РЕАЛИЗАЦИЯ МЕТОДА МАКРОЧАСТИЦ ФИЗИКИ ПЛАЗМЫ НА ГРАФИЧЕСКИХ УСКОРИТЕЛЯХ

А.Ю. Перепёлкина, В.Д. Левченко

ИПМ им. М.В. Келдыша РАН

Введение

В настоящее время появляется множество реализаций различных численных методов на графических ускорителях. Метод макрочастиц не стал исключением [1-3]. Использование графических ускорителей в несколько раз увеличивает скорость вычислений по сравнению с аналогичными версиями на центральных процессорах, но для актуальных задач пока приходится использовать гетерогенные системы с большим числом ядер [2]. Это связано с тем, что существующие реализации устроены так, что требуется большое количество видеокарт для того, чтобы вместить все необходимые для вычислений данные. Эффективные решения существуют для размерности 2D3V [3], но исследование актуальных задач физики плазмы требует трехмерных моделей.

Целью данной работы является поиск наиболее оптимальной реализации метода частиц на графических ускорителях. Используется технология CUDA и языки C++ и Python.

Математическая модель и численные схемы

В качестве математической основы при моделировании бесстолкновительной полностью ионизированной плазмы используется 3D3V система уравнений Власова-Максвелла, из которой при подстановке дискретной функции распределения частиц плазмы получается система дифференциальных уравнений движения макрочастиц, дополненная конечно-разностной аппроксимацией системы уравнений Максвелла для самосогласованных электромагнитных полей.

Численный расчет на каждом шаге по времени состоит из трех этапов. Значения электромагнитных полей обновляются исходя из их предыдущих значений и токов по схеме конечных разностей во временной области (FDTD) второго или четвертого порядка. В этой схеме дискретизация компонент полей проведена на сдвинутых сетках, поэтому для последующих вычислений все значения полей интерполируются на центры ячеек. На втором этапе вычисляются силы, действующие на макрочастицы, проходит их продвижение с учетом релятивизма по схеме Бо́риса. Затем из нового и старого положения частиц вычисляются токи на сетке, и эти значения токов используются для вычисления полей на первом этапе следующего шага.

Учитывая, что ограничения на шаг по времени для обновления полей (условие Куранта) и на шаг для макрочастиц (определяется из плазменной частоты), могут быть различны, удобно отношение этих двух шагов сделать параметром кода. Тогда между двумя последующими передвижениями частиц будет проходить несколько обновлений полей.

Граничные условия по осям y и z периодичны, по оси x – отражение.

Алгоритм обновления полей

Учитывая, что схема для полей имеет шаблон «крест», можно воспользоваться алгоритмом DiamondTorre 2D, который в данном случае решает две основные проблемы, ограничивающих эффективность вычислений на графическом ускорителе: обеспечение векторизованного доступа к данным и преодоление дисбаланса пропускной способности памяти относительно вычислительной производительности (менее 0.1байт/FLOP при требуемых 4). В данном случае векторизация делается по оси z, а требования к пропускной способности памяти снижаются за счёт высокой локальности вычислений алгоритма DiamondTorre. В общих чертах алгоритм, вместе с реализацией при помощи средств CUDA, можно описать так.

Данные полей представлены в виде двумерного массива Nx×Ny, в каждом элементе которого содержатся 9 векторов длиной Nz для трех компонент электрических полей, магнитных полей и плотности токов. Электрические поля и токи определены на гранях ячеек и на полуцелых шагах по времени, магнитные поля – на ребрах и на целых шагах по времени. Под Diamond-ячейкой с координатой (ix,iy) имеются ввиду компоненты полей, проекции которых на плоскость x-y попадают в ромб заданного размера. Для двух полушагов во времени, относящихся к магнитным и электрическим полям соответсвенно, ромб сдвинут на половину ячейки для схемы второго порядка, и на полторы ячейки для схемы четвертого порядка (рисунок 1).

