УДК 519.688

РЕШЕНИЕ УРАВНЕНИЯ БОЛЬЦМАНА НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

Ю. Ю. Клосс[1], Ф. Г. Черемисин[2], П. В. Шувалов[3]

Рассматривается реализация консервативного проекционного метода решения уравнения Больцмана на графических процессорах. В качестве платформы для вычислений использована технология NVidia CUDA. Разработаны методы оптимальной реализации солвера для двумерной геометрии, а также методы задания граничных условий, реализации геометрии и хранения интегрирующих сеток. На основе разработанной методики проведены расчеты задачи о медленном течении газа в прямоугольной каверне и задачи о распространении ударной волны в узком канале. Показана эффективность использования графических процессоров для расчета медленных и высокоскоростных течений разреженного газа.

Ключевые слова: ГПУ, CUDA, массивно-параллельные вычисления, уравнение Больцмана, разреженный газ.

1. Введение. Моделирование течений разреженного газа имеет большое количество применений, среди которых течения в микро- и наноустройствах, изучение структуры ударных волн, явлений в кнудсеновском пограничном слое. Среди методов моделирования широко распространены методы, основанные на релаксационной модели уравнения Больцмана [1], и имитационные методы статистического моделирования [2].

В настоящей статье используется проекционный консервативный метод решения уравнения Больцмана [3–6]. Уравнение решается для двух моделей молекулярного потенциала — модели твердых сфер и потенциала Леннарда–Джонса. Его решением является функция распределения, определенная в фазовом шестимерном пространстве. Задача является сложной в вычислительном смысле и требует высокой эффективности алгоритма. Появляющиеся в последние годы технологии массивно-параллельных вычислений являются перспективными для таких сложных задач. Недавние работы в разных областях физического моделирования показали высокую эффективность применения графических процессорных устройств (ГПУ), например [7–10]. В настоящей статье описаны методы решения двумерных задач в плоской геометрии, однако разработанная методика может быть расширена на трехмерные задачи и задачи с неструктурными сетками [11].

2. Математические и теоретические основы. При моделировании макропараметры газа, такие как плотность $n(\boldsymbol{x},t)$, температура $T(\boldsymbol{x},t)$, скорость $\boldsymbol{U}(\boldsymbol{x},t)$, и др., вычисляются численным интегрированием по скорости $\boldsymbol{\xi}$ функции распределения $f(\boldsymbol{\xi},\boldsymbol{x},t)$, которая определяется из решения уравнения Больцмана

$$\frac{\partial f}{\partial t}+\boldsymbol{\xi}\,\frac{\partial f}{\partial \boldsymbol{x}}=I(f,f). \tag{1}$$

Уравнение (1) решается конечно-разностным методом на фиксированной сетке в пространстве скоростей и координат. Расчетная область в пространстве координат представлена совокупностью соединенных доменов из прямоугольных сеток. Это позволяет считать геометрии, включающие в себя каналы и препятствия, оперируя при этом прямоугольной сеткой. В качестве пространства скоростей выбирается сферический домен Ω на равномерной сетке с шагом $h_v=\dfrac{2V_{\max}}{N_0}$. Радиус сферы, ограничивающей скоростное пространство, равен $V_{\max}$; центр сферы может быть сдвинут относительно нуля в зависимости от задачи. Максимальная скорость $V_{\max}$ принималась равной $4.8\sqrt{\dfrac{kT}{m}}$. Число узлов вдоль одной координаты N_0 в большинстве расчетов было равно 20. Для $N_0=20$ количество узлов скоростного пространства равно 4224.

[1] РНЦ Курчатовский институт, пл. Курчатова, д. 1, 123182, Москва; доцент, начальник отдела, e-mail: kloss@mnpt.kiae.ru

[2] Вычислительный центр им. А. А. Дородницына РАН, ул. Вавилова, 40, 117967, Москва; профессор, гл. науч. сотр., e-mail: tcherem@ccas.ru

[3] Московский физико-технический институт, факультет общей и прикладной физики, Институтский пер., д. 9, 141700, Московская обл., г. Долгопрудный; студент, e-mail: shuvalov.pavel@gmail.com

Уравнение (1) расщепляется на две части — уравнение переноса, представляющее левую часть, и задачу релаксации. Расщепление имеет вид $S_\tau = A_{\tau/2} C_\tau A_{\tau/2}$, где S_τ — оператор, переводящий решение в момент времени t_0 в решение в момент времени $t_0 + \tau$, A_τ — оператор решения уравнения переноса $\dfrac{\partial f}{\partial t} + \boldsymbol{\xi} \dfrac{\partial f}{\partial \boldsymbol{x}} = 0$ с шагом τ и C_τ — оператор решения задачи релаксации $\dfrac{\partial f}{\partial t} = I(f, f)$.

