Жидкие кристаллы и их практическое использование. 2011. Вып. 3 (37). С. 76 – 83

УДК 004.272.2: 519.63

В. В. Соцкий

ОПЫТ ПРИМЕНЕНИЯ ГРАФИЧЕСКИХ КОНТРОЛЛЕРОВ К РЕШЕНИЮ ЗАДАЧ МОЛЕКУЛЯРНОЙ ДИНАМИКИ

EXPERIENCE IN GPU APPLYING TO MOLECULAR DYNAMICS

Ивановский государственный университет, НИИ наноматериалов
153025 Иваново, Ермака, 39. E-mail: nv_usoltseva@mail.ru

Разработаны алгоритмы для метода молекулярной динамики, позволяющие проводить численные эксперименты на современных программируемых видеоконтроллерах. Прирост скорости расчетов, в сравнении с двухпроцессорным вариантом (процессор Intel Core 2 Duo 2.2 Ггц), составляет 28,1 раза (видеокарта GeForce 9800 GT) и 49,2 раза (видеокарта GeForce 295 GT).

Ключевые слова: *графические контроллеры, CUDA, параллельные вычисления, молекулярная динамика.*

Algorithms for molecular dynamics allowing numerical experiments on GPU were developed. Acceleration of calculations in comparison with the dual core processor (Intel Core 2 Duo 2.2 GHz) were 28.1 times for GeForce 9800 GT and 49.2 times for GeForce 295 GT.

Key words: *GPU, CUDA, parallel computing, molecular dynamics.*

Введение

Современные методы физических исследований позволяют детально изучать структуру и получать характеристики материалов. Однако, этих данных не всегда достаточно для интерпретации нано- и мезоявлений, особенно происходящих за субнаносекундные интервалы времени. Способом решения данной проблемы является создание системы математических моделей, позволяющих получать интересующие характеристики. Наиболее используемыми методами в практике компьютерного моделирования являются методы квантовой химии и метод молекулярной динамики. Если рассматривать специфику жидких кристаллов, то их свойства начинают проявляться при достаточно большом количестве молекул. Однако, методы квантовой химии здесь становятся трудно применимыми, ввиду большого количества расчетов. Метод молекулярной механики менее требователен к ресурсам и его можно применять для систем, состоящих из сотен тысяч и миллионов атомов. Но и это требует применения многопроцессорных вычислительных систем. Одним из типов таких систем являются системы на основе графических ускорителей, основное достоинство которых – низкая стоимость, в сравнении с аналогичными по производительности системами. В то же время, графические ускорители обладают рядом особенностей, требующих модификации уже существующих алгоритмов расчетов для эффективного использования ресурсов.

Особенности графических контроллеров

В современных графических ускорителях (GPU) устанавливается большое число вершинных и фрагментных процессоров (до 1024 в двухпроцессорных картах Nvidia), которые по быстродействию обгоняют центральные процессоры (CPU) (рис. 1) [1]. По приведенным данным видно, что одна графическая карта сопоставима по производительности с системой из центральных процессоров (небольшими кластерами).

