Организация вычислений по FDTD-методу на графических процессорах суперкомпьютера "Сергей Королёв" с использованием языка MATLAB

Н.Д. Морунов[1], Д.Л. Головашкин[1,2]

[1]Самарский национальный исследовательский университет им. академика С.П. Королева, Московское шоссе 34А, Самара, Россия, 443086
[2]Институт систем обработки изображений РАН - филиал ФНИЦ «Кристаллография и фотоника» РАН, Молодогвардейская 151, Самара, Россия, 443001

Аннотация. В данной статье рассматривается проблема ограниченности объёма видеопамяти при организации параллельных вычислений по методу FDTD на графических процессорах суперкомпьютера «Сергей Королёв». В качестве решения предложен блочный алгоритм FDTD-метода и его реализация на языке MATLAB. В результате была решена проблема ограниченности графической памяти, максимально возможная дискретизация сетки при вычислениях на GPU была расширена с 50 млн. до 410 млн. узлов со средним ускорением в 15 раз для двумерного случая FDTD-метода.

1. Введение

Достаточно востребованным является численное решение уравнений Максвелла. Численный метод решения таких уравнений был назван методом конечных разностей по временной области или FDTD в 1980 году Аленом Тафловым [1]. В основу метода заложено использование алгоритма Йи, разработанного в 1966 году, который заключается в замене производных центрально разностными отношениями на сеточной области. Метод до сих пор находит своё применение в задачах электродинамики, оптики и нанофотоники. Однако он требует большого количества вычислительных ресурсов: времени и памяти.

Последнее десятилетие для решения подобного рода задач активно применяют параллельные вычисления на графических процессорах (GPGPU) [2]. Среди которых, CUDA самый известный и в ряде случаев наиболее эффективный инструмент параллельного программирования на GPU [3]. Для программирования на CUDA используется язык C. Однако, этот язык неудобен для реализации математических вычислений. Лучше всего для этой задачи подходит MATLAB, в котором для выполнения вычислений на GPU имеется специальный пакет – Parallel Computing Toolbox. К настоящему времени язык MATLAB хорошо зарекомендовал себя в качестве инструмента для параллельных вычислений на GPU [4]. Кроме того на языке MATLAB можно наиболее естественно выразить матричные вычисления.

Для эффективных вычислений сеточных уравнений одной высокопроизводительной техники мало, так как производительность вычислительных процессов порожденных алгоритмами, активно использующими память, ограничена пропускной способностью памяти [5]. Один из

вариантов решения этой проблемы – декомпозиция сеточной области. Именно такой подход применяется в блочных алгоритмах – за счет использования иерархии памяти и высокой локальности данных на каждом уровне уменьшаются простои процессора [6]. Ранее блочные алгоритмы FDTD-метода уже реализовывались на CUDA для вычислений на кластере с использованием графических процессоров [7], но нет реализации блочного алгоритма FDTD-метода на MATLAB.

Итак, целью работы является исследование эффективности блочного алгоритма FDTD-метода и его реализации на языке MATLAB на суперкомпьютере «Сергей Королёв».

2. Эксперимент

Был поставлен эксперимент, целью которого было исследовать эффективность блочного алгоритма двумерного случая FDTD-метода путем исследования реализации этого алгоритма на языке MATLAB, запущенной на вычислительном узле суперкомпьютера «Сергей Королёв».

На вычислительном узле запускался ряд профилирующих скриптов в MATLAB R2012b: измерение времени выполнения вычислительного процесса на CPU и GPU; оценка оптимального размера блока; измерение времени выполнения вычислительного процесса для одномерной и двумерной декомпозиции. Профилирующие скрипты в свою очередь вызывали функции, производящие непосредственные вычисления: на CPU, на GPU без декомпозиции, а также на GPU при использовании блочной декомпозиции двух видов. В качестве примера вычислительной задачи было рассмотрено распространение оптического излучения от электрического диполя, задававшегося методом жесткого источника. На границах прямоугольной вычислительной области были установлены поглощающие слои – CPML с толщиной слоя в 10 узлов. Дискретизация вычислительной области изменялась в зависимости от входных значений функции. Декомпозиция на блоки в MATLAB выполнялась с предварительным обрамлением массивов задействованных в вычислениях внутри блока. Подробно данный подход описан в одной из статей авторов [8].

