УДК 533.6

РЕАЛИЗАЦИЯ МЕТОДА РЕШЕТОЧНЫХ УРАВНЕНИЙ БОЛЬЦМАНА БЕЗ ХРАНИМЫХ ЗНАЧЕНИЙ ФУНКЦИЙ РАСПРЕДЕЛЕНИЯ ДЛЯ GPU

Д. А. Бикулов[1], **Д. С. Сенин**[1]

Рассмотрен новый алгоритм для метода решеточных уравнений Больцмана, оптимизированный для работы на GPU. Алгоритм позволяет существенно сократить объем требуемой для работы DRAM в двухмерном и трехмерном случаях. Предложена теоретическая оценка расхода памяти. Проведен численный эксперимент, подтверждающий теоретическую оценку.

Ключевые слова: вычислительная гидродинамика, высокопроизводительные вычисления, графические ускорители.

1. Введение. Метод решеточных уравнений Больцмана (РМБ) — это метод численного моделирования задач переноса, который основан на решении кинетического уравнения Больцмана и широко применяется для расчетов турбулентных течений [4, 8], многокомпонентных и многофазных течений [3, 5, 6], течений с подвижными границами [9, 13] и др.

Одним из основных преимуществ метода является свойственная алгоритму локализация расчетов, что приводит к эффективному распараллеливанию задачи с использованием массивно-параллельных архитектур на базе GPGPU (General Purpose Graphics Processing Units) [18]. Однако такой вид систем отличается значительно меньшим доступным объемом DRAM (Dynamic Random Access Memory; объем составляет обычно десятки Гб) по сравнению с RAM (Random Access Memory), доступной CPU (Central Processing Unit; объем составляет обычно сотни Гб). Следовательно, возникает проблема оптимизации алгоритма по объему необходимой памяти DRAM при решении задачи с фиксированным размером расчетной области. Возможный подход к оптимизации алгоритма по объему этой памяти лежит в области выбора значений параметров расчета.

Предложенный в настоящей статье подход позволяет существенно сократить объем хранимых данных в DRAM-памяти на каждой итерации за счет расчета значений функций распределения “на лету” с использованием сохраненных значений локальных макропараметров, таких как локальная плотность и скорость среды. Данный подход применим для метода РМБ с интегралом столкновений в форме, предложенной в [7] и являющейся одной из самых распространенных в силу простоты понимания и легкости реализации.

Оптимизация по объему необходимой DRAM-памяти происходит за счет фиксации значения параметра релаксации $\tau = 1$. Это возможно в силу устойчивости такого вида алгоритма только в небольшой области этого значения [1, 2]. Если же существует необходимость проводить вычисления при малых значениях параметра релаксации, то обычно используют многорелаксационный алгоритм (Multiple Relaxation Time), который обладает значительно более широкой областью устойчивости; в этом алгоритме изменение значения релаксационных параметров уже существенно влияет на время расчета.

2. Метод решеточных уравнений Больцмана. Метод решеточных уравнений Больцмана основан на методе решеточного газа [12]. В [17] показано, что метод РМБ с помощью процедуры Чапмена–Энскога сводится к решению уравнений Навье–Стокса для сжимаемой жидкости. В методе решеточных уравнений Больцмана состояние системы описывается через функции распределения $f_i(\boldsymbol{r}, t)$.

С помощью этих функций распределения можно рассчитать локальные плотность ρ и скорость $\boldsymbol{u}$ среды: $\rho(\boldsymbol{r}, t) = \sum_i f_i(\boldsymbol{r}, t)$ и $\boldsymbol{u}(\boldsymbol{r}, t)\rho(\boldsymbol{r}, t) = \sum_i \boldsymbol{c}_i f_i(\boldsymbol{r}, t)$.

