УДК 533.6 doi 10.26089/NumMet.v16r221

ЭФФЕКТИВНАЯ РЕАЛИЗАЦИЯ МЕТОДА РЕШЕТОЧНЫХ УРАВНЕНИЙ БОЛЬЦМАНА ДЛЯ ГИБРИДНЫХ СУПЕРКОМПЬЮТЕРНЫХ СИСТЕМ

Д. А. Бикулов[1]

Рассмотрены особенности эффективной реализации метода решеточных уравнений Больцмана (Lattice Boltzmann method, LBM) для гибридных суперкомпьютерных систем с множеством видеокарт. Описаны основные стратегии по сокращению требуемой для работы LBM памяти на графическом ускорителе. Представлены результаты измерения зависимости производительности реализованного программного модуля от числа задействованных видеокарт, полученные на суперкомпьютере "Ломоносов".

Ключевые слова: высокопроизводительные вычисления, графические ускорители, метод решеточных уравнений Больцмана, Lattice Boltzmann method, LBM, cuda, multi-gpu, масштабируемость.

Введение. Метод решеточных уравнений Больцмана (Lattice Boltzmann method, LBM) используется для моделирования многочисленных процессов в различных областях: однокомпонентных течений [4, 16] (в том числе на неструктурированных сетках [6]), развития стеноза артерий [12], течений неньютоновских жидкостей [13], течений в микроканалах [24], роста кристаллов в перенасыщенном растворе [22], течений жидкости в поровых пространствах [10, 11, 21], поведения двухфазных систем [7, 8, 23], моделирования диффузии [5, 38] и теплопереноса [20] и др. Особенностью этого метода является его универсальность: для моделирования различных процессов используются схожие алгоритмы; следовательно, применимы схожие стратегии эффективной реализации для высокопроизводительных систем.

1. Технология CUDA. Compute Unified Device Architecture (CUDA) — программно-аппаратная платформа, предоставляющая возможность использования графических ускорителей (Graphics Processing Unit, GPU), поддерживающих данную технологию, для расчетов общего назначения, в том числе для ускорения численного моделирования. Для этого при написании программ используется специальный Си-подобный синтаксис, а специализированный компилятор осуществляет трансляцию исходного кода в программу, исполняемую на GPU. Вычисления с использованием CUDA доступны на суперкомпьютере "Ломоносов" в разделе gpu [30].

Архитектура графических ускорителей рассчитана на эффективное выполнение алгоритмов, использующих SIMD-модель (Single Instruction Multiple Data model) вычислений. На GPU — низкие накладные расходы на создание потоков ("нитей"), переключение контекста происходит почти мгновенно, а количество нитей обычно достигает сотен тысяч и позволяет производить массивно-параллельные вычисления. В большинстве случаев работает простое правило: чем в большем количестве параллельных нитей работает программа, тем больше возможностей у планировщика на GPU оптимизировать порядок выполнения инструкций и тем более эффективными оказываются вычисления.

Различные поколения графических ускорителей, поддерживающих CUDA, отличаются архитектурой и вычислительными возможностями (Compute Capability, CC). На суперкомпьютере "Ломоносов" установлены вычислители Tesla X2070, имеющие архитектуру Fermi и CC 2.0. Рассмотрим архитектуру Fermi более подробно.

Графические ускорители с архитектурой Fermi (рис. 1) содержат в сумме до 512 вычислительных ядер (шейдерных процессоров), объединенных в группы по 32 элемента, называемых потоковыми мультипроцессорами (Streaming Multiprocessor, SM) (рис. 2). Каждое ядро, в свою очередь, включает в себя модули целочисленных (Arithmetic Logic Unit, ALU) и вещественнозначных расчетов (Floating-Point Unit, FPU). Помимо шейдерных процессоров в каждом SM расположены 16 банков доступа к общей памяти (Load/Store Units, LD/ST), четыре модуля расчетов специальных функций (Special Function Units, SFU), два планировщика варпов (Warp Schedulers), два модуля управления инструкциями (Dispatch Units), банк для 32 768 32-битных регистров, четыре текстурных модуля, однородный кэш (Uniform Cache), текстурный кэш (Texture Cache), геометрический движок (PolyMorph Engine) и 64 Кб памяти, распределенной между кэшем L1 и общей памятью (Shared Memory), для которой доступны два специализированных

[1] Московский государственный университет им. М. В. Ломоносова, физический факультет, Ленинские горы, 119992, Москва; аспирант, e-mail: bikulov@physics.msu.ru

режима: 48 Кб для L1-кэша и 16 Кб для общей памяти либо 16 Кб и 48 Кб соответственно. Общий для всех потоковых мультипроцессоров кэш L2 имеет размер 768 Кб.

