МОДЕЛИРОВАНИЕ ПЛАЗМЫ МЕТОДОМ ЧАСТИЦ В ЯЧЕЙКАХ С ИСПОЛЬЗОВАНИЕМ СОПРОЦЕССОРОВ INTEL XEON PHI

И.А. Сурмин[1], С.И. Бастраков[1], А.А. Гоносков[2], Е.С. Ефименко[2], И.Б. Мееров[1]

[1] Нижегородский государственный университет им. Н.И. Лобачевского
[2] Институт прикладной физики РАН

Введение

Одной из актуальных задач вычислительной физики является моделирование плазмы методом частиц в ячейках. Существуют задачи, требующие моделирования порядка 10^9 частиц и 10^8 узлов пространственной сетки и более. Для решения подобных задач необходимо создание специализированного программного обеспечения, достигающего высокой производительности и масштабируемости на современных суперкомпьютерах. Можно выделить несколько эффективных реализаций метода частиц в ячейках, которые ориентированы на традиционные и гетерогенные кластерные системы: OSIRIS [1], VLPL [2], VPIC [3], PIConGPU [4].

Программный комплекс PICADOR [5, 6] разрабатывается коллективом сотрудников ННГУ и ИПФ РАН с 2010 года. Программный комплекс ориентирован на решение больших задач в области моделирования лазерной плазмы на гетерогенных кластерных системах и демонстрирует сравнимую с аналогами производительность и масштабируемость. Отличительной особенностью PICADOR является ориентация на широкий класс вычислительных устройств: кластерные системы с многоядерными CPU и GPU на узле; в данной работе рассматривается использование сопроцессоров Intel Xeon Phi.

Сопроцессоры Xeon Phi, появившиеся в 2012 году, ориентированы на решение научных и инженерных задач. Особенностью сопроцессора является использование большого количества «легковесных» ядер x86-архитектуры. В отличие от GPU, поддерживаются традиционные языки и технологии параллельного программирования MPI, OpenMP и другие, что существенно упрощает портирование существующих приложений. Вместе с тем, в ряде задач эффективность такого портирования не всегда высока. В связи с этим выработка подходов к оптимизации кода для Xeon Phi представляет большой практический интерес. Основная сложность в достижении высокой производительности на Xeon Phi состоит в обеспечении хорошей масштабируемости на 60 ядер на общей памяти и эффективном использовании векторных операций.

В данной работе предлагается способ оптимизации метода частиц в ячейках для сопроцессоров Xeon Phi. Основное внимание уделяется вопросам улучшения масштабируемости при большом количестве ядер, обеспечения локальности доступа к памяти и эффективного использования векторных операций. Применение рассматриваемых оптимизаций также приводит к повышению производительности и на CPU, но в меньшей степени по сравнению с Xeon Phi. Предлагаемая реализация для Xeon Phi является одной из первых в мире.

Постановка задачи и Вычислительная схема

В данном разделе приводится краткое описание вычислительной схемы метода частиц в ячейках, подробное описание метода содержится в [7]. Под расчетной областью понимается область пространства, в которой происходит моделирование плазмы. Расчетная область имеет форму прямоугольного параллелепипеда со сторонами, параллельными осям координат. В каждой точке области заданы электрическое поле E и вектор магнитной индукции B. В методе частиц в ячейках плазма моделируется набором из N заряженных частиц, расположенных в расчетной области. Каждая частица определяется переменными импульсом p и положением r, а также постоянными массой m и зарядом q.

Вычислительная схема метода с основными уравнениями и схемой зависимости по данным приведена на рисунке 1, уравнения записаны в системе СГСЭ. Каждый шаг по времени состоит из 4 основных этапов: интегрирование уравнений поля, интерполяция поля и вычисление силы Лоренца, интегрирование уравнений движения частиц, взвешивание токов. Динамика электромагнитного поля определяется системой уравнений Максвелла, где c – скорость света, j – вектор плотности тока, индуцированного движением частиц. Координаты и скорость частиц меняются согласно второму закону Ньютона в релятивистской формулировке. Начальные условия состоят из значений электрического и магнитного полей во всех точках расчетной области, а также положений и скоростей частиц в начальный момент времени.