Уравнение эволюции системы через функции распределения можно представить в следующем виде [10]: $f_i(\boldsymbol{r} + \boldsymbol{c}_i\delta t, t + \delta t) - f_i(\boldsymbol{r}, t) = \Omega_i(\boldsymbol{r}, t)$, где $\Omega_i(\boldsymbol{r}, t)$ —интеграл столкновений для i-го направления скорости. Далее будем рассматривать однорелаксационный метод решеточных уравнений Больцмана с

[1] Московский государственный университет им. М. В. Ломоносова, физический факультет, Ленинские горы, 119992, Москва, Россия; Д. А. Бикулов, аспирант, e-mail: bikulov@physics.msu.ru; Д. С. Сенин, аспирант, e-mail: senindmitry@gmail.com

шаблоном скоростей D2Q9 (рис. 1).

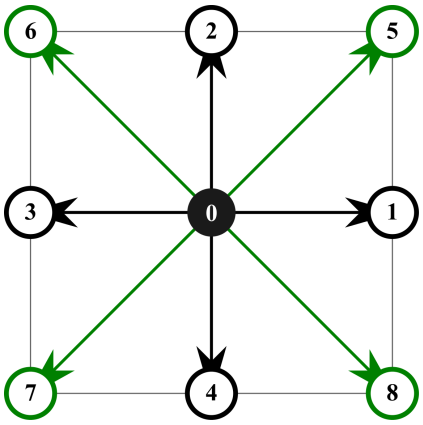

Рис. 1. Шаблон D2Q9 для метода решеточных уравнений Больцмана

Для шаблона D2Q9 коэффициенты $\boldsymbol{c}_i$ имеют вид

$$\boldsymbol{c}_i = \begin{cases} (0,0), & i = 0, \\ \left(\cos\left(\dfrac{(i-1)\pi}{2}\right), \sin\left(\dfrac{(i-1)\pi}{2}\right)\right)c, & i = 1,\ldots,4, \\ \left(\cos\left(\dfrac{(2i-9)\pi}{4}\right), \sin\left(\dfrac{(2i-9)\pi}{4}\right)\right)\sqrt{2}c, & i = 5,\ldots,8, \end{cases}$$

где $c = \dfrac{\delta x}{\delta t}$, δx и δt — шаги пространственной и временно́й дискретизации соответственно. Обычно выбирают $\delta x = \delta t$, т.е. $c = 1$.

3. Технология CUDA. Технология CUDA (Compute Unified Device Architecture) позволяет осуществлять вычисления общего назначения на графических процессорах (GPU). В последнее время все больше мировых суперкомпьютеров оснащаются GPU в качестве сопроцессоров к CPU. Это, с одной стороны, увеличивает их общую производительность, а с другой — увеличивает их энергоэффективность. Суперкомпьютер “Ломоносов”, установленный в НИВЦ МГУ, содержит более тысячи GPU Tesla X2070 с 6 Гб DRAM каждая.

Технология CUDA реализует модель SIMT (Single Instruction Multiple Thread), т.е. рассчитана на выполнение одной и той же операции множеством легковесных нитей. Последние являются аналогами потоков на CPU. Нити объединены в “блоки”, блоки объединены в “грид”. Эта иерархия условна и необходима для организации вычислений на GPU. Важно, что наиболее эффективно на графические ускорители ложатся массивно-параллельные алгоритмы, действия в которых могут выполняться независимо сразу в сотнях тысяч потоков.

Вычислительные ядра на GPU объединены в мультипроцессоры SMX, каждый из которых также содержит планировщики заданий, кэш и т.д. Нити одного блока могут выполняться только на одном процессоре SMX, который может выполнять нити сразу нескольких блоков. Количество блоков на одном таком процессоре зависит от ресурсов, требуемых для расчетов блока. Функции, выполняющиеся на GPU и имеющие спецификатор __global__, называются ядрами.

Метод решеточных уравнений Больцмана является высокопараллелизируемым алгоритмом: вычисления могут быть организованы так, что в каждой нити на каждом шаге расчеты выполняются независимо. В разделе 4 рассмотрены два распространенных алгоритма расчетов с помощью метода РМБ на GPU. Кроме того, предложен новый алгоритм, позволяющий избавиться от необходимости хранения значений всех функций распределения в глобальной памяти графического ускорителя и вычислять их “на лету”.

