Математическое моделирование

УДК 519.712.45, 519.692, 519.694, 519.681.5

Использование технологии NVIDIA CUDA при моделировании динамики пучка в ускорителях заряженных частиц

Е. Е. Перепелкин [*], **В. Л. Смирнов** [*], **С. Б. Ворожцов** [†]

[*] *Лаборатория физики высоких энергий*
Объединённый институт ядерных исследований
141980, Дубна, Московская область, Россия

[†] *Лаборатория ядерных проблем*
Объединённый институт ядерных исследований
141980, Дубна, Московская область, Россия

Оптимизация центральной области циклотрона требует многократного повторения процедуры проводки пучка с большим количеством частиц. Данная задача существенно усложняется, когда встаёт вопрос об учёте пространственного заряда в пучке. Решение такой задачи на основе использования технологии параллельных вычислений CUDA от компании NVIDIA в программе CBDA предложено в данной работе.

Основная часть вычислений производится не на CPU, а на видеокарте с GPU. Такая видеокарта имеет от 100-200 потоковых процессоров. Использование для расчёта видеокарты с GPU вместо многопроцессорных машин является очень выгодным решением с точки зрения себестоимости.

Ключевые слова: ускорительная физика, вычисления на GPU, параллельные вычисления, пространственный заряд.

1. Постановка задачи

Поведение пучка заряженных частиц в современных ускорителях (рис. 1) чрезвычайно сложно, и применение компьютерного моделирования является неотъемлемой частью современной ускорительной физики. В настоящее время для расчёта динамики частиц необходимо учитывать потери частиц на структурных элементах установки и эффекты пространственного заряда пучка (рис. 2). Обе задачи важны, взаимосвязаны и требуют многократных и трудоёмких расчётов. Для минимизации потерь важен выбор оптимального расположения и конфигурации узлов установки, что, в свою очередь, требует многократных расчётов движения пучка частиц в установке. Учёт эффекта пространственного заряда накладывает определённые требования на начальное распределение пучка, которое должно иметь как можно больше пробных (макро) частиц для получения результатов моделирования с достаточной точностью. Из сказанного выше становится очевидным необходимость использования вычислительных машин большой производительности. Альтернативным и более доступным способом ускорения процесса счета является использование технологии параллельных вычислений, которая позволяет разработчикам создавать программное обеспечение для решения сложных вычислительных задач за меньшее время благодаря многоядерной вычислительной мощности графических процессоров. В данной работе на примере RIKEN AVF циклотрона [1] показано применение технологии CUDA [2] для расчёта динамики пучков заряженных частиц. Реализация данного подхода была произведена в программе CBDA [3].

Статья поступила в редакцию 12 сентября 2009 г.

