Реализация разностного решения уравнения Максвелла на языке MATLAB с использованием графического процессора

Н.Д. Морунов[1]

[1]Самарский национальный исследовательский университет им. академика С.П. Королева, Московское шоссе 34А, Самара, Россия, 443086

Аннотация. В работе рассмотрен метод решения уравнений Максвелла FDTD и методы задания плоской волны для такого решения. Также были рассмотрены особенности реализации вычислений в MATLAB с использованием графического процессора. Предлагается сравнительный анализ разностного решения уравнений Максвелла на языке MATLAB на GPU для разных способов задания плоской волны при использовании недорогой графической карты. В результате установлено, что на языке MATLAB с использованием пользовательской видеокарты NVIDIA GTX 960m можно добиться ускорения вычислений до 60 раз.

1. Введение

Численные методы решения уравнения Максвелла требуют большого количества времени на решение простых задач электродинамики, не говоря уже о сложных. Самым эффективным способом уменьшения затрат по времени является параллельное программирование на графических процессорах или как его еще называют - неспециализированные вычисления на графическом процессоре (GPGPU). Известными инструментами GPGPU программирования являются CUDA, OpenCL и вычислительные шейдеры. Из них на сегодняшний день CUDA является самым распространённым и в ряде случаев самым эффективным инструментом.

Для того чтобы использовать в своих вычислениях CUDA необходимо знать языки программирования, в частности C. Однако не все ученые владеют популярным языком программирования C. Синтаксис языка MATLAB очень простой, легко поддается освоению и идеально подходит для таких учёных. В MATLAB есть специальный пакет для параллельных вычислений Parallel Computing Toolbox. С помощью этого пакета, используя несложный синтаксис, можно легко реализовать вычисления на многоядерных процессорах, на кластерах и на GPU.

В книге A. Elsherbeni [4] очень подробно изложен FDTD метод с реализацией на языке MATLAB, но автор не описывает реализации параллельных вычислений на графическом процессоре на том же языке MATLAB. Это было связано с тем, что на время написания книги в MATLAB не было возможности производить вычисления на графическом процессоре. Этот пробел восполняется этой статьей.

В настоящей статье более подробно рассматриваются способы задания плоской волны на языке MATLAB с использованием GPU недорогой графической карты. Для вычислений использовалась пользовательская видеокарта NVIDIA GTX 960m. Похожими характеристиками

обладают карты GTX серий 600-700 для ПК и 800m-900m серий для ноутбуков, цена такой видеокарты невысока, порядка 200$ ~10 000 р.

2. Организация параллельных вычислений на MATLAB с использованием GPU

В дополнительных материалах к книге A. Taflove [3] есть скрипты, написанные его ассистенткой S. Hagness на языке MATLAB. Их всего три: для одномерного, двумерного и трёхмерного случаев метода FDTD. Вычисления в скриптах выполняются на центральном процессоре.

Рассмотрим алгоритм, реализованный в программе для трёхмерного случая. На каждом шаге по времени записываются вычисленные значения компоненты Ez электрического поля в матрицу анимации, затем с помощью функции movie проигрывается анимация.

Ниже представлен алгоритм программы, представленный на языке MATLAB:

```
ex=zeros(ie,jb,kb);
…
hz=zeros(ie,je,kb);

for n=1:nmax
ex(1:ie,2:je,2:ke)=ca*ex(1:ie,2:je,2:ke)+...
            cb*(hz(1:ie,2:je,2:ke)-hz(1:ie,1:je-1,2:ke)+...
              hy(1:ie,2:je,1:ke-1)-hy(1:ie,2:je,2:ke));
…
ez(is,js,1:ke)=ez(is,js,1:ke)+...
        srcconst*(n-ndelay)*exp(-((n-ndelay)^2/tau^2));
…
hz(1:ie,1:je,2:ke)=hz(1:ie,1:je,2:ke)+...
            db*(ex(1:ie,2:jb,2:ke)-ex(1:ie,1:je,2:ke)+...
              ey(1:ie,1:je,2:ke)-ey(2:ib,1:je,2:ke));
end;
```

Листинг 1. Алгоритм для трехмерного случая FDTD на языке MATLAB.

В пакете MATLAB Parallel Computing Toolbox существует три способа реализации параллельных вычислений на GPU: с помощью нового типа переменной gpuArray и различных функций определенных для этого типа; с помощью функции arrayfun, которая позволяет задавать свою функцию по обработке GPU массивов; посредством написания своего CUDA-ядра и использования его в расчётах.