Метод частиц в ячейках оперирует двумя основными наборами данных: ансамбль заряженных частиц (электроны, ионы) и сеточные значения электромагнитного поля и плотности тока. На каждой итерации по времени по текущим данным частиц и поля вычисляется состояние системы в следующий момент времени.

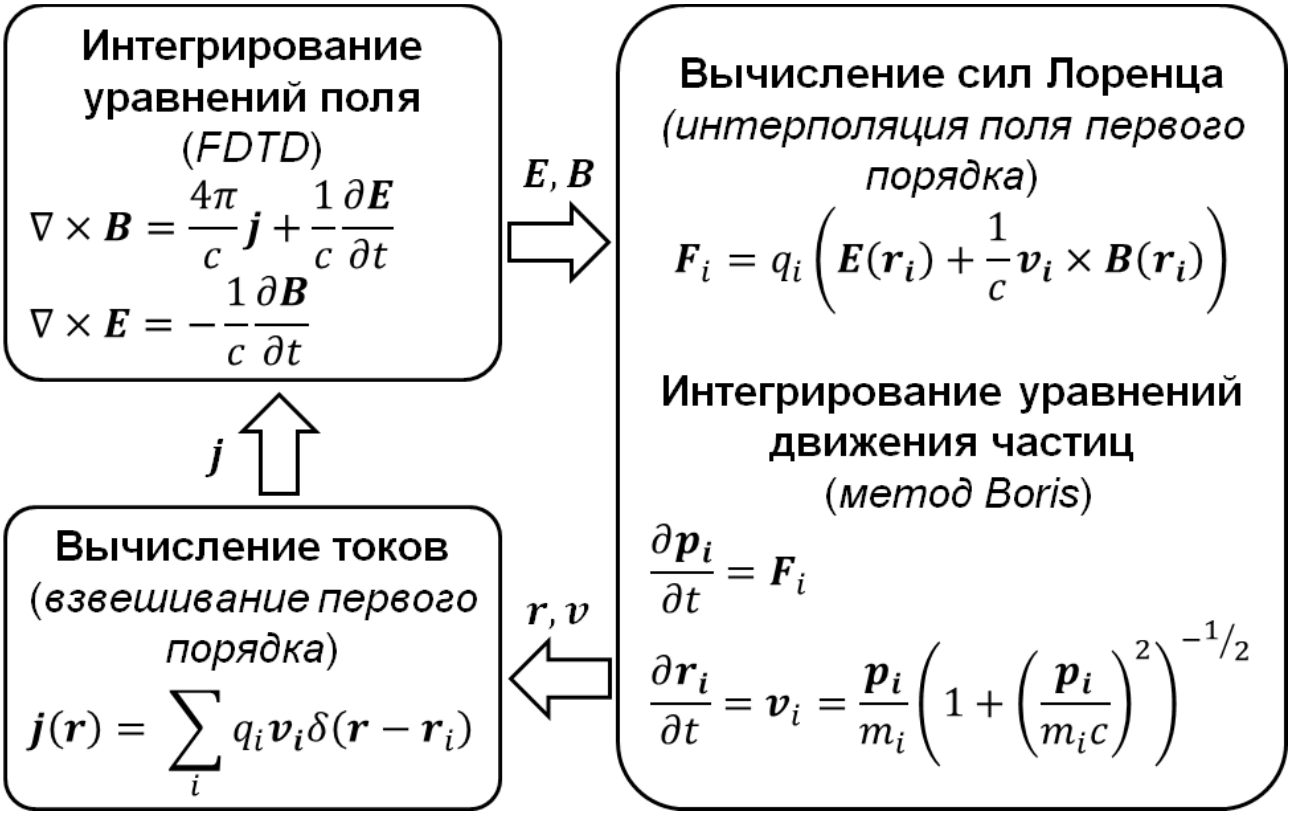

Рис. 1. Вычислительная схема метода частиц в ячейках. В скобках указаны численные схемы, реализованные в программном комплексе PICADOR.