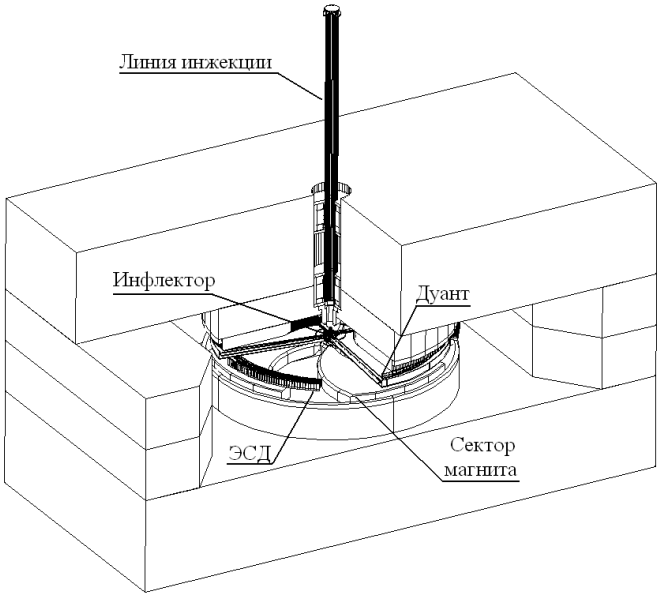

Рис. 1. **Общий вид циклотрона**

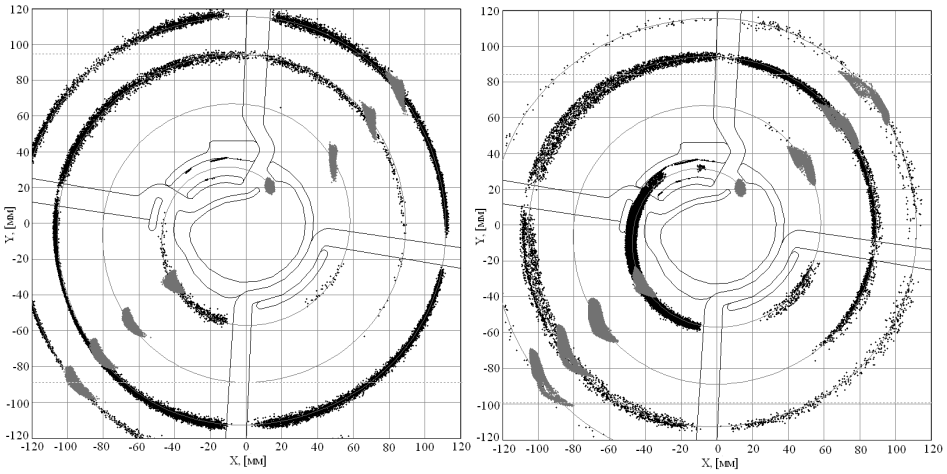

Рис. 2. **Эффект пространственного заряда вызван увеличением кулоновского взаимодействия между частицами в банче. В результате при увеличении тока возрастают потери частиц. Ток $I \sim$0 мА (рис. слева), $I = 4$ мА (рис. справа)**

2. Использование многоядерных процессоров (CPU)

В рассматриваемой здесь программе CBDA в качестве первого шага по ускорению вычислений было исследовано применение центрального многоядерного процессора CPU вместо обычно используемого центрального процессора. Инструментом для адаптации последовательного кода под использование многоядерных процессоров была выбрана свободно распространяемая технология OpenMP, с помощью которой можно легко и быстро создавать многопоточные приложения на C++ и, соответственно, существенно повысить производительность. Технология

OpenMP поддерживается средой разработки Visual C++, на которой написана программа CBDA. OpenMP, достаточно прост в использовании. Он состоит из набора прагм и функций из `omp.h`. Прагмы — это указатели компилятору разбивать код на блоки, которые будут выполняться параллельно. Если OpenMP выключен, компилятор проигнорирует прагмы, а код останется вполне работоспособным. Все это даёт возможность разрабатывать универсальный, переносимый код для реализации программ на различных архитектурах и системах. Адаптация программного кода CBDA с помощью OpenMP оказалась достаточно проста и не слишком трудоёмка. Исходный код подвергся минимальным изменениям, которые состояли в том, что все основные циклы программы предварялись прагмами OpenMP, указывающими участки кода, выполняемые параллельно.

В качестве примера были проведены расчёты для пучка, проходящего через спиральный инфлектор, с учётом пространственного заряда двумя способами: PP (частица на частицу) и PIC (частица в ячейке). Результатом данного этапа работы явилась табл. 1, иллюстрирующая выигрыш в скорости, достигнутый при расчёте динамики пучка для системы из 10000 макро-частиц на различных многоядерных системах с различными вариантами расчёта пространственного заряда пучка.

Таблица 1

Время счета программы с использованием технологии OpenMP

Метод	Без использования OpenMP	С использованием OpenMP	Используемая система
PP	4 ч. 53 мин.	2 ч. 34 мин.	AMD Turion 64×2, 1.6 ГГц
	4 ч. 38 мин.	1 ч. 25 мин.	Intel Core Quad 2.4 ГГц
PIC	∼11 мин.	∼6 мин.	AMD Turion 64×2, 1.6 ГГц
	∼7 мин.	∼2 мин.	Intel Core Quad 2.4 ГГц

Из таблицы видно, что полученное ускорение вычислений практически линейно зависит от количества используемых ядер. В идеальном случае коэффициент ускорения должен равняться количеству ядер, что в нашем случае не было достигнуто лишь из-за того, что оставались участки кода, выполняемые в последовательном режиме. Следует заметить, что при запуске программы, распараллеленной с помощью OpenMP, ресурсы процессора оказываются полностью задействованы. Таким образом, отсутствует возможность запуска других приложений для одновременного выполнения с основной программой.