Программа запускалась под операционной системой Red Hat Enterprise Linux 6 на вычислительном узле кластера «Сергей Королев» n250 с двумя восьмиядерными центральными процессорами Intel Xeon X5670 с тактовой частотой 2,93 ГГц и кэшем 12Мб. Также на узле было установлено 48 Гб оперативной памяти, и две графические карты Nvidia Tesla 2070, с объёмом внутренней оперативной памяти 6 Гб и числом CUDA-совместимых шейдерных процессоров – 448 с тактовой частотой 575 МГц. Однако в программе использовалась только одна графическая карта.

Для того чтобы исследовать эффективность реализации рассмотренного блочного алгоритма был построен график зависимости производительности вычислений (млн. узлов в сек.) от количества узлов в одном временном слое. Анализ производительности вычислений на GPU без применения блочности, показал, что производительность снижается после достижения некоторого значения количества узлов. Это значение связано с особенностями использования памяти в MATLAB и зависит от конкретного алгоритма и системы, на которой производятся вычисления. Эти значения были экспериментально определены для разных вариантов блочности: для одномерной и двумерной декомпозиции оптимальный размер блока составил 24 млн. узлов.

В функциях в качестве параметров задаются количество узлов в одном временном слое по оси X (N), по оси Y (M), количество временных слоев (n), высота блока (T) и размер основания блока (D). Общую модель вычислительного процесса можно представить в следующем виде:

$$t = n\sum_{i=1}^{B} S_i \left(t_{\text{выч.}} + \frac{t_{\text{ком.}}}{T}\right), \quad (1)$$

где S_i – количество узлов в одном временном слое занятое i-ым блоком (проекция блока); B – общее количество блоков; $t_{\text{выч.}}$ – время, затрачиваемое на вычисления одним узлом; $t_{\text{ком.}}$ – время, затрачиваемое на коммуникации одним узлом.

Общее количество узлов, занимаемое всеми блоками, отличается для разных размерностей декомпозиции:

$$\sum_{i=1}^{B} S_i^{1D} = NM\left(1 + \frac{T(B-1)}{N}\right) \approx NM\left(1 + \frac{T}{2D}\right), \quad (2)$$

$$\sum_{i=}^{B} S_i^{2D} \approx 2NM\left(1+\frac{T)}{2D}\right).$$

Очевидно, что эффективность двумерной декомпозиции как минимум в два раза ниже, чем у одномерной декомпозиции. Это связано с большим количеством холостых операций, которые неизбежно добавляются при обрамлении блоков. Тем не менее, у двумерной декомпозиции есть свои преимущества. Так, например, при использовании в расчётах достаточно большой сеточной области (с формой близкой к квадрату) в алгоритме с одномерной декомпозицией возникнет ситуация, когда высоту блока нельзя сделать больше одного. В случае с двумерной декомпозицией такой проблемы не возникает.

3. Результаты

В результате работы профилирующих скриптов были получены: график зависимости производительности вычислений от количества узлов для двумерного случая FDTD и график зависимости ускорения от количества узлов в одном временном слое.

На рисунке 1 можно увидеть четыре графика зависимости производительности вычислений от количества узлов в одном временном слое: для CPU, для GPU без декомпозиции и для GPU с одномерной и двумерной декомпозицией. В графиках для центрального и графического процессоров можно выделить три участка: первый - соответствующий росту производительности (CPU – производительность доходит до 12 млн.узлов/с, GPU – до 290 млн.узлов/с); второй - небольшое падение производительности и удержание её на одном уровне (CPU – падает до 10млн.узлов/с, GPU – падает до 270 млн.узлов/с); третий – стремительное падение производительности вплоть до обрыва графика из-за нехватки памяти(CPU – обрыв происходит на 400 млн. узлах, GPU – на 50 млн. узлах). Последние два участка связаны с особенностями использования памяти в MATLAB. Графики для графического процессора с декомпозицией строились по точкам соответствующим наивысшей производительности для разного количества блоков. На графиках можно выделить два участка: на первом производительность держится на одном уровне (производительность одномерной декомпозиции – удерживается на 190 млн. узлов/с, двумерной декомпозиции – на 60 млн. узлов/с), на втором – падает (производительность одномерной декомпозицией падает, начиная с 410 млн. узлов; двумерной – с 450 млн. узлов).