4. Реализация для фиксированного параметра релаксации $\tau = 1$. В случае приближения Бхатнагара–Гросса–Крука для интеграла столкновений решеточное уравнение Больцмана принимает вид

$$f_i(\boldsymbol{r} + \boldsymbol{c}_i\delta_t, t + \delta_t) = f_i(\boldsymbol{r}, t) + \frac{1}{\tau}\left(f_i^{\mathrm{eq}}(\rho, \boldsymbol{u}) - f_i(\boldsymbol{r}, t)\right). \tag{1}$$

Если параметр релаксации $\tau = 1$, то уравнение (1) упрощается:

$$f_i(\boldsymbol{r} + \boldsymbol{c}_i\delta_t, t + \delta_t) = f_i^{\mathrm{eq}}(\rho, \boldsymbol{u}), \tag{2}$$

В классическом “наивном” алгоритме [15, 16] РМБН (РМБ Наивный) для CUDA в памяти видеокарты хранятся два массива функций распределения: основной и вспомогательный. Вспомогательный массив нужен для обеспечения независимости вычислений в каждой ячейке объема. Шаги столкновения и распространения производятся в одном ядре. Каждая нить считывает из основного массива в глобальной памяти значения функций распределения для ячейки, производит расчет интеграла столкновений и записывает полученные новые значения f_i с учетом распространения во вспомогательный массив. После завершения операций во всех нитях происходит обмен указателей, и старый вспомогательный массив становится новым основным, старый основной — новым вспомогательным.

Обменный алгоритм [10, 11] РМБО (РМБ Обменный) так же является параллельным, при этом используется только один массив функций распределения. Этап столкновения (“collision”) и этап распространения (“streaming” или “propagation”) выполняются в разных ядрах, между ними требуется барьерная синхронизация. На этапе столкновения происходит считывание значений функций распределения для ячейки из массива в глобальной памяти. Производится вычисление интеграла столкновений $\Omega(f_i(\boldsymbol{r},t),\rho,\boldsymbol{u}) = f_i(\boldsymbol{r},t) + \dfrac{1}{\tau}\,(f_i^{\mathrm{eq}}(\rho,\boldsymbol{u}) - f_i(\boldsymbol{r},t))$, измененные значения $\widehat{f_i}$ записываются в ту же область массива в глобальной памяти в обратном порядке (f_1 на место f_3, f_2 на место f_4 и т.д.). После того как интеграл $\Omega(\cdot)$ вычислен для всех ячеек, производится параллельный независимый попарный обмен значений функций распределения между ячейками так, чтобы после выполнения всех операций значения f_i были записаны с учетом шага распространения [10].

Расходуемый объем памяти в обоих случаях достаточно велик. В “наивном” алгоритме при использовании чисел с плавающей точкой одинарной точности объем составляет $\mathcal{M}_n = N^2 \times (4 \times 2 \times Q + 1)$, где N — длина стороны в случае квадратной расчетной области (здесь учтен массив, хранящий тип ячейки: однобайтовое целое) и Q — количество направлений скоростей в шаблоне (в случае D2Q9 это 9). В обменном алгоритме $\mathcal{M}_s = N^2 \times (4 \times Q + 1)$.

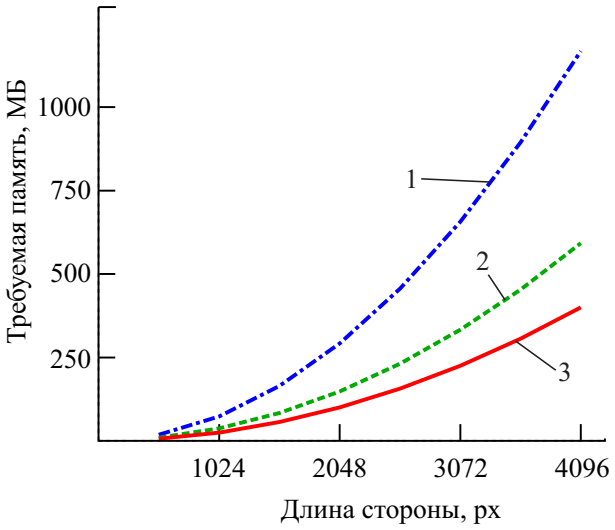

Рис. 2. Теоретическая зависимость количества потребляемой памяти от длины стороны квадратной расчетной области в D2Q9 в случае чисел с плавающей точкой одинарной точности: 1) РМБН, 2) РМБО, 3) РМБПС