Левая часть уравнения (1) аппроксимируется явной разностной схемой первого порядка:

$$\frac{f_{i,j}^{n+1} - f_{i,j}^{n}}{\tau} + |\xi_x| \frac{f_{i+\mathrm{sgn}(\xi_x),j}^{n} - f_{i,j}^{n}}{h_x} + |\xi_y| \frac{f_{i,j+\mathrm{sgn}(\xi_y)}^{n} - f_{i,j}^{n}}{h_y}. \tag{2}$$

Напомним, что $I(f,f) = \int\limits_{-\infty}^{\infty} \int\limits_{0}^{2\pi} \int\limits_{0}^{b_m} \big(f' f'_* - f f_*\big) g\, b\, db\, d\varphi d\xi_*$ — интеграл столкновений для одноатомного газа. Введя обозначение $\phi(\xi_\gamma) = \delta(\xi - \xi_\gamma) + \delta(\xi_* - \xi_\gamma) - \delta(\xi' - \xi_\gamma) - \delta(\xi'_* - \xi_\gamma)$, где δ — дельта-функция Дирака, значение интеграла столкновений в точке ξ_γ может быть представлено в виде

$$I(\xi_\gamma) = \frac{1}{4} \int\limits_{-\infty}^{\infty} \int\limits_{-\infty}^{\infty} \int\limits_{0}^{2\pi} \int\limits_{0}^{b_m} \phi(\xi_\gamma) \big(f' f'_* - f f_*\big) g\, b\, db\, d\varphi\, d\xi\, d\xi_* \,. \tag{3}$$

Интегрирование в уравнении (3) происходит численно на сетке $\xi_{\alpha_\nu}, \xi_{\beta_\nu}, b_\nu, \varphi_\nu$ из N_ν узлов в восьмимерной области $\Omega \times \Omega \times [0, 2\pi] \times [0, b_{\max}]$. Следует отметить, что скорости после столкновения ξ'_{α_ν} и ξ'_{β_ν} в общем случае не попадают в узлы скоростной сетки. Для каждой скорости ξ'_{α_ν} определяется пара узлов ξ_{λ_ν} и ξ_{μ_ν}, аппроксимирующая ее на скоростной сетке, а для скорости ξ'_{β_ν} выбираются симметричные относительно суммарной скорости $g_\nu = \xi_{\alpha_\nu} + \xi_{\beta_\nu}$ узлы $\xi_{\lambda_\nu + s_\nu}$ и $\xi_{\mu_\nu - s_\nu}$, после чего дельта-функция в формуле (3) представляется в виде $\delta(\xi'_{\alpha_\nu} - \xi_\gamma) = (1 - r_\nu)\delta(\xi_{\lambda_\nu} - \xi_\gamma) + r_\nu \delta(\xi_{\mu_\nu} - \xi_\gamma)$, где коэффициент r_ν определяется из закона сохранения энергии: $\big(\xi'_{\alpha_\nu}\big)^2 + \big(\xi'_{\beta_\nu}\big)^2 = (1 - r_\nu)\big(\xi^2_{\lambda_\nu} + \xi^2_{\mu_\nu}\big) + r_\nu \big(\xi^2_{\lambda_\nu + s_\nu} + \xi^2_{\mu_\nu - s_\nu}\big)$.

Для аппроксимации произведения функций распределения в узлах после столкновения используется степенная интерполяция

$$f'_{\alpha_\nu} f'_{\beta_\nu} = (f_{\lambda_\nu} f_{\mu_\nu})^{1 - r_\nu} + (f_{\lambda_\nu + s_\nu} f_{\mu_\nu - s_\nu})^{r_\nu}. \tag{4}$$

Она обеспечивает строгое равенство нулю интеграла столкновений от максвелловской функции распределения. Это свойство особенно важно для моделирования медленных течений и течений, содержащих слабовозмущенные области, где оно значительно повышает точность вычислений.

Пусть решение имеет вид $f = f_M + \varepsilon f^{(1)}$, где $\varepsilon f^{(1)}$ — отклонение от максвелловского распределения, $\varepsilon \ll 1$. Подставив это решение в интеграл столкновений и учитывая (4), получим

$$I(f,f) = I(f_M, f_M) + 2\varepsilon I\big(f_M, f^{(1)}\big) + \varepsilon^2 I\big(f^{(1)}, f^{(1)}\big). \tag{5}$$

Так как главная часть интеграла $I(f_M, f_M)$ вычисляется точно, то при некоторых естественных предположениях о погрешности вычисления интеграла из (5) следует, что погрешность вычисления интеграла столкновений уменьшается в $(2\varepsilon)^{-1}$ раз.

Вычисление интеграла столкновений может быть разделено на подготовительный этап, выполняющийся один раз перед расчетом, и расчет вклада интеграла в каждый узел на каждом временно́м шаге. Этап подготовки можно разделить на несколько подэтапов.

1. Генерация интегрирующей восьмимерной сетки $\xi_{\alpha_\nu}, \xi_{\beta_\nu}, b_\nu, \varphi_\nu$ на основе сеток Коробова [12]. Сетки Коробова обеспечивают бóльшую точность вычисления многомерных интегралов, чем сетки случайных узлов. На этом этапе создается несколько интегрирующих сеток, каждая с одинаковым сдвигом всех узлов, и на каждом шаге по времени при расчете выбирается случайным образом одна из них. Точки, изначально не попавшие в область Ω, исключаются из интегрирующей сетки.

2. Для каждой точки интегрирующей сетки ξ'_{α_ν} и ξ'_{β_ν} вычисляются скорости после столкновения $\xi_{\alpha_\nu}, \xi_{\beta_\nu}, b_\nu, \varphi_\nu$. Вычисления разлетных скоростей выполняются на основе выбранного потенциала (см. [5]).