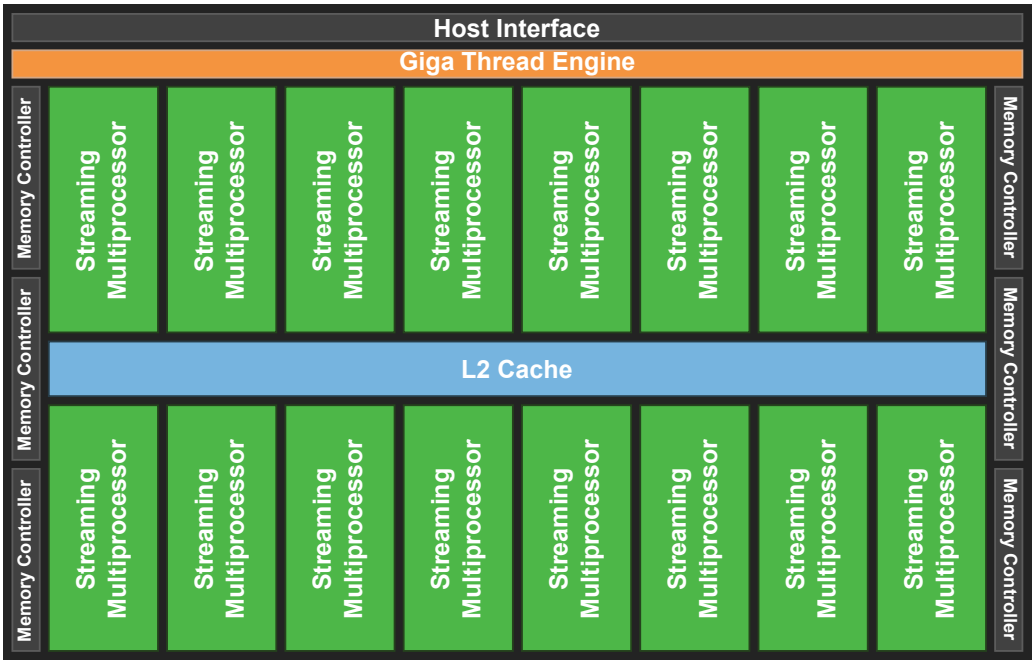


Рис. 1. Общая схема графического ускорителя. Количество потоковых мультипроцессоров варьируется в зависимости от модели ускорителя [27]

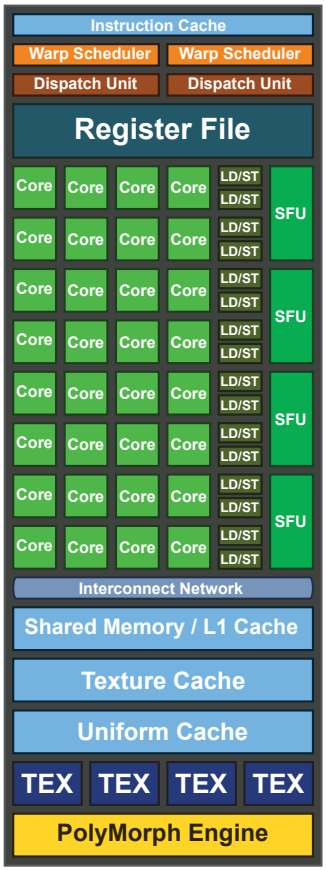


Рис. 2. Схема потокового мультипроцессора поколения Fermi [27]

В CUDA используется многоуровневая иерархия типов памяти, включающая в себя глобальную память (Global Memory), текстурную память (Texture Memory), константную память (Constant Memory), локальную память (Local Memory), общую память (Shared Memory) и регистры (Registers) (см. таблицу). Кроме того, существуют два уровня общего кэширования (L1 и L2) и два специализированных неизменяемых кэша для константных данных и текстур. Все кэши используются автоматически и не доступны для управления (единственное исключение — управление размером кэша L1).

Характеристики иерархической памяти в CUDA. Запись в типы памяти, отмеченные звездочкой, доступна только с хоста

Название	Расположение	Доступ	Область видимости
глобальная	DRAM	чтение и запись	все нити и хост
текстурная	DRAM	чтение*	все нити и хост
константная	DRAM	чтение*	все нити и хост
локальная	DRAM	чтение и запись	внутри нити
общая	на чипе	чтение и запись	все нити одного блока
регистры	на чипе	чтение и запись	внутри нити

Глобальная память — самая большая по объему и самая медленная. Физически глобальная память расположена в DRAM-памяти видеокарты, ее максимальный размер в архитектуре Fermi составляет 6 Гб. Глобальная память доступна из всех нитей сети.

Текстурная и константная памяти также физически располагаются в DRAM, но кэшируются в специализированных текстурном и константном кэшах соответственно. Особенность текстурной памяти состоит в нескольких дополнительных возможностях, полезных для работы с текстурами: кэшировании чтения с предположением о 2D-локальности данных, аппаратной фильтрации и др., но в некоторых случаях она может работать медленнее глобальной памяти. Объем константной памяти ограничен 64 Кб, скорость обращения к ней за счет кэширования на потоковых мультипроцессорах более чем на порядок превосходит скорость доступа к глобальной памяти. Текстурная память и константная память доступны из всех нитей сети.

Общая память располагается непосредственно на потоковых мультипроцессорах, доступ к ней более чем на порядок быстрее доступа к глобальной памяти, размер значительно меньше (16 Кб или 48 Кб в зависимости от режима). Часто ее используют в виде управляемого вручную кэша L1, но для многих задач достаточно автоматического кэширования. К общей памяти имеют доступ только нити одного блока.

Регистры — самая быстрая память на GPU. Память под все переменные, объявленные в ядрах (в том числе и статически созданные массивы), по умолчанию выделяется в регистровой памяти. К регистровой памяти имеет доступ только та нить, в которой она была выделена. В случае если память для переменной не может быть выделена в регистровой памяти (например, из-за ее переполнения), то переменная создается в локальной памяти. Локальная память — специальная область глобальной памяти, поэтому попадание в локальную память существенно снижает производительность. На современных GPU доступ к локальной памяти кэшируется в L1 и L2.

Современные архитектуры графических ускорителей все более ориентированы на задачи, лимитирующим фактором в которых выступает память. К таким задачам относится и моделирование процессов переноса с помощью метода решеточных уравнений Больцмана. Рассмотрим этот метод более подробно.