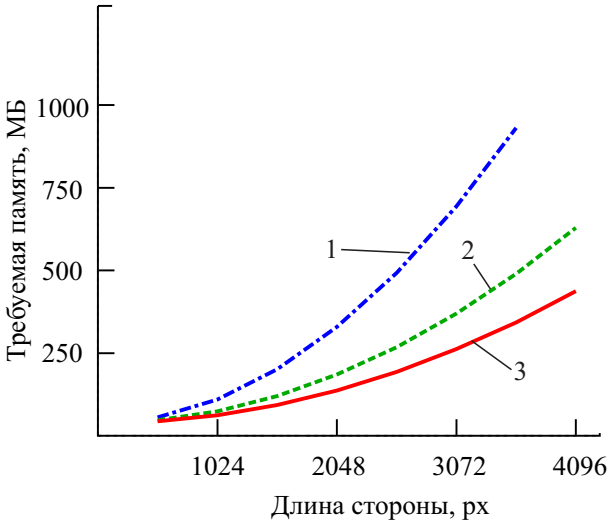

Рис. 3. Измеренная в ходе численного эксперимента зависимость количества потребляемой памяти от длины стороны квадратной расчетной области в D2Q9 в случае чисел с плавающей точкой одинарной точности: 1) РМБН, 2) РМБО, 3) РМБПС

Мы предлагаем алгоритм РМБПС (РМБ Плотность–Скорость), в котором $\tau = 1$. Из (2) следует, что для расчетов функций распределения на новой итерации в этом случае не надо знать значения f_i непосредственно, достаточно локальных значений плотности ρ и скорости $\boldsymbol{u}$ в ячейке. Требуемые для расчетов значения f_i вычисляются “на лету”. Шаги столкновения и распространения производятся в одном ядре. Каждая нить считывает из основных массивов в глобальной памяти значения скорости и плотности для каждого направления из ячеек с учетом распространения (“тянущая” схема). Когда получены f_i для всех направлений, производится расчет новых локальных плотности и скорости, которые записываются во вспомогательные массивы скорости и плотности в глобальной памяти. Когда описанные действия выполнены для всех ячеек, происходит обмен указателей, и основные массивы меняются местами со вспомогательными. Расход памяти составляет $\mathcal{M}_{RU} = N^2 \times (4 \times 2 \times (D + 1) + 1)$, где D — размерность задачи. Потребление памяти в данном алгоритме иллюстрируется на рис. 2.

5. Результаты. Рассмотренные алгоритмы реализованы на C++ CUDA. Используются числа с плавающей точкой одинарной точности. В качестве GPU используется GTX 560Ti с 1 Гб DRAM памяти. Произведено сравнение количества потребляемой памяти (рис. 3) и скорости работы алгоритмов (рис. 4) для разных размеров двумерной квадратной расчетной области. На границах расчетной области использовались граничные условия с заданным давлением [14]. На границах непротекаемых тел выбрано краевое условие "нарешеточный отскок" (on-grid bounce-back) [19]. В начальный момент времени во всех протекаемых и примыкающих к ним непротекаемых (для реализации on-grid bounce-back краевого условия) ячейках были выбраны значения $f_i(\boldsymbol{r}, t=0) = f_i^{\mathrm{eq}}\big(\rho = 1, \boldsymbol{u} = (0,0)\big)$. Расчетная область задавалась в соответствии с правилом: если $(30 < y_i < N_y - 30)$ или $(x < 5)$ или $(x > N_x - 5)$, то через ячейку возможно течение; иначе ячейка является непроницаемой. Здесь x_i и y_i — координаты центра i-й ячейки, N_x и N_y — размеры расчетной области.