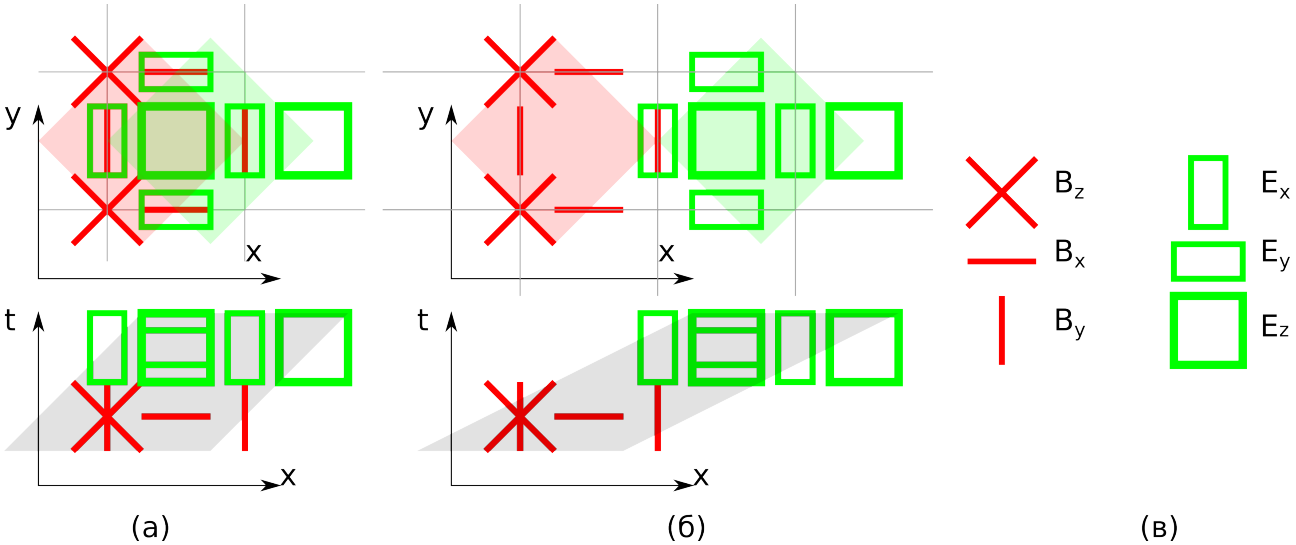

Рис. 1. Diamond-ячейка для схемы FDTD второго порядка (а), схемы четвертого порядка (б), расшифровка обозначений (в).

Сначала вычисления проходят в Diamond-ячейках с индексами (Nx-1,iy,iz) для всех iz от 0 до Nz-1, для всех чётных iy от 0 до Ny-2 (некоторые компоненты в такой ячейке оказываются за границей области; в них вычисления не происходят). Для каждого слоя по y выделен CUDA-блок. В одном блоке обработка данных ячеек по оси z распределена между CUDA-нитями. Для вычислений разностей по оси z необходимые значения сначала сохраняются в shared-память, CUDA-нити синхронизируются, и затем вычисляются разности.

Из этих вычислений сначала обновляются магнитные поля, затем электрические.

После этого так же проходят вычисления в ячейках (Nx-2,iy,iz), iz=1..Nz-1, iy = 1,3..Ny-1 (нечётные). Сдвигаясь каждый раз на 1 по x налево и на ±1 по y, получается покрыть всю область. Все значения компонент полей области обновятся на один шаг по времени.

Бывает необходимо обновить поля на несколько шагов по времени перед тем, как переходить к передвижению частиц и вычислению токов. Тогда каждый раз после вычислений в Diamond-ячейке с координатой (ix,iy,iz), сразу проводятся вычисления в ячейках (ix+(NO-1)*it,iy,iz), где NO – порядок схемы, it = 1..Nt-1, Nt – количество шагов для полей между обновлениями токов. Так, до завершения обхода области различные компоненты отвечают разным слоям по времени, от 0 до Nt. При завершении обхода все данные отвечают одному слою по времени (с точностью до полушага между полями E и B).

В расчёте на одну видеокарту обновление полей проходит с темпом до 1.2×10^9 ячеек в секунду для схемы четвертого порядка, 3.5×10^9 – для второго на устройстве nVidia GeForce GTX 750Ti.

Кодогенерация

Исходный код CUDA-ядер для такой задачи достаточно сложен. Diamond-ячейка имеет повторяющиеся вычисления в различных точках, которые при этом могут быть отличны вблизи границ, из-за подстановки граничных условий. Только для схемы для полей подобный код занимает более тысячи строк.

Благодаря тому, что эти строки получаются из вида схемы и реализации алгоритма по определенным несложным правилам, удобно создавать код вычислений при помощи небольшой программы на языке Python, называемой в данном контексте кодогенератором.

Алгоритм продвижения частиц и подсчета токов

В данном случае векторизация делается по частицам, что приводит к проблеме локализации данных, используемых при ускорении и продвижении частиц, а именно сеточных значений полей и токов.

Перед вычислениями для частиц полученные на предыдущем этапе поля интерполируются в центры ячеек. Из этих значений будут вычисляться силы, действующие на каждую частицу, поэтому выгодно создать для компонент полей в центрах ячеек массив, и обращаться к нему через текстурный кэш видеокарты. Значения компонент токов в ячейках в этот момент зануляются, чтобы подготовить их расчет из передвижений частиц.

Частицы хранятся в структуре, обеспечивающей с одной стороны частичную сортировку по пространственным координатам, а с другой – векторизованный доступ. Один CUDA-блок используется для вычислений в одном кубе с размером 8×8×8 ячеек. Сначала создается массив для компонент тока в кубе в 16^3 ячеек, содержащий нули, в shared-памяти. В кубе с размером 8×8×8 происходит ускорение частиц, передвижение с переходом в соседние ячейки, но не дальше, чем на 4 ячейки по каждой координате от исходной ячейки. В таком случае плотности токов могут обновиться только в кубе в 16^3 ячеек.