3. На основе описанного выше метода для каждой пары скоростей ξ'_{α_ν} и ξ'_{β_ν} вычисляются коэффициенты r_ν.

3. Реализация на ГПУ. Для реализации алгоритмы использовалась технология NVidia CUDA [13], которая предоставляет разработчику набор расширения языка C++ для запуска вычислительного кода на ГПУ и управления памятью ГПУ. Все запускаемые процессы группируются в блок потоков, которые в свою очередь группируются в сетку блоков. На данный момент максимально возможно число запускаемых потоков в блоке равно 512. Все потоки внутри блока не обязательно выполняются в одно время,

но они могут использовать общую память, данные в которой хранятся в течение времени жизни блока. Для адресации потоков внутри блока используется встроенная переменная `threadIdx`. Индекс потока в блоке может быть представлен как двумерный или трехмерный вектор из целых неотрицательных чисел: `threadIdx.x`, `threadIdx.y`, `threadIdx.z`, однако произведение размеров по каждой координате все равно не должно превышать 512. Такая адресация удобна для запуска ядра на данных, представляющих сеточную функцию в физическом пространстве. Для адресации блока в сетке блоков используется встроенная переменная `blockIdx`, которая может иметь двумерную адресацию на основе переменных `blockIdx.x` и `blockIdx.y`. Размер сетки блока может быть много больше, чем размер блока, — ограничение на одно измерение составляет 65 535 блоков. Каждый процесс имеет также доступ к константам, определяющим размер блока и сетки блоков – `blockDim` и `gridDim`. Для запуска вычислительных ядер на ГПУ используется команда `Kernel<<<dimGrid, dimBlock>>>(params)`, которая запускает ядро `Kernel` с параметрами params в виде сетки блоков размером `dimGrid` и с размером блока `dimBlock`.

Рассмотрим адресацию в прямоугольной области размером $S_x \times S_y$, такой, что $S_x \leqslant S_y$. Для исключения конфликтов каждый поток ведет расчет отдельной точки в координатном пространстве. Индекс ячейки по координате X может принимать значения от 0 до $S_x - 1$. Однако для удобства расчета граничных условий к расчетной области добавляются ячейки с индексами -1 и S_x (аналогично для координаты Y). Перед запуском расчета определяется константа $X_{\max}$, определяющая число потоков в блоке. Вычислительный код запускается в следующей конфигурации: размер блока $\left[S_x/X_{\max}\right] \times 1 \times 1$, размер сетки блоков $\left(S_x \bmod X_{\max}\right)S_y \times S_v$, где S_v — размер массива значений функции распределения в одной пространственной ячейке. При запуске вычислительного ядра для интеграла столкновений используется такая же конфигурация, за исключением размера блока по координате z; он равен 1, так как обход всех значений в пространстве скоростей не требуется. Наиболее очевидным порядком адресации значения функции распределения $f(x, y, v_i)$ является адресация по координате, а потом — по скоростному индексу. Однако при таком подходе соседние потоки будут обращаться к удаленным на расстояние S_v ячейкам памяти, что исключает согласованный доступ к памяти ГПУ. Согласованный доступ к памяти позволяет объединить несколько запросов к памяти в один для увеличения быстродействия, для него существуют следующие условия: поток с номером n обращается к слову с номером n в памяти (в нашем случае размер слова равен 4 байтам); минимальный адрес слова в группируемом блоке должен быть кратен 16 размерам слова.

Если значения для соседних по оси X ячеек с одинаковой скоростью будут расположены в соседних ячейках, то будет удовлетворено первое условие. Минимальный адрес слова в блоке будет зависеть от размеров блока и в общем случае может не быть кратным 16 (размер расчетной области может быть произвольным, особенно с учетом фиктивных ячеек). Для устранения этого недостатка вводятся пустые ячейки, используемые только для выравнивания. Вышесказанное иллюстрируется кодом функции `getIdx(x, y, v)`, которая отображает пространственные координаты и скоростной индекс в линейную область памяти следующим образом: $(x, y, v) \to x + 16 + (y + 1)(S_x + 2 + \Delta) + v(S_x + 2 + \Delta)(S_y + 2)$, где Δ определяется из соотношения $(S_x + 2 + \Delta) \equiv 0 \pmod{16}$.

Пока не были сформулированы ограничения на индексы в скоростном пространстве. Использование кубического скоростного пространства позволило бы легко определять значение скорости по известным индексам по каждой координате. Но так как описываемый метод оперирует сферическим скоростным пространством, то такой подход не применим. Определять скорость по индексу требуется для решения уравнения переноса и вычисление макропараметров. Для этих целей используется вспомогательный массив `speedMap[]`, хранящий значения проекции скорости на каждую координату. Этот массив инициализируется перед запуском расчета и хранится в константной памяти ГПУ (ее размер 64 Кб), достаточной для хранения таких данных.

Все расчеты производились с одинарной точностью, так как тесты на центральном процессорном устройстве (ЦПУ) показали достаточность такой точности для данного класса задач. Расчеты с двойной точностью возможны на последних моделях ГПУ от NVidia, таких как NVidia GTX285, но скорость вычислений значительно меньше по сравнению с одинарной точностью на ГПУ.