Конечно, в данном подходе есть ускорение, но для описанных выше задач оно не столь существенно. К тому же для ускорения на порядок и более требуется создание целой фермы таких машин. Поэтому следующим шагом в процедуре ускорения вычислений явилось использование графической видеокарты с большим количеством потоковых процессоров.

3. Использование графических процессоров (GPU)

Для эффективного использования вычислительных возможностей графических процессоров необходимо было выявить в программе участки кода, требующие наибольших затрат времени, и адаптировать их с помощью технологии CUDA [2]. Тесты показали, что большая часть времени уходит на расчёт потерь на структурных элементах и учёт эффектов пространственного заряда. Отдельной и требующей больших затрат времени задачей явилась оптимизация центральной области циклотрона.

Основная работа велась по созданию аналогов трёх основных циклов программы: расчёта движения заряженных частиц в электромагнитных полях, учёта потерь макро-частиц на структурных элементах установки и расчёта собственного поля пространственного заряда пучка. Для каждой из этих задач была создана собственная функция ядра, имеющая свои особенности.

Основной цикл программы — расчёт движения макро-частиц в электромагнитном поле — устроен следующим образом: макро-частицы движутся независимо друг от друга в течение промежутка времени, равного шагу интегрирования, в конце которого происходит пересчёт действующих на них сил. Такое независимое движение макро-частиц позволяет проводить одновременный параллельный расчёт движения для каждой макро-частицы. В реализации на GPU внешний цикл по макро-частицам был заменён параллельным выполнением в разных потоках. Из-за ограничения на число потоков в одном блоке, а также для использования всех имеющихся мультипроцессоров GPU, потоки приходится разбивать на несколько блоков. Индекс, отвечающий за номер макро-частицы, вычислялся с учётом номера потока в блоке и номера блока. Особенностью данной функции ядра явилось то, что она изначально имела слишком много входных параметров ($\sim$ 300), что превышало объем передаваемых параметров на ядро. Выходом из этого явилось создание двух дополнительных массивов, локализованных в константной памяти и содержащих передаваемые параметры, остающиеся неизменными со временем. В дальнейшем в этой функции ядра пришлось максимально уменьшить количество операторов «if», «goto» и «for». Для этого пришлось «развернуть» некоторые имеющиеся в программе циклы. Полностью избавиться от таких операторов невозможно, так как на макро-частицу оказывают влияние различные магнитные и электрические поля. Каждое такое поле имеет свою область определения. В результате в одной и той же точке есть суперпозиция полей. С другой стороны, макро–частицы в пучке могут находиться в различных областях.

Процедура расчёта потерь макро-частиц на элементах установки имела одну важную особенность по сравнению с функцией пересчёта координат: кроме внешнего цикла по макро-частицам имелся вложенный цикл по треугольникам поверхности геометрии. В первом варианте функции ядра параллельным выполнением был заменён лишь внешний цикл по макро-частицам, а внутренний цикл по поверхностям был оставлен без изменения, и треугольники, составляющие поверхности, содержались в массиве, локализованном в глобальной памяти. Каждому параллельному потоку приходилось обращаться к глобальной памяти для считывания очередного треугольника поверхности. При такой простой реализации удалось добиться лишь четырёхкратного увеличения в скорости счета. Следующей итерацией было создание массивов, размещённых в разделяемой памяти (shared memory) блока и доступной только для потоков данного блока. В эти массивы записывались координаты треугольников поверхностей, причём для ускорения записи в массивы использовалась параллельная запись сразу несколькими потоками. Затем, для того, чтобы гарантированно были заполнены все элементы массивов координат, был поставлен оператор синхронизации, после которого шло выполнение цикла по треугольникам. Каждая нить, отвечающая за конкретную макро-частицу, работала с элементами из массивов разделяемой памяти. Из-за ограничения на разделяемую память в 16 Кб приходилось разбивать массив треугольников на несколько блоков, обрабатываемых последовательно. За счёт перехода от использования глобальной памяти к более быстрой разделяемой памяти коэффициент увеличения скорости счета вырос более чем на порядок.