2. Метод решеточных уравнений Больцмана. Метод решеточных уравнений Больцмана представляет собой класс методов вычислительной гидродинамики. Его отличительными особенностями являются высокая параллельность по данным, простота реализации и широкие возможности моделирования. Например, для моделирования анизотропной диффузии [38] и теплопроводности [20] используется один и тот же алгоритм и те же основные уравнения.

В рассматриваемом методе среда представляется в виде набора квазичастиц, перемещающихся между пространственными ячейками вдоль заранее заданных дискретных направлений. Ячейки делятся на два типа: проницаемые и непроницаемые. Перемещение частиц возможно только между проницаемыми ячейками. В ячейках, граничащих с непроницаемыми, применяется локальное граничное условие. Совокупность заданных направлений квазичастиц называется шаблоном и обозначается DdQq, где d — размерность пространства (2 — для плоского случая, 3 — для трехмерного), а q — число выбранных направлений скоростей в шаблоне. Наиболее часто используются [1, 3, 16, 32] шаблоны D2Q9 и D3Q7 в плоском случае и D3Q15 и D3Q19 в трехмерном случае.

Для описания ансамбля квазичастиц используются функции распределения f_i, где i соответствует

направлению выбранного шаблона. Эволюция функций f_i задается решеточным уравнением Больцмана

$$\big|f(\boldsymbol{r}+\boldsymbol{e}\delta t, t+\delta t)\big\rangle = \big|f(\boldsymbol{r},t)\big\rangle + \Omega\big(f(\boldsymbol{r},t)\big), \tag{1}$$

где

$$\big|f(\boldsymbol{r}+\boldsymbol{e}\delta t, t+\delta t)\big\rangle = \big(f_0(\boldsymbol{r}+\boldsymbol{e}_0\delta t, t+\delta t), \ldots, f_{q-1}(\boldsymbol{r}+\boldsymbol{e}_{q-1}\delta t, t+\delta t)\big)^{\mathrm{T}}, \quad \big|f(\boldsymbol{r},t)\big\rangle = \big(f_0(\boldsymbol{r},t), \ldots, f_{q-1}(\boldsymbol{r},t)\big)^{\mathrm{T}}.$$

Здесь $\Omega\big(f(\boldsymbol{r},t)\big)$ — интеграл столкновений в общем виде, $\big|f(\boldsymbol{r}+\boldsymbol{e}\delta t, t+\delta t)\big\rangle$ — вектор функций распределения после шага распространения и $\big|f(\boldsymbol{r},t)\big\rangle$ — вектор функций распределения до шага столкновения.

Одна итерация расчета, таким образом, разделяется на два шага — столкновение и распространение:

$$\big|f^*(\boldsymbol{r},t)\big\rangle = \big|f(\boldsymbol{r},t)\big\rangle + \Omega\big(f(\boldsymbol{r},t)\big), \tag{2}$$

$$\big|f(\boldsymbol{r}+\boldsymbol{e}\delta t, t+\delta t)\big\rangle = \big|f^*(\boldsymbol{r},t)\big\rangle. \tag{3}$$

Шаг столкновения (2) описывает взаимодействие квазичастиц внутри ячейки. Он различен при моделировании различных процессов переноса, но чаще всего локален и меняет значения функций распределения только внутри текущей ячейки, поэтому может выполняться параллельно. Шаг распространения (3) заключается в обмене с перезаписью значений функции распределения между соответствующими соседними ячейками и при наивной реализации не может выполняться параллельно без использования дублирующего массива для хранения новых значений f_i, поскольку без дублирующего массива приходится учитывать порядок обхода ячеек. В случае двух массивов (будем называть их “старый” и “новый”) параллельный алгоритм расчета одной итерации выглядит следующим образом:

Алгоритм 1. Расчет одной итерации с использованием двух массивов на шаге распространения.

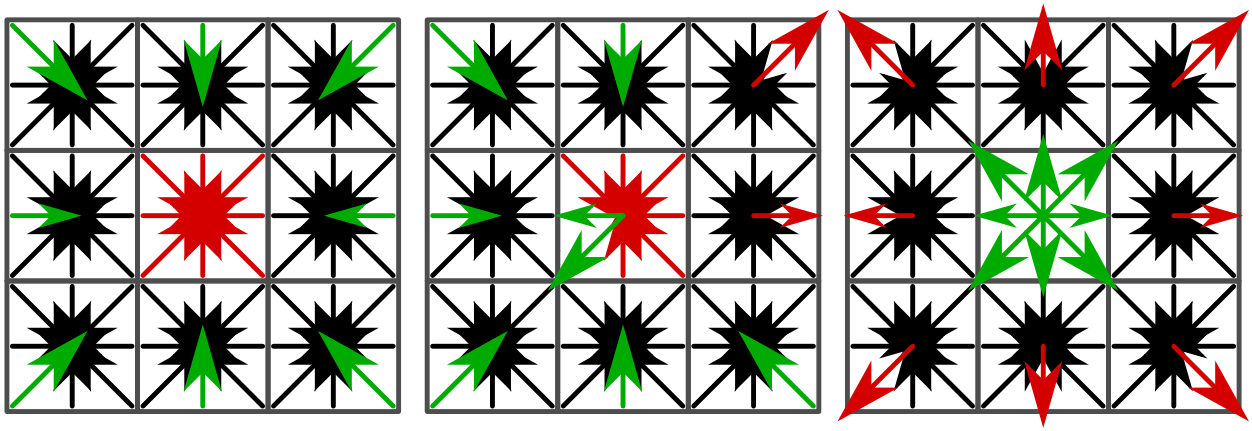
for *каждый процесс* **do**
    параллельно посчитать столкновение в каждой ячейке;
    записать измененные значения f_i с учетом шага распространения в “новый” массив;
end
барьерная синхронизация;
for *каждый процесс* **do**
    поменять местами указатели на “новый” и “старый” массивы;
    применить граничные условия;
end