3.1. Уравнение переноса. Решение уравнение переноса разбивается на два этапа — расчет граничных условий и расчет значений на новом слое на основе формулы (2). Граничные условия представляют собой зеркальное или диффузное отражение от стенок расчетной области и записываются в фиктивные ячейки, описанные ранее. Рассмотрим подготовку диффузных граничных условий. Благодаря использованию оптимизированной функции `getIdx` вся работа с памятью происходит синхронизованно. Каждый процесс сначала суммирует поток в цикле по всем скоростным ячейкам, а потом после нормировки записывает значения в фиктивную ячейку. Для расчета используется следующий код, выполняемый на ГПУ

(приведен пример для диффузного отражения от нижней горизонтальной стенки):

```
__global__ void fillDiffuseKernel(float *data){
  int y = threadIdx.x;
  float flux = 0.0f;
  for(int i = 0; i <NV; i++) {
    int vx = speedMapX[i];
    float speedX = getSpeed(vx,NV, VMAX);
    if(speedX < 0.0f ) {
      int index = getIdx(0, y, i);
      flux += - speedX * data[index];
    }
  }
  flux = flux / NORM;
  for(int i = 0; i <NV; i++) {
    int vx = speedMapX[i];
    int vy = speedMapY[i];
    int vz = speedMapZ[i];
    float speedX = getSpeed(vx,NV, VMAX);
    if(speedX > 0.0f) {
      int index = getIdx(XMAX, y, i);
      data[index] = flux * Maxwellian(vx, vy, vz, NV, VMAX, T);
    }
  }
}
```

После этого запускается расчет значений на новом временно́м слое при помощи следующего кода:

```
 __global__ void advectionKernel(float* data, float *dataNew) {
  int x = threadIdx.x + blockDim.x * (blockIdx.x / SY);
  int y = blockIdx.x;
  int v = blockIdx.y;
  float vx = getSpeed(speedMapX[v], NV, VMAX);
  float vy = getSpeed(speedMapY[v], NV, VMAX);
  int index = getIdx(x, y, v);
  float fX = data[getIdx(x + sign(vx), y, v)];
  float fY = data[getIdx(x, y + sign(vy), v)];
  float f = data[index];
  float newF = f + abs(vx) * DT / H * (fX - f) + abs(vy) * DT / H * (fY - f);
  dataNew[index] = newF;
}
```

3.2. Интеграл столкновений. Подготовка интегрирующей сетки проводилась на ЦПУ. Для этого использовался код из солвера NSOLVER, описанного в работе [14]. При расчете интеграла вычислительному ядру передается параметр `randomSet`. Этот параметр определяет, какую сетку из заранее приготовленного набора использовать для расчета на данном шаге. Набор выбирается случайным на каждом шаге. Все процессы работают с одинаковой интегрирующей сеткой, поэтому для ускорения расчета доступ к ней происходит через буфер `grid` в разделяемой локальной памяти. Размер буфера принимается равным размеру блока процессов, т.е. `XMAX`. Это позволяет выполнять заполнение буфера по схеме: процесс номер i заполняет ячейку номер i. Такая схема позволяет осуществлять согласованный доступ к памяти. Ниже приводится код вычислительного ядра для интеграла столкновений:

```
__global__ void collisionKernel(float *data, int randomSet) {
  __shared__ KorobovGrid grid[XMAX];
  int xIndex = threadIdx.x + blockDim.x * blockIdx.x;
  int yIndex = blockIdx.y;
  for(int gridPart = 0; (gridPart + 1) * XMAX < gridSize; gridPart++){
    grid[threadIdx.x] = GridsArray[randomSet][XMAX * gridPart + threadIdx.x];
    __syncthreads();
```

```
    for(int i = 0; i < XMAX i++) {
      float fB1 = data[getIdx(x, y, grid[i].before1)];
      float fB2 = data[getIdx(x, y, grid[i].before2)];
      float fAI1 = data[getIdx(x, y, grid[i].afterI1)];
      float fAI2 = data[getIdx(x, y, grid[i].afterI2)];
      float fAE1 = data[getIdx(x, y, grid[i].afterE1)];
      float fAE2 = data[getIdx(x, y, grid[i].afterE2)];
      float r = grid[i].r;
      float delta = grid[i].gb * B * (fB1 * fB2 - pow(fAI1 * fAI2, 1.0 - r) *
pow(fAE1 * fAE2, r));
      data[getIdx(x, y, grid[i].before1)] = abs(fB1 - delta);
      data[getIdx(x, y, grid[i].before2)] = abs(fB2 - delta);
      data[getIdx(x, y, grid[i].afterI1)] = abs(fAI1 + delta * (1.0f - r));
      data[getIdx(x, y, grid[i].afterI2)] = abs(fAI2 + delta * (1.0f - r));
      data[getIdx(x, y, grid[i].afterE1)] = abs(fAE1 + delta * r);
      data[getIdx(x, y, grid[i].afterE2)] = abs(fAE2 + delta * r);
    }
  }
}
```