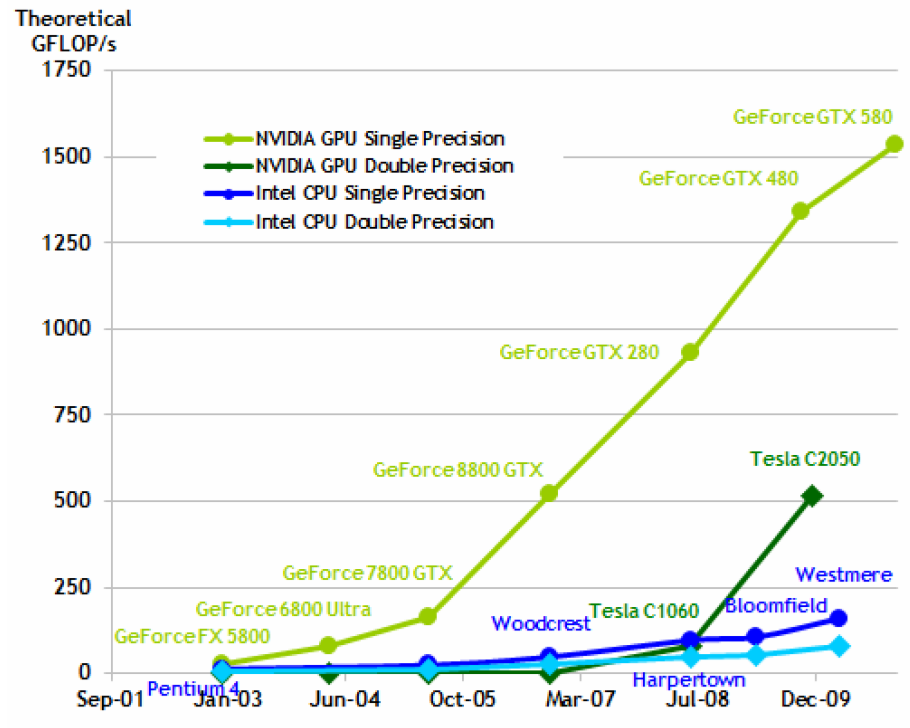

Рис. 1. Сравнение производительности центральных процессоров и видеокарт Nvidia [1]

Использование графических ядер для решения задач математического моделирования (таких как задачи молекулярной динамики), ввиду большой производительности данных устройств, позволяет существенно ускорить расчеты, а также увеличить размерность рассматриваемых систем. Однако, графические ускорители являются специализированными устройствами, изначально не предназначенными для решения сложных математических задач. С появлением интерфейса программирования CUDA стало возможным разрабатывать программное обеспечение на языке высокого уровня (подобного языкам C и C++). При этом GPU рассматривается в нем как специализированное вычислительное устройство, которое является сопроцессором к CPU, обладает собственной памятью (DRAM) и возможностью параллельного выполнения огромного количества отдельных нитей. При этом их создание и управление требует минимальных ресурсов, а для эффективной реализации возможностей GPU нужно использовать многие тысячи таких нитей. Это связано с тем, что при выполнении программы чередуется выполнение расчетов и чтение данных из памяти, а также с особенностью выполнения участков кода на графическом ядре. Одновременно могут выполняться только идентичные участки. Если, например, в коде присутствует разветвление, то сначала выполнятся все участки с первым условием, затем все участки со вторым.

Другой особенностью вычислений является выделение программных блоков, каждый из которых выполняется на мультипроцессоре, состоящем из 8-ми потоковых процессоров. Если количество блоков будет меньше чем число мультипроцессоров, возможности графического ядра не будут полностью использованы [2].

Кроме того существует также несколько различных типов памяти. Поскольку быстродействие приложения очень сильно зависит от скорости работы с памятью, в центральном процессоре большую часть кристалла занимает кэш, предназначенный для ускорения работы с памятью, в то время как для GPU основную часть кристалла занимают арифметико-логические устройства (рис. 2).

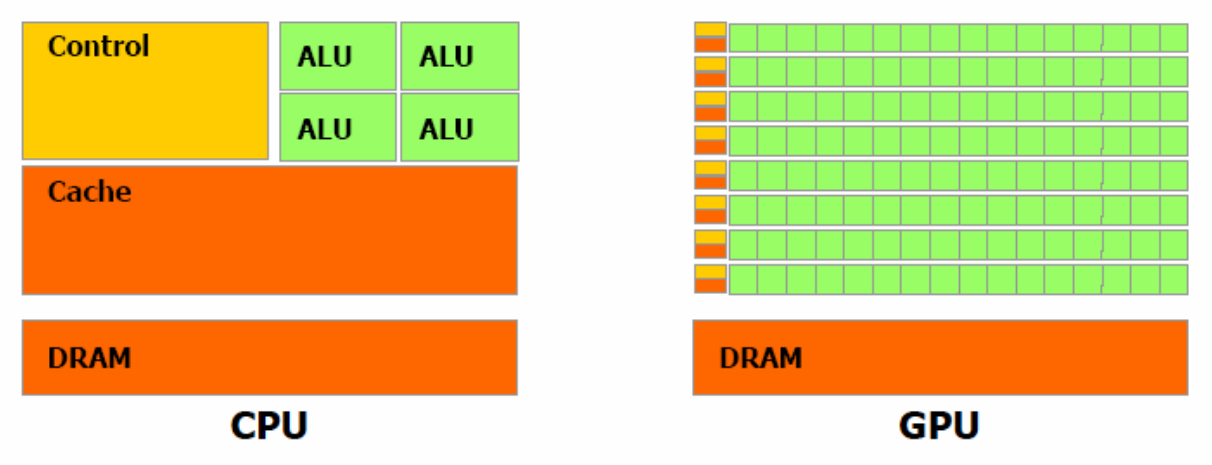

Рис. 2. Сравнение архитектуры центрального процессора и графического ядра

В CUDA для GPU существует несколько различных типов памяти, доступных потокам, сильно различающихся между собой (см. табл.).