Реализация параллельной версии процедуры учёта пространственного заряда включала в себя следующие этапы:

- распределение заряда пучка по узлам сетки;
- решение уравнения Пуассона;
- нахождение электрического поля в узлах сетки.

Так как процедура распределения заряда представляет собой цикл по макрочастицам, с независимым определением ячейки сетки, в которую попадает макрочастица и раздачей заряда в узлы данной ячейки, то естественно было написать функцию ядра, в которой отдельный поток отвечал за номер макро-частицы.

Единственной особенностью функции распределения заряда являлось то, что запись в узлы ячеек велась одновременно для всех макро-частиц, и из-за отсутствия алгоритма параллельной редукции происходил так называемый «конфликт записи». Для того, чтобы этого избежать, алгоритм записи заряда в узлы был вынесен из функции ядра на host. Далее, для того, чтобы можно было воспользоваться алгоритмом решения уравнения Пуассона с помощью быстрого преобразования Фурье (БПФ) [4], была реализована функция трёхмерного разложения вещественной

функции в ряд Фурье по синусам, основанная на последовательном трёхкратном повторении одномерного разложения по синусам вдоль трёх координатных осей. Отметим, что данная процедура отличается от библиотечной функции CUFFT [2], так как она оптимизирована на разложение только по базисным функциям $\sin(\frac{\pi \cdot n}{N})$.

Процедуры перехода от коэффициентов Фурье правой части уравнения Пуассона к коэффициентам левой части и вычисление градиента от полученных значений потенциала электрического поля в узлах сетки довольно просты, поэтому не составляет труда написание для них функция ядра.

4. Результаты

Программа CBDA [3] была полностью адаптирована под использование графических процессоров GeForce NVIDIA и Tesla C1060. Основные результаты по увеличению скорости счета для каждой из функций представлены в табл. 2. Данные приведены для системы из 100000 макро-частиц с учётом пространственного заряда на сетке $2^5 \times 2^5 \times 2^5$.

Таблица 2

Скорость счета для различных функций ядра программы в последовательном (CPU) и параллельном (GPU) коде

Функция	Время счета [мсек]		Увеличение скорости
	CPU 2.4 ГГц	GPU 8800GTX	[разы]
Track	486	30	16
Losses	6997	75	93
Rho	79	6	14
Poisson/FFT	35	3	13
E_SC	1.2	0.8	1.4
Total	7598	114	67

Функция Track производит расчёт траектории макро-частицы методом Рунге-Кутта 4-го порядка. Входными параметрами для неё являются карты магнитных и электрических полей, а также начальные координаты макро-частицы.

Функция Losses выполняет расчёт потерь макро-частиц на геометрической структуре. Геометрия задана посредством набора треугольников. На каждом шаге интегрирования траектории для каждой макро-частицы производится проверка: попала ли макро-частица в геометрическую структуру или нет. Входными параметрами функции Losses является геометрическая структура и положение макро-частиц в моменты времени t и $t + \triangle t$.

Функция Rho производит раздачу заряда в узлы сетки, на которой решается уравнение Пуассона. Входными параметрами являются координаты макро-частиц и сетка.

Функция Poisson/FFT производит решение уравнения Пуассона с помощью быстрого преобразования Фурье.

Функция E_SC вычисляет вектор напряжённости электрического поля в узлах сетки по известному скалярному потенциалу.

Видно, что максимальная разница во времени счета достигнута для функции учёта потерь, что понятно, так как наибольшее влияние на скорость расчётов оказывает уменьшение числа операций с медленной памятью (за счёт кэширования).

Для того, чтобы эффективно задействовать все скалярные процессоры видеокарты, необходимо в расчётах участие как можно большего количества макрочастиц. Естественно ожидать, что наибольшее ускорение будет достигнуто на системах с наибольшим количеством макро-частиц. В табл. 3–4 приведены данные о времени счета для полного расчёта динамики макро-частиц в циклотроне с

учётом потерь на геометрии установки в зависимости от количества используемых макро-частиц.