Модуль CUDA-ядер для частиц реализован отдельно. За секунду обрабатываются более 3×10^8 частиц в расчёте на одну видеокарту.

Обновление плотностей токов проходит при помощи атомарных операций.

Чтобы вычисления для кубов не перекрывались, вся область вычислений разбивается на кубы по 16^3 ячеек. Такие блоки содержат 8 кубов по 8^3 ячеек. В одном слое таких блоков (например, от Nx-17 до Nx-1) производятся вычисления для одного из подкубов каждого куба асинхронно (см. рисунок 3). Затем еще семь оставшихся подкубов обрабатываются так же, и вычисления переходят на следующий слой (от Nx-34 до Nx-18). Получение токов из перемещений частиц – аддитивная операция. После вычислений в каждом подкубе токи для 16^3 ячеек сохраняются (добавлением) в основной массив.

DiamondTorre и окно вычислений

Выше вычисления для полей и частиц описаны отдельно, и вычисления проходят справа налево по оси x. При совместной реализации описанных алгоритмов возникает возможность использовать окно вычислений. Так, для ячеек, где поля посчитаны для Nt шагов, можно начинать передвижения макрочастиц и подсчет плотности токов. В ячейках, где посчитаны токи, можно начинать последующие обновления полей и т. д. до некоторого шага по времени T. Так, при достаточном количестве ячеек по x, часть данных могут отвечать начальному условию, часть – моменту T, и также некоторое количество данных на всех шагах от 0 до T, над которыми в конкретный момент проходят вычисления. Этот «склон» является окном вычислений при такой реализации. Только эти данные необходимо помещать в памяти видеокарты. Таким образом, снимается ограничение на размер области, которую можно моделировать на одной видеокарте.

Алгоритм обобщается на гетерогенные вычислительные системы с кластерной архитектурой. В этом случае размер задачи ограничен суммарной оперативной памятью вычислительных узлов, а накладные расходы на пересылку данных между видеокартами пренебрежимо малы.

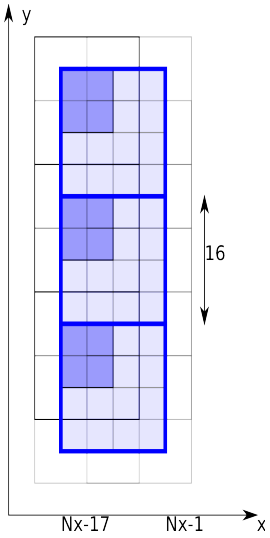

Рис. 2. Схематичное изображение асинхронных блоков при смещении частиц и вычислении токов в проекции на плоскость x-y

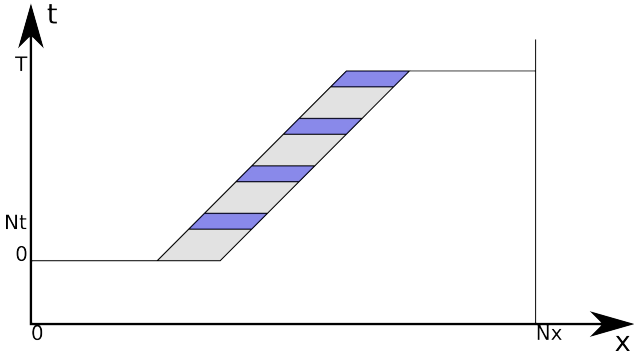

Рис. 3. Схематичное изображение окна вычислений алгоритма. Серым отмечены операции над полями, синим – над частицами и токами

Заключение

Реализованы эффективные алгоритмы для моделирования плазмы методом частиц для графических ускорителей. Основное преимущество данного подхода состоит в снятии ограничения на размер задачи, доступной для моделирования на одной видеокарте при сохранении высокого темпа счета за счет использования алгоритма DiamondTorre. Алгоритм допускает обобщение на гетерогенные вычислительные системы с кластерной архитектурой

Работа поддержана грантами РФФИ 14-01-31483-мол_а,12-01-00708-а.

ЛИТЕРАТУРА:

1. S. Bastrakov, R. Donchenko, A. Gonoskov, E. Efimenko, A. Malyshev, I. Meyerov, I. Surmin, Particle-in-cell plasma simulation on heterogeneous cluster systems, Journal of Computational Science, Volume 3, Issue 6, November 2012,
2. A. Huebl et al., Visualizing the Radiation of the Kelvin-Helmholtz Instability, 2014, arXiv:1404.2507 [physics.plasm-ph]
3. Xianglong Kong, et al., Particle-in-cell simulations with charge-conserving current deposition on graphic processing units // Journal of Computational Physics, Volume 230, Issue 4, 20 February 2011,