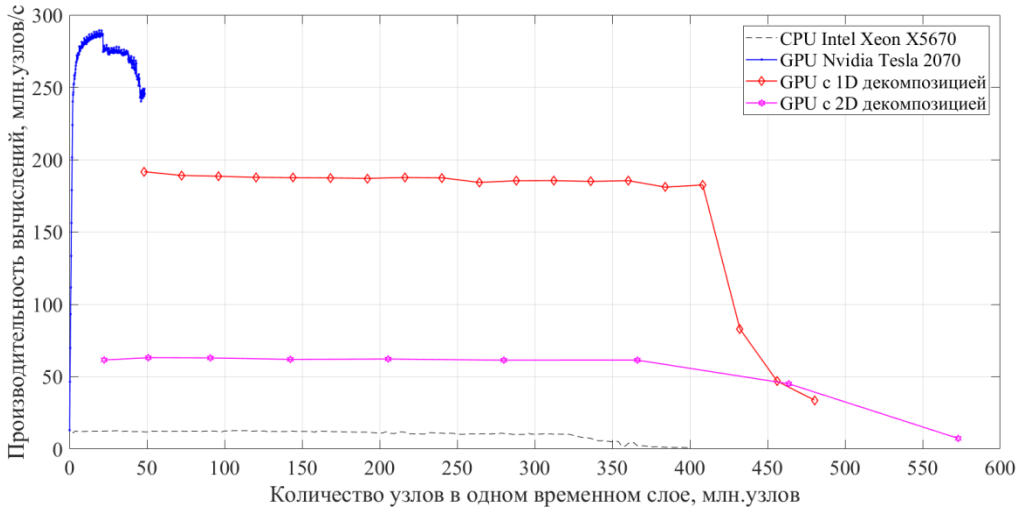

Рисунок 1. Графики зависимости производительности вычислений от количества узлов (млн. узлов) для двумерного случая FDTD.

На рисунке 2 представлены графики зависимости ускорения от количества узлов в одном временном слое: для трёх реализаций на GPU. Значение производительности справа на графиках резко возрастает. Это связано с тем, что с помощью блочного алгоритма удалось сэкономить оперативную память. При вычислениях на CPU память расходовалась как на хранение промежуточных результатов, так и на вычисления. В случае с декомпозицией

оперативная память стала использоваться только для хранения результатов. Таким образом ускорение на GPU без декомпозиции колеблется от 20 до 23; с одномерной декомпозицией – ускорение в 15 раз; с двумерной декомпозицией – в 5 раз.

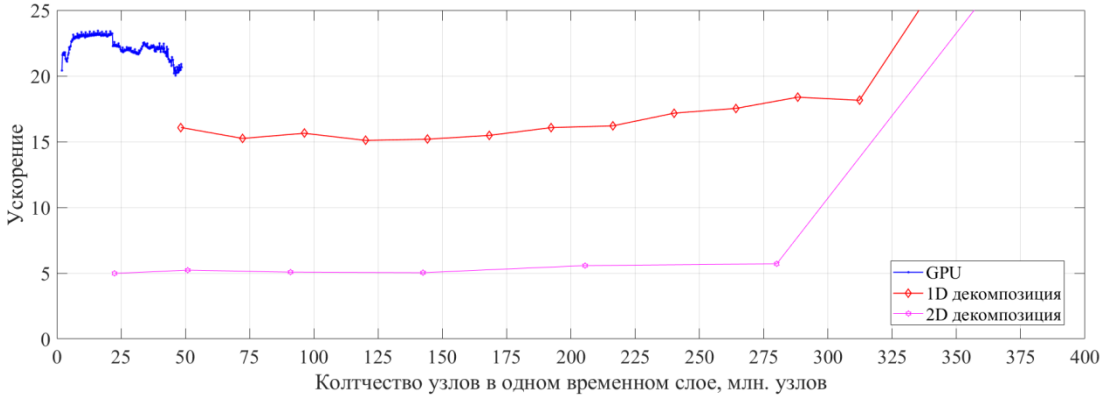

Рисунок 2. График зависимости ускорения от количества узлов в одном временном слое.