Программный комплекс PICADOR

Программный комплекс PICADOR предназначен для трехмерного численного моделирования плазмы на гетерогенных кластерных системах. PICADOR основан на методе частиц в ячейках с многочисленными расширениями для повышения точности и учета дополнительных физических эффектов. В настоящее время поддерживаются конечно-разностные схемы FDTD и NDF для численного интегрирования уравнений поля, метод Бориса для интегрирования уравнений движения, формфакторы частиц первого и второго порядка, схема взвешивания Езиркепова, моделирование ионизации и рождения электрон-позитронных пар, бегущее окно, генерация лазерного импульса на границе и различные виды граничных условий, динамическая балансировка нагрузки.

Программный код PICADOR написан на языке программирования C++, поддерживается сборка под Windows и POSIX-совместимые системы. Работа на кластерных системах осуществляется через интерфейс передачи сообщений MPI. На уровне одного узла реализован параллелизм на общей памяти с помощью технологии OpenMP и использование GPU с помощью CUDA.

Сопроцессор Intel Xeon Phi

Сопроцессор Intel Xeon Phi реализован в виде отдельной платы, присоединяемой через слот PCI Express. Сопроцессор содержит 60 ядер (в некоторых модификациях 61 ядро), соединенных высокоскоростной кольцевой шиной. Каждое ядро является полнофункциональным, но облегченным по сравнению с обычными CPU, в частности, не поддерживаются предсказание ветвлений и внеочередное выполнение. Xeon Phi поддерживает выполнение векторных инструкций с 512-битными векторами по сравнению со 128- или 256-битными на CPU. Ядра Xeon Phi слабее, чем ядра CPU, но сочетание множества таких ядер с высокопроизводительной шиной данных предоставляет большую, по сравнению с традиционными процессорами, пиковую производительность и дает широкие возможности для использования параллелизма в научных и инженерных приложениях.

Xeon Phi поддерживает стандартные языки и средства параллельного программирования: языки программирования C, C++, Fortran, технологии MPI, OpenMP, Cilk Plus, OpenCL, библиотеки TBB, MKL, – что упрощает портирование существующих кодов и разработку новых. Разработка программ для Intel Xeon Phi предполагает выбор одного из режимов: нативного, offload или симметричного.

Режим offload предполагает использование Xeon Phi в режиме сопроцессора, то есть дополнительного вычислительного устройства, доступ к которому осуществляется с помощью специальных команд из кода, выполняемого на обычном центральном процессоре. При выполнении MPI-программы в этом режиме, ранги присваиваются только центральным процессорам.

В нативном режиме программа выполняется полностью на сопроцессорах, CPU в вычислениях не задействован. Между процессами, работающими на разных ускорителях, возможен обмен сообщениями посредством MPI. Для запуска программы в нативном режиме достаточно скомпилировать код для сопроцессора с помощью флага –mmic. Для эффективной работы программы в нативном режиме необходимо, чтобы доля последовательного кода была минимальна.

В симметричном режиме как центральный процессор, так и сопроцессор являются отдельными вычислительными узлами и могут взаимодействовать между собой посредством обмена MPI-сообщениями. В этом случае нужно иметь отдельные исполняемые файлы для CPU и Xeon Phi. При использовании данного режима необходимо обеспечить балансировку нагрузки между различными устройствами.

Анализ архитектуры Xeon Phi позволяет сделать следующие выводы:

1. Xeon Phi сочетает значительное число ядер с широкими векторными регистрами и высокопроизводительную шину данных, что дает широкие возможности для использования параллелизма. Для эффективного решения многих задач потребуется разработка новых вычислительных схем, ориентированных на существенное число параллельно работающих потоков и векторную обработку данных. Игнорирование этих аспектов может привести к катастрофически низкой производительности.

2. Ядра Xeon Phi имеют близкую к x86 архитектуру, что позволяет запускать на ускорителе обычный код без его переписывания с использованием новых технологий и средств программирования. Вместе с тем, эффективная работа не гарантирована. Для многих приложений может понадобиться оптимизация кода.