Первым делом для того, чтобы переделать код MATLAB написанный для центрального процессора в код, написанный для графического процессора, необходимо все большие массивы перевести в тип gpuArray, сделать это можно как через явное приведение типа к gpuArray, так и заданием массива напрямую в памяти графической карты (листинг 2). Для перевода массива из внутренней памяти графического процессора в оперативную память можно воспользоваться функцией gather. Однако для вывода массива в виде графика, такой операции не потребуется. При рисовании графиков, набор вершин, задающих геометрические примитивы, должен находиться в памяти видеокарты, поэтому производить обмен данными не имеет смысла.

```
ex= zeros(ie,jb,kb,'gpuArray');
…
hz= zeros(ie,je,kb,'gpuArray');
```

Листинг 2. Объявление массивов для электрического и магнитного полей напрямую в графической памяти.

Функция arrayfun идеально подходит для выполнения операций над матрицами разной размерности, например, такой прием используется при оптимизации расчёта CPML (листинг 4). Также arrayfun позволяет выполнять над матрицами операции, заданные своими функциями. Однако последовательность обработки элементов матрицы такой функцией менять нельзя.

```
ex(il:ir-1,jl:jr,kl) = arrayfun (@plus,ex(il:ir-1,jl:jr,kl), cb*hy_1D(il:ir-1));
ex(il:ir-1,jl:jr,kr+1) = arrayfun (@minus,ex(il:ir-1,jl:jr,kr+1), cb*hy_1D(il:ir-1));
```

Листинг 3. Обновление электрического поля на границе полного поля (TF) и рассеянного поля (SF) с помощью arrayfun.

```
Psi_eyx_xn(1:L,1:je,2:ke) = arrayfun (@times,Psi_eyx_xn(1:L,1:je,2:ke),cpml_b_en(1:L))+...
   arrayfun (@times,hz(1:L,1:je,2:ke)-hz(2:L+1,1:je,2:ke),cpml_a_en(1:L));
ey(2:L+1,1:je,2:ke) = ey(2:L+1,1:je,2:ke) + cb_psi(1:L,1:je,2:ke).*Psi_eyx_xn(1:L,1:je,2:ke);
Psi_ezx_xn(1:L,2:je,1:ke) = arrayfun (@times,Psi_ezx_xn(1:L,2:je,1:ke),cpml_b_en(1:L))+...
   arrayfun (@times,hy(2:L+1,2:je,1:ke)-hy(1:L,2:je,1:ke),cpml_a_en(1:L));
ez(2:L+1,2:je,1:ke) = ez(2:L+1,2:je,1:ke) + cb_psi(1:L,2:je,1:ke).*Psi_ezx_xn(1:L,2:je,1:ke);
```

Листинг 4. Изменение Ex компоненты CPML с помощью arrayfun.

MATLAB поддерживает ряд функций по работе с CUDA напрямую: загрузка ядра, изменение параметров ядра и т.д. Этот метод подразумевает написание ядра на языке C (фактически процедуры, которая будет выполняться на графическом процессоре) и подключение её к приложению MATLAB. Далее необходимо провести настройку загруженного ядра: указать количество максимально возможных параллельных потоков; указать количество блоков, и выполнить процедуру ядра.

Третий метод хоть и является лучшим, но задачей статьи была реализация разностного решения средствами MATLAB, а для подключения CUDA ядер, требуется сперва их написать на языке C.

MATLAB больше подходит для векторных вычислений, поэтому весь код следует первоначально векторизовать. При работе с Parallel Computing Toolbox была замечена такая особенность: в случае если размер вектора небольшой - порядка 10 элементов, то такой вектор не стоит переносить в память графической карты, так как в ряде случаев это только замедлит работу. Более того если на протяжении большого числа итераций в алгоритме происходят обращения лишь к одному элементу вектора, и его значения в течение этих итераций не изменяется, стоит заранее исключить это значение из вектора и перенести его в локальную рабочую область.

В соответствии со всеми выше перечисленными особенностями пакета Parallel Computing Toolbox алгоритм S. Hagness (листинг 1) был адаптирован для выполнения на GPU.

3. Экспериментальное исследование реализации FDTD на MATLAB

В эксперименте использовалась видеокарта NVIDIA GTX 960m с объёмом внутренней оперативной памяти - 2Gb и числом CUDA-совместимых шейдерных процессоров – 640. А также 4-х ядерный центральный процессор Intel Core i5-6300HQ с базовой тактовой частотой 2.3 ГГц и максимальной частотой 3.2 ГГц, с объёмом кэш памяти 6Мб.