Сравнивая результаты двумерной декомпозиции с результатами одномерной декомпозиции можно сделать вывод, что эффективность реализации двумерной декомпозиции в 3 раза меньше чем у одномерной. Такое сильное падение эффективности связано не только с большим количеством холостых операций (падение эффективности в 2 раза), но и сведением блоков к универсальному виду, из-за чего блоки в двумерной декомпозиции покрыты множеством веток условий (которые необходимы для учета поглощающих слоев), и как видно вычисляются в полтора раза медленнее, чем блоки в одномерной декомпозиции. Для того, чтобы повысить эффективность двумерной декомпозиции нужно каким-то образом убрать из расчётов холостые операции. Узлы вносящие наибольший вклад в дополнительно затрачиваемое время находятся у самого основания блока, они формируют из ромба квадрат, тем самым удваивая время вычислений.

4. Заключение

Блочный алгоритм FDTD-метода и его реализация на языке MATLAB на суперкомпьютере «Сергей Королёв» исследованы. Рассмотрены блочные алгоритмы с одномерной и двумерной декомпозицией с реализацией на языке MATLAB. В результате было установлено, что эффективность одномерной декомпозиции в 3 раза выше эффективности двумерной декомпозиции, средняя производительность реализации блочного алгоритма FDTD-метода с одномерной декомпозицией – 190 млн.узлов/с, что на рассматриваемой системе соответствует ускорению в 15 раз. Максимально возможная дискретизация сетки при вычислениях на GPU была расширена с 50 млн. до 410 млн. узлов. Кроме того, расширенная дискретизация сетки превысила максимальную дискретизацию при вычислении на CPU – 350 млн. узлов. В дальнейшем планируется исследовать производительность вычислений при использовании нескольких графических процессоров и нескольких узлов с большим объёмом оперативной памяти.

5. Благодарности

Работа выполнена при поддержке гранта РФФИ 19-07-00423 А.

6. Литература

[1] Taflove, A. Computational Electrodynamics: the finite-difference time-domain method / A. Taflove. – London: Artech House, 1995. – 866 p.
[2] Unnc, M. GPGPU-FDTD method for 2-dimensional electromagnetic field simulation and its estimation / M. Unnc, Y. Inoue, H. Asar // IEEE 18th Conference on Electrical Performance of Electronic Packaging and Systems. – Portland, 2009. – P. 239-242.

[3] Micikevicius, P. 3D finite difference computation on GPUs using CUDA / P. Micikevicius // Proceedings of 2Nd Workshop on General Purpose Processing on Graphics Processing Units, ser. GPGPU-2, 2009. – P. 79-84.
[4] Diner, J.E. FDTD Acceleration using MATLAB Parallel Computing Toolbox and GPU / J.E. Dinier, A.Z. Elsherbeni // Aces Journal. – 2017. – Vol. 32(4). – P. 283-288.
[5] Williams, S. Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures / S. Williams, A. Waterman, D. Patterson // Communications of the ACM. – 2009. – Vol. 52. – P. 65-76.
[6] Левченко, В.Д. Асинхронные параллельные алгоритмы как способ достижения эффективности вычислений // Инф. технологии и выч. системы. – 2005. – № 1. – С. 68-87.
[7] Закиров, А.В. Алгоритм DiamondTorre и высокопроизводительная реализация FDTD метода для суперкомпьютеров с графическими ускорителями / А.В. Закиров, В.Д. Левченко, А.Ю. Перепёлкина, Я. Земпо // Суперкомпьютерные дни в России, 2016. – С. 80-94.
[8] Морунов, Н.Д. Особенности построения блочных алгоритмов FDTD-мнтода при организации вычислений на графическом процессоре с использованием языка MATLAB / Н.Д. Морунов, Д. Л. Головашкин // Компьютерная оптика. – 2019. – Т.43, №4. – С. 670-675. DOI: 10.18287/2412-6179-2019-43-4-671-676.

Performing of the FDTD method calculations on Supercomputer "Sergey Korolev" GPUs via MATLAB language

N.D. Morunov[1], D.L. Golovashkin[1,2]

[1]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086
[2]Image Processing Systems Institute of RAS - Branch of the FSRC "Crystallography and Photonics" RAS, Molodogvardejskaya street 151, Samara, Russia, 443001

Abstract. The problem of limited video memory when organizing parallel computing using the FDTD method on cluster Sergey Korolev GPUs was considered in this article. As a solution, a block algorithm of the FDTD method and its implementation in the MATLAB language are proposed. As a result, the problem of limited graphics memory was solved, the maximum possible discretization of the grid in calculations on the GPU was expanded from 50 mln to 410 mln nodes with average speedup 15 times for 2D FDTD method.