3. Подходы к оптимизации кода для CPU и Xeon Phi в целом похожи, следовательно, оптимизация производительности для Xeon Phi обычно будет приводить к выигрышу и на CPU, и наоборот. Это позволит разработчикам использовать весь имеющийся опыт распараллеливания и оптимизации.

Использование Сопроцессоров Intel Xeon Phi в коде PICADOR

Бенчмарк. Для анализа производительности кода PICADOR использовался бенчмарк с сеткой 40x40x40 ячеек и 50 частицами на ячейку, 1000 итераций по времени. Использовались схемы интерполяции и взвешивания первого порядка, двойная точность. Замерялось только время работы вычислительной части, которое складывалось из времени движения частиц, времени взвешивания токов и обновления полей. В качестве меры производительности использовалось количество наносекунд на обработку частицы на одной итерации, что является общепринятой метрикой для оценки реализаций метода частиц в ячейках.

Эксперименты проводились на узле кластера ННГУ «Лобачевский» с двумя процессорами Intel Xeon E5-2660 с частотой 2,2 ГГц (Sandy Bridge, 8 ядер, 20 MB кэш-памяти, поддержка AVX) и двумя сопроцессорами Intel Xeon Phi 5110P (60 ядер, 240 потоков, память 8 GB). Пиковая производительность используемого CPU составляет 140 GFlops, Xeon Phi – 1 TFlops.

Базовая версия. Операции типа Частица-Сетка, выполняемые при интерполяции поля и взвешивании токов, являются пространственно локальными. Хорошо известно, что для эффективной реализации метода частиц в ячейках необходимо использовать эту физическую локальность для достижения локальности шаблона доступа к памяти. В коде PICADOR для каждой ячейки хранится отдельный массив частиц, содержащихся в данной ячейке. Обход частиц происходит по ячейкам. При этом достигается локальность доступа к памяти как по частицам, так и по сеточным значениям.

Так как Xeon Phi является многоядерным устройством, для приемлемой производительности необходима масштабируемость кода на большое число потоков. Поэтому перед портированием кода на Xeon Phi были предприняты меры по улучшению масштабируемости на CPU. Кроме того, последовательные участки кода, быстро выполняющиеся на CPU, могут стать узким местом на Xeon Phi. К таким участкам относится, например, миграция частиц между ячейками.

При взвешивании токов первого порядка каждая частица вносит вклад в 8 ближайших сеточных значений тока, вклады всех частиц суммируются. Из-за необходимости суммирования данная операция не является независимой по данным. Ранее использовалась схема, при которой каждый поток хранил собственный набор сеточных значений тока и в процессе параллельного обхода частиц записывал в него частичные результаты , по окончании взвешивания вклады всех потоков суммировались. Данная схема плохо применима для Xeon Phi из-за гораздо большего по сравнению с CPU количества потоков. Была реализована новая параллельная схема взвешивания: параллельно обрабатываются ячейки, расположенные в шахматном порядке с шагом в две ячейки по каждой из трех осей. Таким образом, обход всех ячеек разбивается на 27 обходов в шахматном порядке. Распараллеливание производится на уровне внутреннего обхода, так как на этом уровне частицы разных ячеек не могут вносить вклад в ток одного и того же узла, что избавляет от необходимости синхронизации.

Аналогичный подход применяется при распараллеливании миграций частиц между ячейками. Для каждой ячейки вводится дополнительный буфер для вновь прибывающих частиц. После вычисления новых координат частицы производится проверка, покидает ли она текущую ячейку, в этом случае она записывается в буфер для новой ячейки. Благодаря тому, что за один шаг частица не может пролететь расстояние, большее размера ячейки, гонки данных невозможны и единственная синхронизация производится по окончании итерации внешнего обхода.

Для использования Xeon Phi был выбран нативный режим, так как доля последовательного кода в расчетной части невелика. Портирование базовой версии заключалось в перекомпиляции кода и используемых сторонних библиотек с опцией компилятора для использования Xeon Phi (-mmic).