В шаблоне D3Q19 используется 19 дискретных направлений скоростей, в схеме D3Q7 — семь дискретных направлений. В общем случае на каждую расчетную ячейку требуется 1 байт для хранения типа (проницаемая или непроницаемая), 76 или 28 байт для хранения значений функции распределения для шаблонов D3Q19 и D3Q7 соответственно. В случае двух компонент объем памяти, необходимый для хранения значений функции распределения, удваивается. Максимальный размер доступной DRAM-памяти на GPU сейчас достигает 24 Гб, но такие устройства еще массово не распространены. На суперкомпьютере “Ломоносов” установлены вычислители с 6 Гб, максимальный размер модели однокомпонентного течения в этом случае примерно равен 340^3 ячейкам. Этого недостаточно для решения многих реальных задач, поэтому приходится использовать различные методы оптимизации потребления памяти, а для массивно-параллельных вычислений больших объемов используется блочная декомпозиция расчетной области и расчет на множестве видеокарт.

3. Оптимизация использования памяти на графическом ускорителе. Опубликованы многочисленные работы на тему эффективной реализации различных вариантов метода решеточных уравнений Больцмана с использованием вычислений на графических ускорителях [2, 8, 9, 26], в том числе и на суперкомпьютерах с множеством GPU (Multi-GPU кластерах) [14, 15, 18, 25, 31, 33, 37]. В настоящее время совместное использование MPI и CUDA становится де-факто стандартом переносимых суперкомпьютерных научных вычислений [14, 15, 18, 25, 31, 33, 37].

Существует несколько различных способов избавления от дублирующего массива для f_i [35]. Рассмотрим обменный алгоритм, позволяющий за счет обратной записи значений f_i на шаге столкновения попарно обменивать эти значения между соседними ячейками (рис. 3) на шаге распространения. Важно, что в этом случае обмен может осуществляться параллельно и независимо для каждой пары направлений в соседних ячейках.

В обменном алгоритме после шага столкновения значения f_i записываются в обратном порядке (рис. 3а). Обратной считается запись f_m вместо f_n (и наоборот), если $\boldsymbol{e}_m = -\boldsymbol{e}_n$, где $\boldsymbol{e}_i$ — направления в выбранном шаблоне скоростей. Противоположные направления удобно обозначать отрицательными индексами: $\boldsymbol{e}_i = -\boldsymbol{e}_{-i}$. Затем, на шаге распространения, производится обмен каждой пары f_i и f_{-i} в



a) b) c)

Рис. 3. Обменный алгоритм для шага распространения: a) после столкновения; b) в процессе распространения; c) после распространения

соседних ячейках (рис. 3b). После прохождения шага распространения во всех ячейках f_i оказываются записаны в правильном порядке с учетом распространения (рис. 3c).

Все значения функций распределения хранятся в виде единого массива, общий размер которого составляет $N = N_x \times N_y \times N_z$ для каждого направления скоростей для каждой компоненты (в случае многокомпонентного течения). Точки в трехмерном массиве ставятся в соответствие точкам в одномерном массиве следующим образом: $n = i \times N + x + y \cdot N_x + z \cdot N_x N_y$, где n — индекс в одномерном массиве, i — номер направления, x, y, z — координаты в трехмерном массиве. Для хранения значений f_i используются вещественные числа одинарной точности. Возможные направления упорядочены так, чтобы $\boldsymbol{e}_{-i} = \boldsymbol{e}_{i+(q-1)/2}$ при $i = 1, \ldots, (q-1)/2$, где q — количество скоростей в шаблоне DdQq ($q = 19$ в случае D3Q19 и $q = 7$ для D3Q7).

В случае моделирования процессов переноса в пористых средах с малой пористостью (менее 50%) эффективно хранить значения f_i только в проницаемых ячейках, этот принцип известен также как fluid-only. Для этого в момент загрузки программы все проницаемые ячейки нумеруются, составляется редуцированный массив со значениями f_i (и, если требуется, плотностей ρ) только в проницаемых ячейках. Кроме того, сохраняется карта соответствия $\mathfrak{M} : m \to (x_m, y_m, z_m)$, в которой каждому индексу m в редуцированном массиве ставится в соответствие точка (x_m, y_m, z_m) в начальном массиве. После этого в вычислениях используется редуцированный массив, а для определения соседних ячеек на шаге распространения используется карта $\mathfrak{M}$.

Использование обменного алгоритма и принципа fluid-only позволяет сократить количество используемой алгоритмом памяти более чем в 6 раз для образцов с пористостью около 33%.

4. Совместное использование CUDA и MPI. При блочном разбиении расчетная область разделяется на равные слои перпендикулярно оси Z (рис. 4).

Размер фрагментов и, соответственно, количество GPU подбираются на основе минимума по максимально доступной памяти на каждом из массивов ускорителей. Каждый фрагмент считается независимо в отдельном процессе на отдельном графическом ускорителе. Затем каждый процесс на каждой итерации на шаге распространения синхронизирует с соседними по номерам процессами значения функции распределения в граничных ячейках, пересылка сообщений между процессами осуществляется посредством MPI (обмен показан стрелками). Аналогичный подход используется в [28, 29, 36].