Типы памяти в CUDA

Тип памяти	Уровень выделения	Доступ для GPU	Скорость работы
registers	поток	Чтение/запись	Высокая
local	поток	Чтение/запись	Низкая
shared	блок	Чтение/запись	Высокая
global	всем потокам	Чтение/запись	Низкая
constant	всем потокам	Только чтение	Высокая
texture	всем потокам	Только чтение	Высокая

При этом CPU имеет доступ на чтение/запись только к глобальной, константной и текстурной памяти (находящейся в DRAM GPU) и только через функции копирования памяти между CPU и GPU (предоставляемые CUDA API) [2]. Основными проблемами, связанными с наличием нескольких видов памяти, являются следующие: в процессе реализации алгоритмов необходимо самостоятельно создавать кэш и проверять соответствие записанных в нем данных, учитывать объем памяти для размещения соответствующих массивов, учитывать последовательность считывания из памяти (доступ при последовательной выборке осуществляется быстрее).

Все эти особенности накладывают ограничения на использования алгоритмов молекулярной динамики, разработанных для использования на центральных процессорах. Различия в особенностях архитектуры CPU и GPU требуют создания новых алгоритмов, при этом необходимо учитывать и особенности самого метода молекулярной динамики [3].

Реализация расчетов на GPU

В качестве метода, позволяющего моделировать системы, состоящие из нескольких тысяч частиц, нами был выбран метод молекулярной механики [4]. Основу метода составляет описание исследуемой системы при помощи потенциалов парного взаимодействия. Полная энергия системы складывается из энергии взаимодействия валентных связей, валентных углов, торсионных углов, несвязанных и электростатических взаимодействий:

$$U_{пот} = \sum_{1}^{N} k_r (r - r_0)^2 + k_\alpha (\alpha - \alpha_0)^2 +$$

$$+ \frac{V_\phi}{2}(1 + \cos(\phi n - \phi_0)) + \left(\frac{A}{r^{12}} - \frac{B}{r^6}\right) + \frac{q_1 q_2}{\varepsilon r}.$$

Константы при уравнениях были взяты из силового поля AMBER [5]. После этого система получает механическое описание и можно проводить численные эксперименты методом молекулярной динамики. Основу метода составляет численное решение уравнений второго закона Ньютона для системы взаимодействующих частиц:

$$m_i \frac{d^2 \vec{r}_i(t)}{dt^2} = \vec{F}_i(\vec{r}), \;\; i = 1, 2, ..., N$$

или

$$\begin{cases} m \dfrac{d\vec{\upsilon}_i}{dt} = \vec{F}_i \\ \dfrac{dx_i}{dt} = \vec{\upsilon}_i \end{cases} \qquad i = 1, 2, ..., N,$$

где $\vec{r}_i$ – радиус-вектор i-го атома, m_i – его масса, $\vec{F}_i$ – суммарная сила, действующая на i-ый атом со стороны остальных частиц, которая определяется как частная производная соответствующего потенциала по координатам, взятая с обратным знаком:

$$\vec{F}_i(\vec{r}) = -\frac{\partial U(\vec{r})}{\partial \vec{r}_i}$$

Здесь: $\vec{r} = \{\vec{r}_1, \vec{r}_2, ..., \vec{r}_n\}$; $U(\vec{r})$ – потенциальная энергия, зависящая от взаимного расположения всех атомов.

Дополнительно, для поддержания термодинамических характеристик в моделируемой системе, применялась совокупность методов, приведенная ранее в публикации [6].

Наиболее затратной операцией при решении задач молекулярной динамики является вычисление суммы сил, действующих на атомы, поскольку количество уравнений пропорционально квадрату количества частиц в системе. Поэтому суммирование лучше проводить на графическом ядре, а интегрирование уравнений движения, занимающих меньшую часть вычислений (количество пропорционально числу частиц) [7], на центральном процессоре (рис. 3). Функция передачи данных от CPU к GPU является синхронизирующим барьером для всех потоков, выполняющихся на видеокарте. Также,

данная схема наиболее удобна для систем, содержащих несколько графических контроллеров, что делает возможным дальнейшую модернизацию алгоритма.