На рис. 3 показаны результаты измерений объема используемой памяти различными алгоритмами в зависимости от размеров расчетной сетки. Измерение объема производилось с помощью утилиты nvidia-smi. При этом графическая оболочка операционной системы использовала вторую идентичную видеокарту и не влияла на результаты измерений.

Алгоритм РМБН не смог отработать на сетке 4096^2 из-за нехватки DRAM (произошло завершение работы с ошибкой Out of Memory). На графике можно видеть, что измеренный экспериментально объем используемой памяти всегда отличается от измеренного теоретически на 37 Мб. Это расхождение вызвано дополнительными расходами на константную память, исполняемые инструкции и т.д., которые не учитывались при теоретической оценке. РМБПС оказывается наиболее экономичным с точки зрения потребляемых ресурсов, что согласуется с проведенной выше теоретической оценкой.

Ниже приведен алгоритм РМБПС без хранимых функций распределения.

Data: $\widehat{\rho}$ и $\widehat{\boldsymbol{u}}$ на предыдущем шаге
Result: ρ и $\boldsymbol{u}$ после этапов распространения и столкновения
foreach ячейка C из решетки $N \times N$ **do**
загрузить значения $\widehat{\rho}$ и $\widehat{\boldsymbol{u}}$ из "старого" массива в DRAM в C и примыкающих к ней;
посчитать значения $\widehat{f}_i(\boldsymbol{r}) = f_i^{\mathrm{eq}}\left(\widehat{\rho}(\boldsymbol{r} - \boldsymbol{c}_i), \widehat{\boldsymbol{u}}(\boldsymbol{r} - \boldsymbol{c}_i)\right)$;
посчитать $\boldsymbol{u}(\boldsymbol{r})\rho = \sum \widehat{f}_i(\boldsymbol{r})\boldsymbol{c}_i$, $\rho = \sum \widehat{f}_i(\boldsymbol{r})$;
сохранить новые значения $\rho(\boldsymbol{r})$ и $\boldsymbol{u}(\boldsymbol{r})$ в "новый" массив в DRAM в C;
end
обмен указателей на "новый" и "старый" массивы в DRAM.

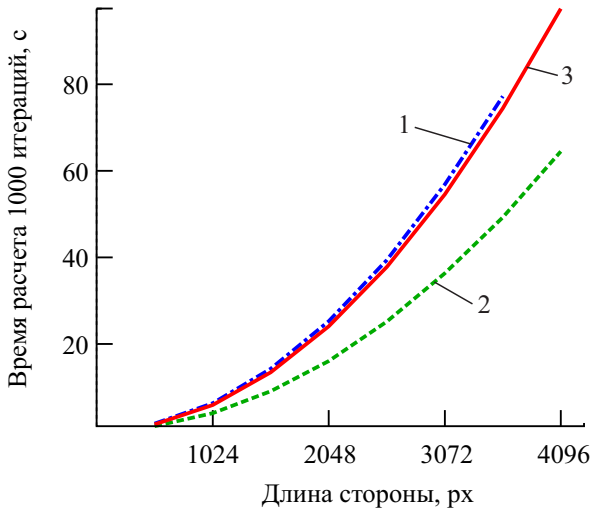

Рис. 4. Измеренная в ходе численного эксперимента зависимость времени работы реализованных алгоритмов от длины стороны квадратной расчетной области в D2Q9: 1) РМБН, 2) РМБО, 3) РМБПС

На рис. 4 показаны результаты измерений времени расчета 1000 итераций различными алгоритмами в зависимости от размеров расчетной сетки. Точность измерений составляет ± 0.05 с. Таким образом, с одной стороны, РМБПС эффективен с точки зрения потребляемой памяти (рис. 3), с другой стороны, его время работы сравнимо с временем работы РМБН.

Таким образом, алгоритм РМБПС оказывается наиболее эффективным в случае, когда требуется

проводить вычисления для большой расчетной области при ограниченных ресурсах GPU и можно зафиксировать параметр релаксации $\tau = 1$. Когда важна скорость расчетов, алгоритм РМБО оказывается наиболее эффективным.