На каждом вычислительном узле суперкомпьютера “Ломоносов” в разделе gpu располагается по два графических ускорителя, поэтому для использования всех доступных видеокарт запускается по два процесса на каждый узел. При такой конфигурации запуска важно, чтобы каждый процесс использовал отдельную видеокарту, в то время как номера процессов могут быть распределены по вычислительным узлам случайным образом с единственным условием: не более двух на один узел. Для корректного задания соответствия номера видеокарты номеру процесса используется следующий алгоритм.

Алгоритм 2. Задание неповторяющихся номеров видеокарт в системе с несколькими GPU.

for *каждый процесс* **do**
    послать нулевому процессу имя узла;
end

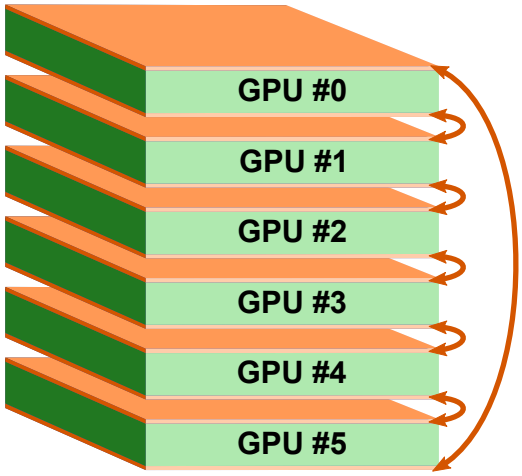


Рис. 4. Одномерное разбиение области на слои и распределение по нескольким процессам

if *нулевой процесс* **then**
принять и записать все имена хостов в один массив N;
линейно пройти по массиву и задать номер видеокарты по правилу:
номер видеокарты в процессе k равен числу вхождений имени узла в подмассиве N[0:k];
послать номера видеокарт всем процессам;
end
for *каждый процесс* **do**
получить номер GPU;
задать номер GPU;
end

5. Оценка масштабируемости программного комплекса. Все описанные в настоящей статье рекомендации были использованы при реализации программного модуля.

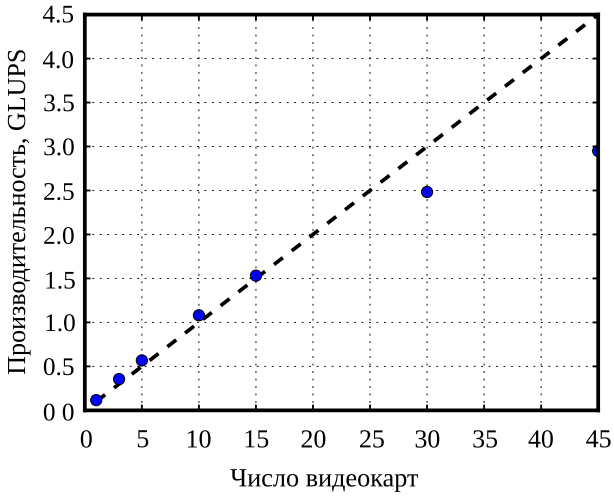


Рис. 5. Зависимость производительности от числа видеокарт. Вычислительный объем состоит из 450^3 ячеек, 38% которых являются проницаемыми

Была проведена серия измерений производительности модуля для расчетов однокомпонентного течения в зависимости от числа задействованных видеокарт на суперкомпьютере “Ломоносов”. Вплоть до 15 GPU виден линейный рост количества рассчитанных ячеек в единицу времени (в миллиардах ячеек в секунду, Giga Lattice Units Per Second) (рис. 5), далее наблюдается загиб тренда из-за задержек на пересылку сообщений по сети между узлами. По сравнению с работой [1] виден загиб линейного тренда

на больших числах задействованных GPU. Различие вызвано двумя факторами: во-первых, в этой работе рассматривалось максимум 30 GPU; во-вторых, существенно увеличена производительность расчетного модуля, поэтому задержки на уровне пересылки данных по сети начинают оказывать влияние на тренд. В общем случае чем больше вычислительных узлов используется в моделировании, тем с большей вероятностью они будут выделены в физически разнесенных частях суперкомпьютера и тем больше времени может потребоваться на пересылку сообщений между ними.

Заключение. В работе описаны общие принципы реализации метода решеточных уравнений Больцмана на гибридных суперкомпьютерных системах. При рассмотрении особенностей реализации использовался общий вид этого метода, что позволяет применять описанные оптимизации при реализации моделей различных процессов переноса: одно- и многокомпонентного течения, диффузии и теплопереноса.

Произведена оценка масштабируемости реализованного программного комплекса на суперкомпьютере "Ломоносов" вплоть до 45 GPU. Сублинейный рост производительности от числа видеокарт показывает целесообразность использования Multi-GPU подхода при массивно-параллельных расчетах.

Автор выражает благодарность сотрудникам суперкомпьютерного комплекса МГУ им. М. В. Ломоносова за предоставленный доступ к вычислительным ресурсам.