Рис. 3. Блок-схема вычислений по методу молекулярной динамики при использовании графических процессоров

Для реализации задач молекулярной динамики, в рамках модели программирования CUDA, была предложена следующая концепция: поскольку для эффективного использования ресурсов GPU требуется создание большого количества потоков, каждому атому выделяется отдельный поток. При этом в нем рассчитываются все действующие на атом силы (рис. 4), что обеспечивает независимость потоков друг от друга. Тогда эффективность использования графического ядра растет пропорционально количеству атомов в системе (поскольку увеличивается количество программных блоков) [8].

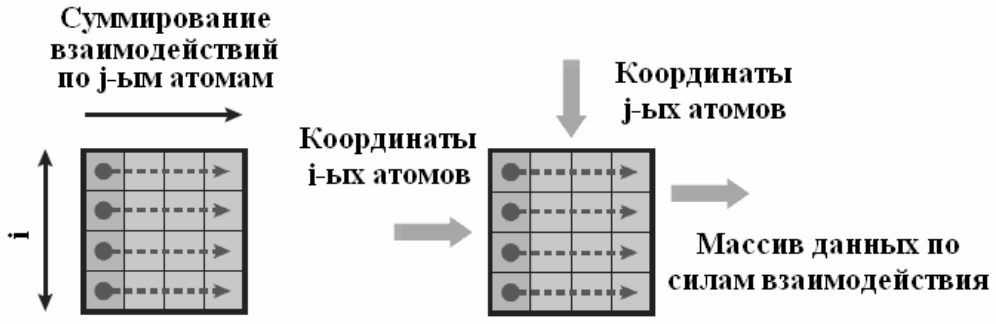

Рис. 4. Схема использования графического ядра в задачах молекулярной динамики

Описание молекулярной системы требует большого количества оперативной памяти для хранения координат, параметров атомов, силовых постоянных внутримолекулярного взаимодействия. Большим объемом обладает только глобальная память видеокарты, однако доступ к ней может достигать 400 – 600 циклов. Частично эта про-

блема решается использованием схемы расчетов со множеством потоков – запросы на чтение и запись чередуются с расчетами на графическом ядре. В то же время, более быстрым доступом обладает константная и общая память. Ограничением для константной памяти является доступ к ней для записи только с CPU, общая же память открыта для записи и чтения графическому ядру. Однако, объем общей памяти составляет 16 кб (48 кб для новых версий видеокарт) на один мультипроцессор, а объем данных, используемых в процессе расчета достигает сотен мегабайт. Для решения этой проблемы был создан алгоритм блочного копирования данных из глобальной в общую память. Каждый поток копирует одно значение из массива глобальной памяти в массив локальной памяти (рис. 5), с предварительной синхронизацией, затем, после цикла расчетов, производится дополнительная синхронизация в блоке, что гарантирует соответствие находящихся в общей памяти данных. При этом на N обращений к глобальной памяти приходится $N \cdot N - N$ обращений к общей памяти, за счет чего и обеспечивается прирост производительности программы. Сравнительные численные эксперименты показали, что использование данного алгоритма обеспечивает прирост производительности приложения в 4,13 раза.

Рис. 5. Схема работы алгоритма блочного копирования данных

Для сравнения эффективности использованного алгоритма проводились следующие вычислительные эксперименты: запускались приложения скомпилированные с одинаковыми ключами (ключ –O2 для оптимизации) и брались молекулярные тестовые системы одинаковой размерности с одинаковыми параметрами моделирования (шаг интегрирования, температура, плотность). Затем измерялось количество итераций, выполненных на центральном процессоре и видеокарте за минуту реального времени. Для сравнения использовались процессор Intel Core 2 Duo 2.2 GHz (параллельный вариант программы) и видеокарты Nvidia GeForce 9800GT (112 потоковых процессоров) и Nvidia GeForce 8600 GT (32 потоковых процессора). Зависимость эффективности использования графических ядер от количества частиц представлена на рис. 6.

Из приведенных зависимостей видно, что максимальное ускорение достигается при количестве частиц, пропорциональном количеству потоковых процессоров видеокарты, с учетом выделенных потоков (в нашем случае выделялось 256 потоков на один блок). Рост скорости расчетов, в сравнении с двухпроцессорным вариантом, составил: 7

раз (видеокарта GeForce 8600 GT) и 28,1 раза (видеокарта GeForce 9800 GT). Дополнительно проводились аналогичные эксперименты с видеокартой GeForce 295 GT (использовалось только одно графическое ядро), при этом производительность возросла в 49,2 раза.