3.3. Сохранение результатов. Так как все данные находятся в памяти ГПУ, а скорость обмена данными между ЦПУ и ГПУ может быть узким местом, то расчет макропараметров также происходит на ГПУ. В итоге копирование производится для нескольких макропараметров, что уменьшает объем данных, копируемых из памяти ГПУ, на несколько порядков. Для вычисления макропараметров в памяти ГПУ используется следующий код:

```
__global__ void calcMacroKernel(float* data, float *results) {
  int x = threadIdx.x + blockDim.x * blockIdx.x;
  int y = blockIdx.y;
  float densTmp = 0.0f;
  float tempTmp = 0.0f;
  float flowTmp[3] = {0.0f, 0.0f, 0.0f};
  float speed[3] = {0.0f, 0.0f, 0.0f};
  float mul = VMAX / NV;
  mul = mul * mul * mul;
  for(int i = 0; i < NV; i++) {
    int index = getIdx(x, y, i);
    float f = data[index];
    densTmp += f;
    flow[0] += f * getSpeed(speedMapX[i], NV, VMAX);
    flow[1] += f * getSpeed(speedMapY[i], NV, VMAX);
    flow[2] += f * getSpeed(speedMapZ[i], NV, VMAX);
  }
  speed[0] = flow[0] / densTmp;
  speed[1] = flow[1] / densTmp;
  speed[2] = flow[2] / densTmp;
  for(int i = 0; i < NV; i++) {
    int index = getDataIndex(x, y, i);
    float f = data[index];
    float square = getSquaredSpeed(speedMapX[i], speedMapY[i], speedMapZ[i], speed, NV, VMAX);
    tempTmp += res * sq;
  }
  int index = x + XMAX * y;
  dens[index] = densTmp * mul;
  temp[index] = tempTmp / densTmp / 3.0f;
  flowx[index] = flow[0] * mul;
  flowy[index] = flow[1] * mul;
}
```

Как можно видеть из приведенного кода, он состоит из трех этапов. Сначала вычисляется плотность в данной точке и скорость газа посредством суммирования вкладов в каждой точке фазового пространства. Далее во втором цикле вычисляется вклад в температуру от каждой точки (два цикла необходимы, так как во втором цикле нам требуется знать скорость газа в точке, посчитанную в первом). На последнем этапе значения записываются в специальные массивы размером `XMAX*YMAX`. Как уже отмечалось, размер этих массивов много меньше основного массива со значениями функции распределения, и их можно скопировать в основную память для последующей записи в файл следующим кодом:

```
cudaMemcpy(dens, cudaDens, XMAX * YMAX * sizeof(float), cudaMemcpyDfeviceToHost)
```

4. Примеры расчетов.

4.1. Каверна. Тестирование алгоритма производилось на задаче о каверне. Геометрия задачи изображена на рис. 1. Рассматривается газ, заключенный в двумерную квадратную область. Вследствие равномерного движения со скоростью V_w, на верхней стенке в газе индуцируется течение. Стенки каверны имеют одинаковую температуру T_0, такую же как и газ в начальный момент времени. Газ рассеивается на стенках диффузно, на верхней стенке молекулы приобретают скорость V_w, функция распределения отраженных молекул имеет вид

$$f(\boldsymbol{x},\boldsymbol{\xi}) = \frac{N_f}{N_{\text{norm}}}\left(\frac{m}{2\pi kT}\right)^{3/2} e^{-m(\boldsymbol{\xi}-V_w)^2/(2kT)}.$$

В этой формуле значение N_f равно потоку на стенку $\int\limits_{\xi_y>0} \xi_y f\, d\xi$, а величина N_{norm} определяется из условия равенства потоков на стенку и от нее.

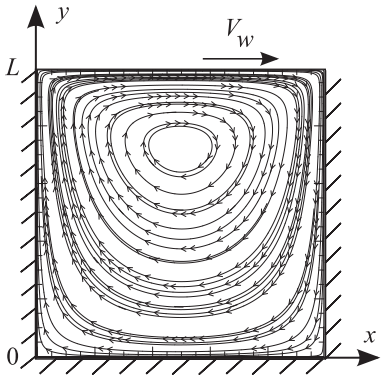

Рис. 1. Геометрия задачи о газе в каверне и линии тока в установившемся режиме

В работах [10, 15, 16] проведен расчет этой задачи для модельного уравнения для $V_w = 0.01\sqrt{\dfrac{2kT_0}{m}}$. Нами проведено решение уравнения Больцмана при $V_w = 0.003\sqrt{\dfrac{2kT_0}{m}}$ на двумерной пространственной сетке с 160×160 узлами и на трехмерной скоростной сетке с 20 узлами по каждому направлению. Размер L каверны равен 10λ, где $\lambda = \left(\sqrt{2}\pi n_0\sigma^2\right)^{-1}$ — длина свободного пробега, что дает число Кнудсена, равное 0.1 . При расчете этапа релаксации размер сетки Коробова был равен 50 000 узлов ($\approx$ 11 200 эффективных узлов, лежащих внутри сферы, ограничивающей пространство скоростей), что обеспечивало достаточную точность. Один шаг по времени был равен $\tau = 0.0186\tau_0$, где $\tau_0 = \lambda\left(\sqrt{\dfrac{2kT_0}{m}}\right)^{-1}$ — среднее время свободного пробега. На ГПУ NVidia GTX285 время расчета составило 40 минут, в то время как аналогичный расчет на одном ядре ЦПУ Intel Core Quad 2.66GHz занимал около 2.5 дней; общее ускорение составило 135 раз.