СПИСОК ЛИТЕРАТУРЫ

1. *Бикулов Д.А., Сенин Д.С., Демин Д.С., Дмитриев А.В., Грачев Н.Е.* Реализация метода решеточных уравнений Больцмана для расчетов на GPU-кластере // Вычислительные методы и программирование. 2012. **13**. 13–19.
2. *Бикулов Д.А., Сенин Д.С.* Реализация метода решеточных уравнений Больцмана без хранимых функций распределения для GPU // Вычислительные методы и программирование. 2013. **14**. 370–374.
3. *Грачев Н.Е., Дмитриев А.В., Сенин Д.С.* Моделирование динамики газа при помощи решеточного метода Больцмана // Вычислительные методы и программирование. 2011. **12**. 227–231.
4. *Захаров А.М., Сенин Д.С., Грачев Е.А.* Моделирование течений методом решеточных уравнений Больцмана со многими временами релаксации // Вычислительные методы и программирование. 2014. **15**. 644–657.
5. *Кривовичев Г.В.* Анализ устойчивости решеточных схем Больцмана для решения уравнения диффузии // Вычислительные методы и программирование. 2013. **14**. 175–182.
6. *Кривовичев Г.В.* О решеточной схеме Больцмана для расчетов на неструктурированных сетках // Вычислительные методы и программирование. 2013. **14**. 524–532.
7. *Куперштох А.Л.* Метод решеточных уравнений Больцмана для моделирования двухфазных систем типа жидкость–пар // Сб. научных статей "Современная наука". 2010. **4**, № 2. 56–63.
8. *Куперштох А.Л.* Трехмерное моделирование двухфазных систем типа жидкость–пар методом решеточных уравнений Больцмана на GPU // Вычислительные методы и программирование. 2012. **13**. 130–138.
9. *Banari A., Janssen C., Grilli S.T., Krafczyk M.* Efficient GPGPU implementation of a lattice Boltzmann model for multiphase flows with high density ratios // Computers & Fluids. 2014. **93**. 1–17.
10. *Bikulov D., Saratov A., Grachev E.* Prediction of the permeability of proppant packs under load // International Journal of Modern Physics C. 2015 (doi 10.1142/S012918311550117X).
11. *Boek E.* Pore Scale Simulation of Flow in Porous Media Using Lattice–Boltzmann Computer Simulations // Proc. SPE Annual Technical Conference and Exhibition (ATCE 2010). Vol. 6. Red Hook: Curran Associates, 2010. 5119–5132.
12. *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. Med. Biol. 2005. **50**, N 20. 4783–4796.
13. *Boyd J., Buick J., Green S.* A second-order accurate lattice Boltzmann non-Newtonian flow model // J. Phys. Math. Gen. 2006. **39**, N 46. 14241–14247.
14. *Chang L.-C., El-Araby E., Dang V.Q., Dao L.H.* GPU acceleration of nonlinear diffusion tensor estimation using CUDA and MPI // Neurocomputing. 2014. **135**. 328–338.
15. *Cheng J., Grossman M., McKercher T.* Professional CUDA C programming. New York: Wrox, 2014.
16. *D'Humiéres D., Ginzburg I., Krafczyk M., Lallemand P., Luo L.-S.* Multiple-relaxation-time lattice Boltzmann models in three dimensions // Philos. Trans. R. Soc. Lond. A. 2002. **360**. 437–451.
17. *Feichtinger C., Habich J., Köstler H., Rüde U., Aoki T.* Performance Modeling and Analysis of Heterogeneous Lattice Boltzmann Simulations on CPU-GPU Clusters // Parallel Computing. 2014. **46**, 1–13.
18. *Galizia A., d'Agostino D., Clematis A.* An MPI-CUDA library for image processing on HPC architectures // Journal of Computational and Applied Mathematics. 2015. **273**. 414–427.
19. *Hong P.-Y., Huang L.-M., Lin L.-S., Lin C.-A.* Scalable multi-relaxation-time lattice Boltzmann simulations on multi-GPU cluster // Computers & Fluids. 2014. **110**, 1–8.
20. *Ikeda M.K., Rao P.R., Schaefer L.A.* A thermal multicomponent lattice Boltzmann model // Computers & Fluids. 2014. **101**. 250–262.
21. *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 // Adv. Eng. Mater. 2011. **13**, N 3. 208–215.

22. *Kang Q.* Lattice Boltzmann model for crystal growth from supersaturated solution // Geophys. Res. Lett. 2004. **31**, N 21. 21604-1–L21604-5
23. *Leclaire S., Reggio M., Trépanier J.-Y.* Isotropic color gradient for simulating very high-density ratios with a two-phase flow lattice Boltzmann model // Comput. Fluids. 2011. **48**, N 1. 98–112.
24. *Lim C.Y., Shu C., Niu X.D., Chew Y.T.* Application of lattice Boltzmann method to simulate microchannel flows // Phys. Fluids. 2002. **14**, N 7. 2299–3009.
25. *Lu F., Song J., Yin F., Zhu X.* Performance evaluation of hybrid programming patterns for large CPU/GPU heterogeneous clusters // Computer Physics Communications. 2012. **183**, N 6. 1172–1181.
26. *McClure J.E., Prins J.F., Miller C.T.* A novel heterogeneous algorithm to simulate multiphase flow in porous media on multicore CPU–GPU systems // Computer Physics Communications. 2014. **185**, N 7. 1865–1874.
27. NVIDIA. NVIDIA's next generation CUDA compute architecture. Fermi: White Paper. NVIDIA. 2009. Available online (Version 1.1, 22 pages).
28. *Obrecht C., Kuznik F., Tourancheau B., Roux J.-J.* The TheLMA project: Multi-GPU implementation of the lattice Boltzmann method // Int. J. High Perform. Comput. Appl. 2011. **25**, N 3. 295–303.
29. *Obrecht C., Kuznik F., Tourancheau B., Roux J.-J.* Multi-GPU implementation of the lattice Boltzmann method // Comput. Math. Appl. 2011. **65**, N 2. 252–261.
30. Moscow University Supercomputing Center (http://parallel.ru/cluster).
31. *Pereira C., Mól A., Heimlich A., Moraes S., Resende P.* Development and performance analysis of a parallel Monte Carlo neutron transport simulation program for // Progress in Nuclear Energy GPU-Cluster using MPI and CUDA technologies. 2013. **65**. 88–94.
32. *Raabe D.* Overview of the lattice Boltzmann method for nano- and microscale fluid dynamics in materials science and engineering // Model. Simul. Mater. Sci. Eng. 2004. **12**, N 6. R13–R46.
33. *Rakić P.S., Milašinović D.D., Živanov Ž., Suvajdžin Z., Nikolić M., Hajduković M.* MPI-CUDA parallelization of a finite-strip program for geometric nonlinear analysis: a hybrid approach // Advances in Engineering Software. 2011. **42**, N 5. 273–285.
34. *Tölke J.* Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA // Comput. Vis. Sci. 2008. **13**, N 1. 29–39.
35. *Wittmann M., Zeiser T., Hager G., Wellein G.* Comparison of different propagation steps for lattice Boltzmann methods // Comput. Math. Appl. 2013. **65**, N 6. 924–935.
36. *Xian W., Takayuki A.* Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster // Parallel Comput. 2011. **37**, N 9. 521–535.
37. *Yang C.-T., Huang C.-L., Lin C.-F.* Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters // Computer Physics Communications. 2011. **182**, N 1. 266–269.
38. *Yoshida H., Nagaoka M.* Multiple-relaxation-time lattice Boltzmann model for the convection and anisotropic diffusion equation // Journal of Computational Physics. 2010. **229**, N 20. 7774–7795.