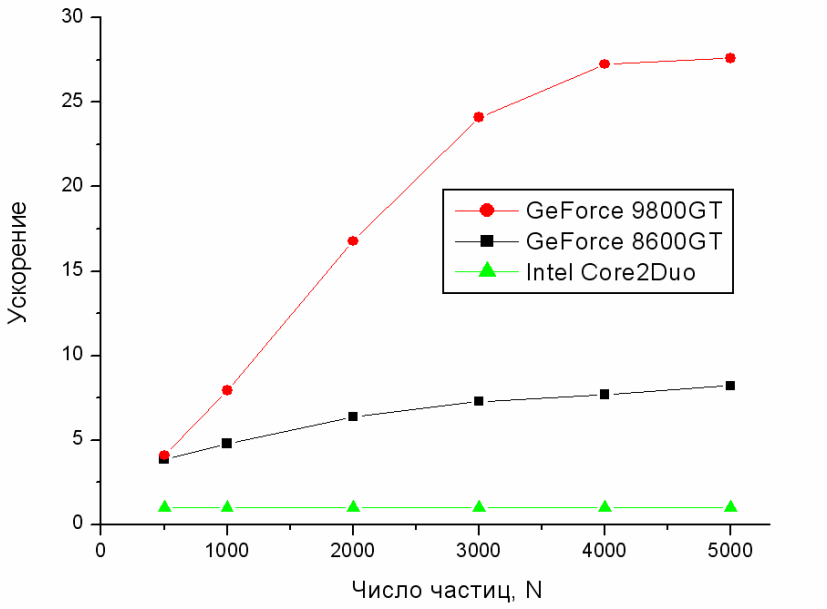

Рис. 6. Зависимость эффективности использования графических ядер от количества частиц в моделируемой системе

С применением описанных выше методов нами была разработана система молекулярно-динамического моделирования [9] с дополнительным модулем [10], которые в совокупности позволяют проводить численное моделирование на видеокартах динамики поведения жидкокристаллических систем.

Выводы

Разработан алгоритм для метода молекулярной динамики, позволяющий проводить численные эксперименты на современных программируемых видеоконтроллерах.

Создан алгоритм блочно-циклического кэширования данных, дающий рост производительности программного обеспечения (4,13 раза).

Рост скорости расчетов, в сравнении с двухпроцессорным вариантом (процессор Intel Core 2 Duo 2.2 Ггц), составляет: 7 раз (видеокарта GeForce 8600 GT), 28,1 раза (видеокарта GeForce 9800 GT) и 49,2 раза (видеокарта GeForce 295 GT).

Разработанные алгоритмы реализованы в виде программного обеспечения [9, 10].

Работа поддержана грантом ФЦП 16.740.11.0206.

Список литературы

1. Документация по интерфейсу программирования CUDA, CUDA Documentation (http://developer.nvidia.com/category/zone/cuda-zone).
2. *Боресков А. В., Харламов А. А.* Основы работы с технологией CUDA // Издательство : ДМК, 2010. 232 с.
3. *Боярченков А. С., Поташников С. И.* // Вычислительные методы и программирование. 2009. Т. 10. С. 9 – 23.
4. *Allinger N. L., Burkett U.* Molecular Mechanics. ACS, Washington DC. 1982. 326 p.
5. *Cornell W. D., Cieplak P., Bayly C. I. et al.* // J. Am. Chem. Soc. 1995. Vol. 117. P. 5179 – 5197.
6. *Усольцева Н. В., Ясинский Ф. Н., Соцкий В. В., Костин М. С.* // Вестник ИГЭУ. Иваново, 2009. Вып. 4. С. 45 – 48.
7. *Калиткин Н. Н.* Численные методы. М. : Наука, 1979. 512 с.
8. *Nyland L., Harris M., Prins J.* // GPU Gems3. Chapter 31. Addison Wiesley. 2007. 1230 p.
9. *Соцкий В. В.* Свидетельство № 2011613855 об официальной регистрации программы для ЭВМ «Система молекулярно-динамического моделирования MDsimGrid» в Федеральной службе по интеллектуальной собственности, патентам и товарным знакам. М., 2011.
10. *Соцкий В. В.* Свидетельство № 2011613854 об официальной регистрации программы для ЭВМ «Программный модуль молекулярно-динамического моделирования MDsimGrid-GPU» в Федеральной службе по интеллектуальной собственности, патентам и товарным знакам. М., 2011.

Поступила в редакцию 15.06.2011 г.