6. Выводы. Показано, что реализация однорелаксационного алгоритма метода РМБ с фиксированным параметром релаксации для шаблона скоростей D2Q9 позволяет существенно сократить объем необходимой DRAM-памяти при решении задач с фиксированным размером расчетной области (по сравнению с алгоритмом РМБН почти в 3 раза, а по сравнению с алгоритмом РМБО — почти в 1.5 раза); изначальная вычислительная эффективность сохраняется. Для трехмерных задач с шаблоном скоростей Q3Q27 выигрыш по памяти еще существеннее: по сравнению с алгоритмом РМБН почти в 7 раз, а по сравнению с алгоритмом РМБО — почти в 3.5 раза.

Фиксация параметра релаксации не вносит существенных ограничений на время расчетного шага в прикладных задачах, так как изначально этот параметр может быть изменен в узких пределах, обусловленных устойчивостью расчета. Таким образом, мы получаем существенный выигрыш по объему необходимой памяти DRAM путем незначительных ограничений на время изменения расчетного шага.

СПИСОК ЛИТЕРАТУРЫ

1. *Кривовичев Г.В.* Об устойчивости решеточной кинетической схемы Больцмана для расчета плоских течений // Вычислительные методы и программирование. 2011. **12**. 194–204.
2. *Кривовичев Г.В.* Исследование устойчивости явных конечно-разностных решеточных кинетических схем Больцмана // Вычислительные методы и программирование. 2012. **13**. 332–340.
3. *Куперштох А.Л.* Трехмерное моделирование двухфазных систем типа жидкость–пар методом решеточных уравнений Больцмана на GPU // Вычислительные методы и программирование. 2012. **13**. 130–138.
4. *Asinari P.* Multiple-relaxation-time lattice Boltzmann scheme for homogeneous mixture flows with external force // Phys. Rev. E 2008. **77**. 1–13.
5. *Bao J., Schaefer L.* Lattice Boltzmann equation model for multi-component multi-phase flow with high density ratios // Applied Mathematical Modelling. 2012. **37**, N 4. 1860–1871.
6. *Basit R., Basit M.A.* Simulation of phase separation process using lattice Boltzmann method // Canadian J. on Computing in Mathematics, Natural Sciences, Engineering & Medicine. 2010. **1**, N 3. 71–76.
7. *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.
8. *Kaehler G., Wagner A.J.* Derivation of hydrodynamics for multi-relaxation time lattice Boltzmann using the moment approach // Computational Physics. 2013. **13**, N 3. 614–628.
9. *Kao P.-H., Yang R.-J.* An investigation into curved and moving boundary treatments in the lattice Boltzmann method // J. of Computational Physics. 2008. **227**, N 11. 5671–5690.
10. *Latt J.* How to implement your DdQq dynamics with only q variables per node (instead of 2q) // Technical Report. Medford: Tufts University, 2007.
11. *Mattila K., Hyväluoma J., Timonen J., Rossi T.* Comparison of implementations of the lattice-Boltzmann method // Computers & Mathematics with Applications. 2008. N 7. 1514–1524.
12. *McNamara G.R., Zanetti G.* Use of the Boltzmann equation to simulate lattice-gas automata // Phys. Rev. Lett. 1988. **61**. 2332–2335.
13. *Meldi M., Vergnault E., Sagaut P.* An arbitrary Lagrangian–Eulerian approach for the simulation of immersed moving solids with lattice Boltzmann method // J. of Computational Physics. 2013. **235**. 182–198.
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 & Mathematics with Applications. 2013. **65**, N 2. 252–261.
16. *Tölke 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.
17. *Viggen E.M.* The lattice Boltzmann method with applications in acoustics. Master's Thesis. Trondheim: Norwegian Univ. of Science and Technology, 2009.
18. *Xian W., Takayuki A.* Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster // Parallel Computing. 2011. N 37. 521–535.
19. *Zou Q., He X.* On pressure and velocity boundary conditions for the lattice Boltzmann BGK model // Physics of Fluids. 1997. **9**. 1591–1599.

Поступила в редакцию
14.06.2013