В результате использования GPU произошло ускорение вычислений от 70–90 раз. Этот разброс определяется конфигурацией расчётов, например, используется ли учёт пространственного заряда или потерь на геометрии, какова размерность сетки для БПФ, число макро-частиц, и т.д. Отметим, что использование таких карт, как Tesla S1070, может дать дополнительное ускорение от 3–4 раз, т.е. получить пиковое ускорение на уровне 270 раз.

Таблица 3
Время счета программы в зависимости от количества используемых макро-частиц

Число макро-частиц	Время счета		Ускорение
	CPU 2.4 ГГц	GPU 8800GTX	[разы]
1 000	3 мин. 19 с.	12 с.	17
10 000	34 мин. 14 с.	42 с.	49
100 000	5 ч. 41 мин.	6 мин.	56
1 000 000	2 дня 8ч. 53 мин.	1 ч.	60

Таблица 4
Время счета программы в зависимости от количества используемых макро-частиц

Число макро-частиц	Время счета		Ускорение
	CPU 2.5ГГц	GPU C 1060	[разы]
1 000	3 мин. 12 с.	11 с.	18
10 000	32 мин. 24 с.	27 с.	72
100 000	5 ч. 14 мин. 31 с.	3 мин. 34 с.	88
1 000 000	2 дня 4 ч. 25 мин.	34 мин. 29 с.	91

5. Заключение

Использование GPU даёт существенное увеличение производительности вычислений. Цена карточки GeForce 8800 GTX составляет ∼10 тыс. руб., Tesla C1060 ∼ 50 тыс. руб. GeForce больше ориентирован на игры, а Tesla — на вычисления. Поэтому для произведения многочасовых серьёзных расчётов Tesla предпочтительней GeForce, ибо цена ошибки в расчётах может оказаться фатальной.

Платой за скорость в данном случае является необходимость в достаточно аккуратном программировании при распараллеливании кода программы.

Полученная программа может быть также использована и в других физических приложениях, таких как задачи динамики пучка с предельно большим пространственным зарядом, учёт гало-пучка и т.д.

Литература

1. *Vorozhtsov S. et al.* Beam Simulations in Computer-Modelled 3D Fields for RIKEN AVF Cyclotron Upgrade.
2. CUDA Technology. — `http://www.nvidia.com,http://www.nvidia.ru`.

3. *Perepelkin E. E., Vorozhtsov S. B.* CBDA — Cyclotron Beam Dynamics Analysis code // In Proc.: The XXI Russian Accelerator Conference (RuPAC2008), Zvenigorod, Russia, September 28 – October 3. — 2008. — Pp. 40–42. — `http://cbda.jinr.ru/`.
4. *Рошаль А. С.* Быстрое преобразование Фурье в вычислительной физике // Изв. вузов. Радиофизика. — 1976. — Т. XIX, № 10. — С. 1425–1454.

UDC 519.712.45, 519.692, 519.694, 519.681.5

Beam Dynamic Calculation by NVIDIA CUDA Technology

E. E. Perepelkin [*], **V. L. Smirnov** [*], **S. B. Vorozhtsov** [†]

[*] *Veksler and Baldin Laboratory of High Energy Physics*
Joint Institute for Nuclear Research
141980, Dubna, Moscow region, Russia

[†] *Dzhelepov Laboratory of Nuclear Problems*
Joint Institute for Nuclear Research
141980, Dubna, Moscow region, Russia

The optimization of the central region structure of the cyclotron requires a tedious iterative procedure using a substantial amount of the particle tracking in the machine. The problem becomes even more complicated when the beam space charge effects should be taken into account. The solution of the problem was attempted via upgrade of the recently developed computer code CBDA based on the beam dynamics calculation by the NVIDIA CUDA parallel computing architecture. The main part of calculation was performed without application of the CPU, and only a video card with GPU of about 100-200 processors was used. An application of the video card with the GPU instead of multi-processor computer is very cost effective solution in this case.

Key words and phrases: accelerator physics, GPU computing, parallel calculations, space charge effect.