На одном Xeon Phi время работы составляет 70 нс/частицу, на одном CPU достигается аналогичное время (рисунок 2). За счет использования эффективных схем распараллеливания код имеет хорошую масштабируемость с эффективностью 78% на 60 ядрах (рисунок 3).

Основными причинами плохой производительности на CPU и Xeon Phi являются неэффективное использование памяти и большая доля скалярного кода. Тем не менее, портирование без дополнительных усилий позволило получить такую же производительность на Xeon Phi, как и на CPU.

Рис. 2. Производительность базовой версии

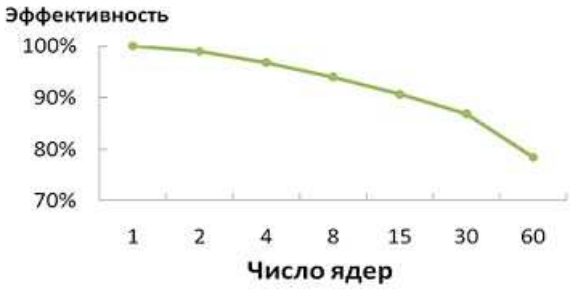

Рис. 3. Эффективность масштабируемости базовой версии на Xeon Phi

Оптимизация 1 – использование локальности данных. Способ хранения частиц оказывает существенное влияние на производительность и специфику производимых оптимизаций. Для достижения хорошей производительности как на CPU, так и на Xeon Phi код должен эффективно использовать кэш-память. С этой целью в PICADOR частицы хранятся по ячейкам. При интерполяции и взвешивании токов для каждой частицы в ячейке необходимо обращаться к одним и тем же данным, которые хранятся в глобальном массиве. Для интерполяции используются соседние сеточные значения по каждому измерению, но в памяти эти значения хранятся со значительным смещением. Поэтому для улучшения локальности имеет смысл сгруппировать используемые данные. Для реализации этой идеи на этапе интерполяции и взвешивания были введены локальные массивы. Перед началом обработки частиц в ячейке необходимые для интерполяции значения поля предзагружаются в локальные массивы 3x3x3. Далее интерполяция производится только с использованием локальных массивов. Аналогичный подход применяется для взвешивания: вклады частиц в токи узлов ячейки предварительно накапливаются в локальном массиве. По окончании обхода частиц в ячейке элементы локального массива добавляются в глобальный массив токов. Данная оптимизация позволила улучшить время работы на CPU в 3,7 раз, до 18,8 нс/частицу, и на Xeon Phi в 4,5 раз, до 15,7 нс/частицу (рисунок 4).

Рис. 4. Производительность оптимизированной версии с использованием локальности

Оптимизация 2 – векторизация кода. Векторизация заключается в одновременном выполнении одной и той же инструкции над несколькими операндами. Частным случаем векторизации является векторизация циклов, когда одновременно выполняются операции на соседних итерациях цикла. Некоторые компиляторы

способны автоматически векторизовывать циклы простой структуры. Векторизуемый цикл должен содержать известное количество итераций, не содержать условий и зависимостей по данным между итерациями, все вызовы функций должны быть встроены. Программист может оказывать влияние на процесс векторизации компилятора с помощью директив (например, #pragma vector always, #pragma ivdep для компилятора Intel). В коде PICADOR таким способом удалось векторизовать циклы обновления полей и движения частиц.

При хранении частиц в виде массива структур не получилось добиться ускорения от векторизации на Xeon Phi. Это связано с тем, что загрузка данных из разных мест памяти приводит к накладным расходам. Хранение данных в виде структуры массивов более предпочтительно при векторизации, так как позволяет добиться того, чтобы данные лежали последовательно и могли загружаться в векторный регистр одной инструкцией. Замена способа хранения на структуру массивов позволила эффективно векторизовать метод Бориса на Xeon Phi.