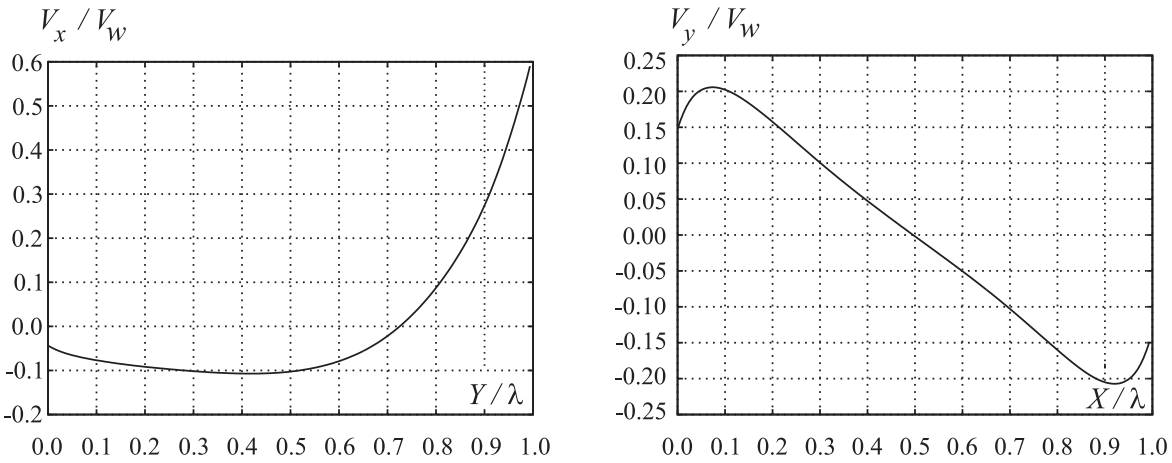

Рис. 2. Профили скорости

На рис. 1 показаны линии тока в установившемся режиме. Координаты нормированы на длину каверны L. Результаты расчета представлены также на рис. 2 профилями скорости ξ_x в вертикальном сечении, проходящем через середину каверны, и профилями скорости ξ_y в горизонтальном сечении, проходящем через центр вихря. Отметим отсутствие статистического шума на рисунке линий тока и графиках.

4.2. Анализ производительности. Ускорение счета определялось отдельно для уравнения переноса и интеграла столкновений. Большее ускорение было получено для уравнения переноса. Это объясняется большей загрузкой ГПУ, так как независимость решений для разных скоростей позволяло запускать для каждой скорости отдельный поток, и планировщик потоков на ГПУ имел больший набор потоков. Однако и для интеграла столкновений был получен значительный выигрыш во времени. В качестве вычислительного кода для ЦПУ применялся параллельный солвер NSOLVER [14], использованный ранее для расчета ряда задач [17, 18].

Таблица 1

Сравнение ускорения для разных размеров сеток на ГПУ Nvidia GTX 285

Сетка	64×64	96×96	128×128	160×160
Ускорение уравнения переноса	210	347	395	422
Ускорение уравнения релаксации	32.5	35.6	39.5	33
Суммарное ускорение	105	139	150	135

Таблица 2

Сравнение производительности на ЦПУ и различных ГПУ в задаче о каверне с сеткой 128×128

Устройство	ЦПУ (2.66 GHz)	9600GT	GTS250	GTX285
Количество ядер	1	64	128	240
Уравнение переноса (мин.)	1469.14	10.436	6.21	3.81
Уравнение релаксации (мин.)	437.43	45.61	20.1	11.47
Общее время (мин.)	1906.57	56.05	26.32	15.28
Доля этапа релаксации	23%	81%	76%	75%
Ускорение уравнения переноса	1	140.8	246.6	385
Ускорение уравнения релаксации	1	9.8	21.8	38.1
Суммарное ускорение	1	34	72.4	124.8

Код на ЦПУ запускался в виде одного процесса на одном ядре процессора Intel Core2 Quad 2.66 GHz. Основные расчеты на ГПУ проводились на ГПУ NVidia GTX285. В табл. 1 приведены значения ускорения для разных размеров сеток. Как можно видеть, увеличение сетки дает прирост скорости для уравнения переноса, однако на ускорение интеграла столкновений это влияет мало. Из таблицы следует, что расчет интеграла на ЦПУ занимает меньшее время, чем расчет уравнения переноса. Это объясняется эффективностью алгоритма, а именно высокой точностью при малом отклонении от равновесного решения (как было показано в формуле (5)).

Кроме того, проводилось сравнение работы кода на разных ГПУ: NVidia 9600GT, NVidia GTS 250 и NVidia GTX 285. Для тестов использовалась описанная выше задача с размером координатной сетки 128×128 и $N_0 = 18$. В табл. 2 приведены сводные данные о количестве ядер для перечисленных ГПУ, а также о полученном ускорении на них. Из таблицы видно, что с увеличением числа процессоров на ГПУ производительность возрастает. Это касается как интеграла, так и уравнения переноса.

Приведенные результаты можно сравнить со значениями, представленными в работе [10], в которой описывается решение модельного уравнения BGKW на ГПУ. Авторы получили сравнимое время расчета (7 минут для 4100 временны́х шагов) для решения задачи каверны в случае $V_w = 0.01\sqrt{\dfrac{2kT_0}{m}}$ и двумерной сетки размером 160×160 узлов.