Поступила в редакцию
19.03.2015

An Efficient Implementation of the Lattice Boltzmann Method for Hybrid Supercomputers

D. A. Bikulov [1]

[1] *Lomonosov Moscow State University, Faculty of Physics; Leninskie Gory, Moscow, 119992, Russia; Graduate Student, e-mail: bikulov@physics.msu.ru*

Received March 19, 2015

Abstract: A number of features of an efficient implementation of the lattice Boltzmann method (LBM) for hybrid supercomputers with many graphics processing units (GPU) are discussed. The main strategies for reducing the memory space required by LBM are described. The performance dependence of the implemented solver on the number of the GPUs in use is analyzed for the Lomonosov supercomputer installed at Moscow State University.

Keywords: high-performance computing, graphics processing unit, lattice Boltzmann method, CUDA, multi-gpu, scalability.

References

1. D. A. Bikulov, D. S. Senin, D. S. Demin, et al., "Implementation of the Lattice Boltzmann Method on GPU Clusters," Vychisl. Metody Programm. **13**, 13–19 (2012).

2. D. A. Bikulov and D. S. Senin, “Implementation of the Lattice Boltzmann Method without Stored Distribution Functions on GPU,” Vychisl. Metody Programm. **14**, 370–374 (2013).

3. N. E. Grachev, A. V. Dmitriev, and D. S. Senin, “Simulation of Gas Dynamics with the Lattice Boltzmann Method,” Vychisl. Metody Programm. **12**, 227–231 (2011).

4. A. M. Zakharov, D. S. Senin, and E. A. Grachev, “Flow Simulation by the Lattice Boltzmann Method with Multiple-Relaxation Times,” Vychisl. Metody Programm. **15**, 644–657 (2014).

5. G. V. Krivovichev, “Stability Analysis of the Lattice Boltzmann Schemes for Solving the Diffusion Equation,” Vychisl. Metody Programm. **14**, 175–182 (2013).

6. G. V. Krivovichev, “A Lattice Boltzmann Scheme for Computing on Unstructured Meshes,” Vychisl. Metody Programm. **14**, 524–532 (2013).

7. A. L. Kupershtokh, “The Lattice Boltzmann Method for the Simulation of Two-Phase Liquid–Vapor Systems,” Sovremen. Nauka: Issled., Idei, Resul’taty, Tekhnol., No. 4, 56–63 (2010).

8. A. L. Kupershtokh, “Three-Dimensional Simulations of Two-Phase Liquid–Vapor Systems on GPU Using the Lattice Boltzmann Method,” Vychisl. Metody Programm. **13**, 130–138 (2012).

9. A. Banari, C. Janssen, S. T. Grilli, and M. Krafczyk, “Efficient GPGPU Implementation of a Lattice Boltzmann Model for Multiphase Flows with High Density Ratios,” Comput. Fluids **93**, 1–17 (2014).

10. D. Bikulov, A. Saratov, and E. Grachev, “Prediction of the Permeability of Proppant Packs under Load,” Int. J. Mod. Phys. C (2015). doi 10.1142/S012918311550117X

11. E. Boek, “Pore Scale Simulation of Flow in Porous Media Using Lattice-Boltzmann Computer Simulations,” in *Proc. SPE Annual Technical Conference and Exhibition (ATCE 2010), Florence, Italy, September 20–22, 2010* (Curran Associates, Red Hook, 2010), Vol. 6, pp. 5119–5132.

12. J. Boyd, J. Buick, J. A. Cosgrove, and P. Stansell, “Application of the Lattice Boltzmann Model to Simulated Stenosis Growth in a Two-Dimensional Carotid Artery,” Phys. Med. Biol. **50** (20), 4783–4796.

13. J. Boyd, J. Buick, and S. Green, “A Second-Order Accurate Lattice Boltzmann Non-Newtonian Flow Model,” J. Phys. A: Math. Gen. **39** (46), 14241–14247 (2006).