Рис. 5. Производительность оптимизированной версии с использованием локальности

Однако автоматическая векторизация далеко не всегда приводит к улучшению производительности. Попытка векторизации таким способом циклов интерполяции и взвешивания привела к замедлению кода в несколько раз. Это связано с тем, что используемые для интерполяции значения зависят от положения частицы внутри ячейки. Косвенная адресация по трем измерениям привела к генерированию компилятором большого числа инструкций scatter/gather. Накладные расходы на эти операции значительно превышают ускорение от векторизации. Поэтому предлагается векторизовать интерполяцию не по итерациям цикла обработки частиц, а по компонентам поля. Для этого компоненты помещаются в векторный регистр, в дальнейшем обрабатываются векторными инструкциями. Недостаток такого подхода состоит в том, что компилятор плохо векторизует блоки кода, поэтому приходится использовать более низкоуровневый подход – использование интринсиков. Это позволяет добиться ускорения от векторизации интерполяции и взвешивания на Xeon Phi.

За счет векторизации удалось улучшить время работы на CPU в 1,3 раза, до 14 нс/частицу, и на Xeon Phi – в 2 раза, до 7,7 нс/частицу (рисунок 5). Версия кода для Xeon Phi демонстрирует ускорение в 1,8 раз по сравнению с CPU.

Заключение

В работе рассказано об опыте оптимизации программной реализации метода частиц в ячейках для архитектуры Intel Xeon Phi. Разработка выполнена в контексте программного комплекса PICADOR и находится в стадии опытной эксплуатации. Выполненные оптимизации в основном затронули версию кода для систем с общей памятью.

Проведенная работа показала, что простая пересборка кода для Xeon Phi не всегда приводит к хорошим результатам, так как реализации, изначально ориентированные на 8-16 потоков и длину векторного регистра 128-256 бит, могут плохо масштабироваться на 60-240 потоков и векторные регистры размером 512 бит. Применяемые стандартные способы оптимизации обычно приводят к выигрышу как на CPU, так и на Xeon Phi, но степень влияния может существенно отличаться. В целом в работе достигнута эффективность 99% на 8 ядрах CPU и 78% на 60 ядрах Xeon Phi. В результате проделанных оптимизаций код на Xeon Phi стал обгонять оптимизированную и сопоставимую по скорости с лучшими аналогами CPU-версию почти в 2 раза.

Работа выполнена в лаборатории ННГУ-Intel «Информационные технологии» при поддержке гранта РФФИ № 14-07-31211.

ЛИТЕРАТУРА:

1. R.A. Fonseca, et al. OSIRIS: A Three-Dimensional, Fully Relativistic Particle in Cell Code for Modeling Plasma Based Accelerators // Lecture Notes in Computational Science 2331 (2002). – P. 342–351.
2. A. Pukhov. Three-Dimensional Electromagnetic Relativistic Particle-in-Cell code VLPL // Journal of Plasma Physics, 61 (1999). – P. 425–433.
3. K.J Blowers, B.J. Albright, L. Yin, W. Daughton, V. Roytershteyn, B. Bergen, T.J.T. Kwan. Advances in petascale kinetic plasma simulation with VPIC and Roadrunner // J. Phys.: Conf. Ser., 180 (2009). – P. 1–10.
4. H. Burau, R. Widera, W. Honig et al. PIConGPU: A Fully Relativistic Particle-in-Cell Code for a GPU Cluster // IEEE Transactions on Plasma Science, 33 (2010). – P. 2831–2839.
5. 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. 3 (2012). – P. 474-479.

6. S. Bastrakov, I. Meyerov, I. Surmin, E. Efimenko, A. Gonoskov, A. Malyshev, M. Shiryaev. Particle-in-Cell Plasma Simulation on CPUs, GPUs and Xeon Phi Coprocessors // J.M. Kunkel, T. Ludwig, and H.W. Meuer (Eds.): ISC 2014, LNCS 8488, Springer International Publishing Switzerland, 2014. – P. 513–514 (принято к печати).
7. Ч. Бэдсел, А. Ленгдон. Физика плазмы и численное моделирование: Пер. с англ. – М.: Энергоатомиздат, 1989. – 452 С.