При оптимизации кода использовалась программа CUDA Visual Profiler, входящая в комплект разработчика NVidia CUDA. С помощью этой программы была также оценена пропускная способность при работе с глобальной памятью, которая составила 40 Гб/с для расчета уравнения переноса и этапа релаксации и 50 Гб/с при вычислении макропараметров. Отметим, что для ГПУ NVidia GTX 285 пиковая

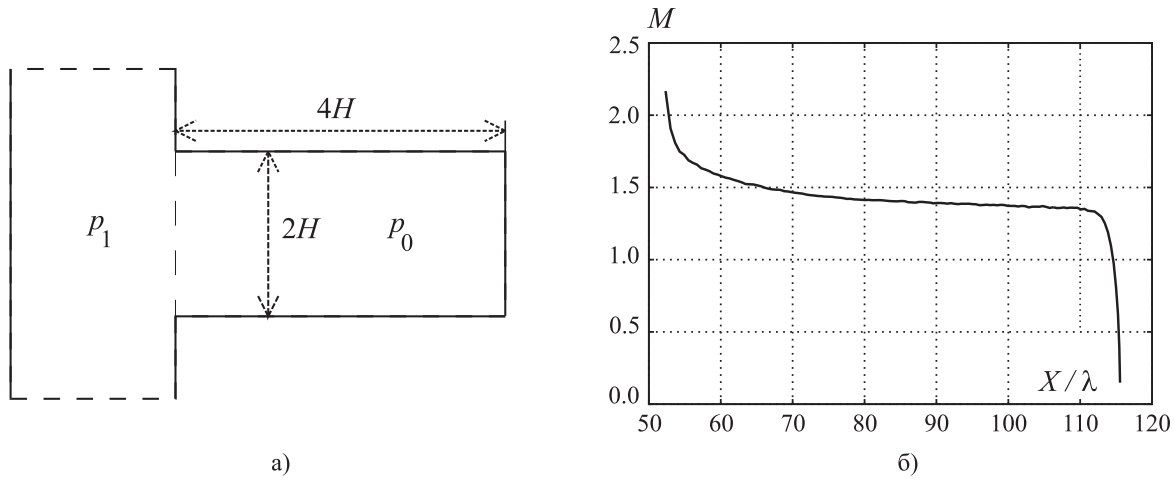

Рис. 3. Геометрия задачи УВ (а), распад УВ (б)

пропускная способность памяти составляет 159 Гб/с. Загрузка вычислительных ядер была приблизительно 100% для уравнения переноса и 88% для интеграла столкновений. Меньшая загрузка на этапе релаксации объясняется затратами на работу с разделяемой памятью, что оправдано, так как сокращает число обращений к глобальной памяти.

4.3. Ударная волна. Кроме того, алгоритм тестировался на задаче о распространении ударной волны (УВ) в узком канале, которая демонстрирует его применимость для нестационарных и сверхзвуковых течений. На рис. 3а показана геометрия ударной трубы, состоящей из широкой толкающей секции (секция A) и узкой измерительной секции (секция B). В начальный момент времени давление в толкающей секции в 10 раз больше, чем в измерительной, а температуры в обеих секциях равны. Распад начального разрыва давления создает ударную волну, которая движется внутри измерительной секции. Теоретическое значение числа Маха ударной волны, вычисленное по соотношениям Рэнкино–Гюгонио на основе одномерной теории распада произвольного разрыва, дает $M = 1.55$. Эта задача детально изучена в работе [19], в которой проведено моделирование прямой и обратной УВ на кластере МФТИ-60.

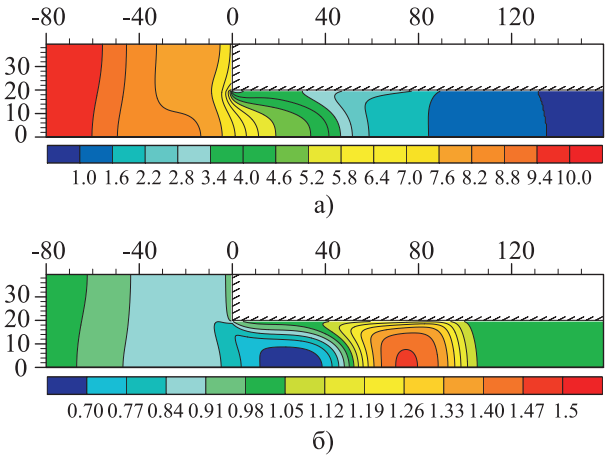

Рис. 4. Поля плотности и температуры для момента времени $t = 50$

В данной работе счетная область разделялась на два домена, содержащих секции A и B. Для каждого домена проводился расчет отдельно, а на этапе подготовки граничных условий в фиктивные ячейки копировались данные из смежных ячеек соседнего домена. Такой подход позволяет масштабировать решение задачи на несколько ГПУ. Размер домена A в расчете был равен 160×80, размер домена B был равен 320×40, шаг по координате $h = 0.5\lambda$, параметры пространства скоростей были следующие: $N_0 = 18$, $V_{\max} = 6.8\sqrt{\dfrac{kT_0}{m}}$, интегрирующая сетка состояла из 2×10^5 точек Коробова, шаг по времени был равен $\tau = 0.0111\tau_0$. При моделировании использовался потенциал взаимодействия Леннарда–Джонса с параметрами для аргона при 300 K. На рис. 4 показаны поля плотности и температуры в момент времени $t = 50\tau_0$, после расчета 4500 временны́х шагов. Расчет 4500 шагов занял на ГПУ 94 минуты, аналогичный расчет на кластере MIPT-60 с использованием 15 узлов (4 ядра с частотой 3.00 GHz) занял более 7 часов. Таким образом, получено ускорение в 74 раза. На рис. 3б показан график затухания скорости УВ (выраженной числом Маха) при распространении вдоль канала.

