in Science. 2010. **13**, N 1. 29–39.
20. *Raabe D.* Overview of the lattice Boltzmann method for nano- and microscale fluid dynamics in materials science and engineering // Modelling Simul. Mater. Sci. Eng. 2004. N 12. R14–R45.
21. *Xian W., Takayuki A.* Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster // Parallel Computing. 2011. **37**, N 9. 521–535.
22. *Welleina G., Lammers P., Hagera G., Donatha S., Zeisera T.* Towards optimal performance for lattice Boltzmann applications on terascale computers // Proc. Parallel CFD Conference. Amsterdam: Elsevier, 2006. 31–40.
23. *Ye Z.* Lattice Boltzmann based PDE solver on the GPU // Visual Comput. 2008. **24**, N 5. 323–333.
24. *Zou Q., He X.* On pressure and velocity boundary conditions for the lattice Boltzmann BGK model // Phys. of Fluids. 1997. **9**, N 6. 1591–1599.

Поступила в редакцию
16.02.2012