14. L.-C. Chang, E. El-Araby, V. Q. Dang, and L. H. Dao, “GPU Acceleration of Nonlinear Diffusion Tensor Estimation Using CUDA and MPI,” Neurocomputing **135**, 328–338 (2014).

15. J. Cheng, M. Grossman, and T. McKercher, *Professional CUDA C Programming* (Wrox, New York, 2014).

16. D. d’Humiéres, I. Ginzburg, M. Krafczyk, et al., “Multiple-Relaxation-Time Lattice Boltzmann Models in Three Dimensions,” Phil. Trans. R. Soc. Lond. A **360**, 437–451 (2002).

17. C. Feichtinger, J. Habich, H. Köstler, et al., “Performance Modeling and Analysis of Heterogeneous Lattice Boltzmann Simulations on CPU-GPU Clusters,” Parallel Comput. **46**, 1–13 (2014).

18. A. Galizia, D. d’Agostino, and A. Clematis, “An MPI-CUDA Library for Image Processing on HPC Architectures,” J. Comput. Appl. Math. **273**, 414–427 (2015).

19. P.-Y. Hong, L.-M. Huang, L.-S. Lin, and C.-A. Lin, “Scalable Multi-Relaxation-Time Lattice Boltzmann Simulations on Multi-GPU Cluster,” Comput. Fluids **110**, 1–8 (2014).

20. M. K. Ikeda, P. R. Rao, and L. A. Schaefer, “A Thermal Multicomponent Lattice Boltzmann Model,” Comput. Fluids **101**, 250–262 (2014).

21. Z. Jiang, K. Wu, G. D. Couples, and J. Ma, “The Impact of Pore Size and Pore Connectivity on Single-Phase Fluid Flow in Porous Media,” Adv. Eng. Mater. **13** (3), 208–215 (2011).

22. Q. Kang, D. Zhang, P. C. Lichtner, and I. N. Tsimpanogiannis, “Lattice Boltzmann Model for Crystal Growth from Supersaturated Solution,” Geophys. Res. Lett. **31** (21), L21604-1–L21604-5 (2004).

23. S. Leclaire, M. Reggio, and J.-Y. Trépanier, “Isotropic Color Gradient for Simulating Very High-Density Ratios with a Two-Phase Flow Lattice Boltzmann Model,” Comput. Fluids **48** (1), 98–112 (2011).

24. C. Y. Lim, C. Shu, X. D. Niu, and Y. T. Chew, “Application of Lattice Boltzmann Method to Simulate Microchannel Flows,” Phys. Fluids **14** (7), 2299–3009 (2002).

25. F. Lu, J. Song, F. Yin, and X. Zhu, “Performance Evaluation of Hybrid Programming Patterns for Large CPU/GPU Heterogeneous Clusters,” Comput. Phys. Commun. **183** (6), 1172–1181 (2012).

26. J. E. McClure, J. F. Prins, and C. T. Miller, “A Novel Heterogeneous Algorithm to Simulate Multiphase Flow in Porous Media on Multicore CPU–GPU Systems,” Comput. Phys. Commun. **185** (7), 1865–1874 (2014).

27. NVIDIA. NVIDIA’s next generation CUDA compute architecture.
http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf. Cited March 17, 2015.

28. C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux, “The TheLMA Project: Multi-GPU Implementation of the Lattice Boltzmann Method,” Int. J. High Perform. Comput. Appl. **25** (3), 295–303 (2011).

29. C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux, "Multi-GPU Implementation of the Lattice Boltzmann Method," Comput. Math. Appl. **65** (2), 252–261 (2011).

30. Moscow University Supercomputing Center. http://parallel.ru/cluster. Cited March 17, 2015.

31. C. Pereira, A. Mól, A. Heimlich, et al., "Development and Performance Analysis of a Parallel Monte Carlo Neutron Transport Simulation Program for GPU–Cluster Using MPI and CUDA Technologies," Prog. Nucl. Energy **65**, 88–94 (2013).

32. D. Raabe, "Overview of the Lattice Boltzmann Method for Nano- and Microscale Fluid Dynamics in Materials Science and Engineering," Model. Simul. Mater. Sci. Eng. **12** (6), R13–R46 (2004).

33. P. S. Rakić, D. D. Milašinović, *Ž*. *Ž*ivanov, et al., "MPI–CUDA Parallelization of a Finite-Strip Program for Geometric Nonlinear Analysis: A Hybrid Approach," Adv. Eng. Softw. **42** (5), 273–285 (2011).

34. J. Tölke, "Implementation of a Lattice Boltzmann Kernel Using the Compute Unified Device Architecture Developed by nVIDIA," Comput. Visual. Sci. **13** (1), 29–39 (2010).

35. M. Wittmann, T. Zeiser, G. Hager, and G. Wellein, "Comparison of Different Propagation Steps for Lattice Boltzmann Methods," Comput. Math. Appl. **65** (6), 924–935 (2013).

36. W. Xian and A. Takayuki, "Multi-GPU Performance of Incompressible Flow Computation by Lattice Boltzmann Method on GPU Cluster," Parallel Comput. **37** (9), 521–535 (2011).

37. C.-T. Yang, C.-L. Huang, and C.-F. Lin, "Hybrid CUDA, OpenMP, and MPI Parallel Programming on Multicore GPU Clusters," Computer Phys. Commun. **182** (1), 266–269 (2011).

38. H. Yoshida and M. Nagaoka, "Multiple-Relaxation-Time Lattice Boltzmann Model for the Convection and Anisotropic Diffusion Equation," J. Comput. Phys. **229** (20), 7774–7795 (2010).