5. Заключение. Использование ГПУ позволяет значительно ускорить решение задач динамики разреженного газа на основе уравнения Больцмана. Несмотря на очевидные ограничения по памяти, расчеты на ГПУ позволяют получать данные с достаточной точностью для плоских задач как для медленных течений, так и для быстрых нестационарных. Описанный метод разделения расчетной области на прямоуголь-

ные домены позволяет масштабировать солвер на несколько узлов, содержащих ГПУ, при этом отношение размера граничной области к размеру домена (порядка объема памяти) будет мало́ и затраты на пересылку граничных условий должны быть небольшими. Значительное ускорение в решении уравнения переноса по сравнению с интегралом столкновения делает возможным применение схем более высокого порядка аппроксимации без потери производительности.

СПИСОК ЛИТЕРАТУРЫ

1. *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. **94**. 211–225.
2. *Bird G.A.* Molecular gas dynamics and the direct simulation of gas flows. Oxford: Clarendon Press, 1994.
3. *Черемисин Ф.Г.* Консервативный метод вычисления интеграла столкновений Больцмана // Докл. РАН. 1997. **357**, № 1. 53–56.
4. *Tcheremissine F.G.* Direct numerical solution of the Boltzmann equation // Proc. 24-th Intern. Symp. in Rarefied Gas Dynamics. New York: AIP Conf. Proc. 2005. 667–685.
5. *Черемисин Ф.Г.* Решение кинетического уравнения Больцмана для высокоскоростных течений // Журн. вычисл. матем. и матем. физики. 2006. **46**, № 2. 329–343.
6. *Tcheremissine F.G.* Solution of the Boltzmann kinetic equation for low speed flows // Transport Theory and Statistical Physics. 2008. **37**, N 5. 564–575.
7. *Anderson J.A., Lorenz C.D., Travesset A.* General purpose molecular dynamics simulations fully implemented on graphics processing units // J. Comput. Phys. 2008. **227**, N 10. 5342–5359.
8. *Molnár F.Jr., Szakály T., Mészáros R., Lagzi I.* Air pollution modelling using a graphics processing unit with CUDA // Computer Physics Communications. 2010. **180**, N 12. 105–112.
9. *Brandvik T., Pullan G.* Acceleration of a 3D Euler solver using commodity graphics hardware // 46th AIAA Aerospace Sciences Meeting and Exhibit. Reno (Nevada, USA), 2008.
10. *Frezzotti A., Ghiroldi G.P., Gibelli L.* Solving kinetic equations on GPUs. I: Model kinetic equations. 2009. arXiv : 0903.4044v1 [physics.comp-ph].
11. *Corrigan A., Camelli F., Löhner R., Wallin J.* Running unstructured grid CFD solvers on modern graphics hardware // 19th AIAA Computational Fluid Dynamics Conference. No. AIAA 2009-4001, 2009.
12. *Коробов Н.М.* Тригонометрические суммы и их приложения. М.: Наука, 1989.
13. NVIDIA Corporation, NVIDIA CUDA Programming Guide, Version 2.3.1. 2009. URL: http://www.nvidia.com/object/cuda_develop.html.
14. *Клосс Ю.Ю., Черемисин Ф.Г., Хохлов Н.И., Шурыгин Б.А.* Программно-моделирующая среда для исследования течей газа в микро- и наноструктурах на основе решения уравнения Больцмана // Атомная энергия. 2008. **105**, № 4. 211–217.
15. *Valougeorgis D., Vauritis S., Sharipov F.* Application of the integro-moment method to steady-state two-dimensional rerafeied gas flows subject to boundary induced discontinuities // J. Comput. Phys. 2008. **227**. 6272–6287.
16. *Naris S., Valougeorgis D.* The driven cavity flow over the whole range of the Knudsen number // Phys. Fluids. 2005. **17**, N 9. 907106.1–907106.12.
17. *Khokhlov N.I., Kloss Yu.Yu., Shurygin B.A., Tcheremissine F.G.* Simulation of the temperature driven micro pump by solving the Boltzmann equation // 26-th Int. Symp. on Rarefied Gas Dynamics. Books of abstract. Kyoto: Kyoto Univ. Press, 2008.
18. *Khokhlov N.I., Kloss Yu.Yu., Shurygin B.A., Tcheremissine F.G.* Application of MPI-technology for solving the Boltzmann equation // 20-th Int. Conf. on Transport Theory. Book of Abstracts. Obninsk, 2007. 189.
19. *Клосс Ю.Ю., Черемисин Ф.Г., Шувалов П.В.* Решение уравнения Больцмана для нестационарных течений с ударными волнами в узких каналах // Журн. вычисл. матем. и матем. физики. 2010. **50**, № 6. 1–15.

Поступила в редакцию
22.03.2010