Следующим шагом был разработан отдельный скрипт для профилирования работы вычислительного процесса на разных процессорах. На основе рассмотренных ранее алгоритмов для выполнения расчётов на CPU и на GPU были написаны функции. На вход такой функции подаётся размер сеточной области и количество итераций, а на выходе получается время выполнения расчётов. Основываясь на этом, представим код профилировщика в более простом виде:

```
c_min = 30; c_max = 250; dc = 10; nmax=6;
T = (c_max-c_min)/dc+1;
cells = c_min:dc:c_max;
for i = 1:T
    cpu_time(i) = fdtd3D_cpu(cells(i), nmax);
    gpu_time(i) = fdtd3D_gpu(cells(i), nmax);
end;
speedup = cpu_time./gpu_time;
cells_axis = cells.^3 / 10^6;
```

```
cpu_perfomance_axis = cells.^3 ./ cpu_time / 10^6;
gpu_perfomance_axis = cells.^3 ./ gpu_time / 10^6;
figure;
subplot(3,1,1);
plot(cells_axis, cpu_perfomance_axis);
subplot(3,1,2);
plot(cells_axis, gpu_perfomance_axis);
subplot(3,1,3);
plot(cells_axis, speedup);
```

Листинг 5. Код профилировщика на языке MATLAB.

Были рассмотрены 4 комбинации начальных и граничных условий и под каждую из них были реализованы 2 функции fdtd3D_cpu и fdtd3D_gpu (Листинг 5). В качестве начальных условий был задан источник, генерирующий плоскую волну двумя способами TFSF и TFRF [2]. В двух случаях для способа TFRF задаётся универсальная сеточная область, ограниченная по оси распространения плоской волны идеальными электрическими стенками (PEC), а для остальных направлений заданы цикличные граничные условия. В двух других случаях для способа TFSF в качестве граничных условий задавалась идеальная электрическая стенка (PEC). Также для каждого способа задания волны рассматривался случай с заданными на границах вычислительной области идеально поглощающими слоями CPML [4].

В результате выполнения профилирующего скрипта были получены графики зависимости производительности вычислений на каждом процессоре, измеряемой в миллионах обработанных ячеек за секунду, от количества ячеек. На рисунке 1 представлены характеристики вычислительного процесса, выполняющего расчёт распространения плоской волны в вычислительной области, ограниченной идеальной электрической стенкой без использования CPML, и на рисунке 2 с использованием CPML. В итоге был рассчитан график зависимости ускорения от количества ячеек для каждого случая представленный на рисунке 3.

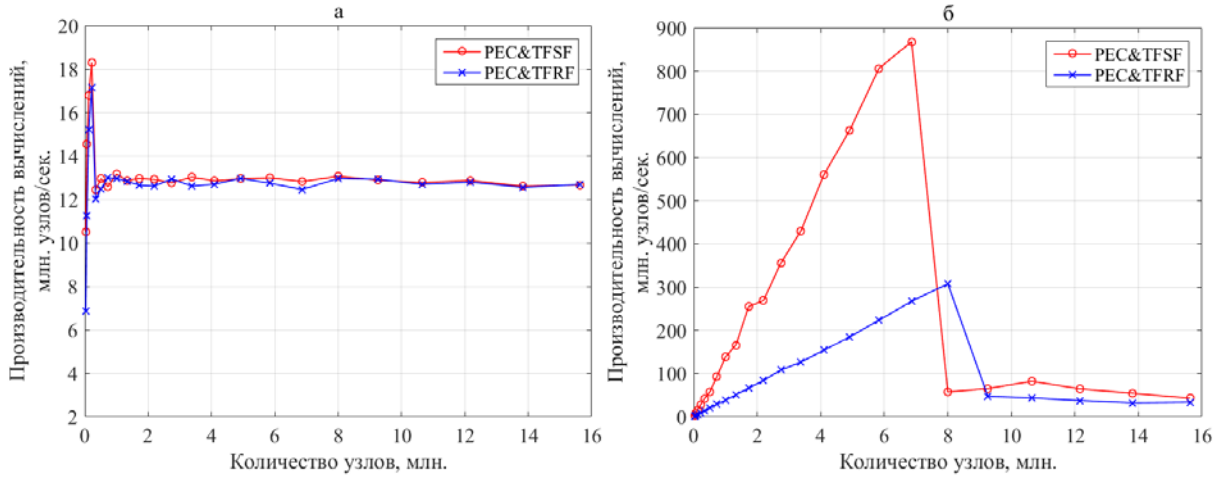

Рисунок 1. Графики зависимости производительности вычислений (млн. узлов/сек.) от количества обработанных узлов(млн.) в случае с идеальной электрической стенкой: а) на CPU; б) на GPU.

На графиках (рисунки 1б и 2б) после некоторого значения размера сетки виден резкий спад производительности вычислений, выполненных на GPU. Это связано с недостаточным объёмом видео памяти на графической карте, и особенностями самого пакета Parallel Computing Toolbox. На графиках зависимости производительности вычислений, выполненных на CPU, от количества узлов (рисунки 1а и 2а) в самом начале виден резкий скачок. Такое происходит из-за того, что объём данных необходимый для вычислительной задачи помещается в кэш центрального процессора и скорость доступа к памяти высокая. С ростом объёма данных,

необходимых для вычислительной задачи, увеличивается количество кэш-промахов, в следствие чего скорость обращения к памяти уменьшается, как и производительность вычислений.

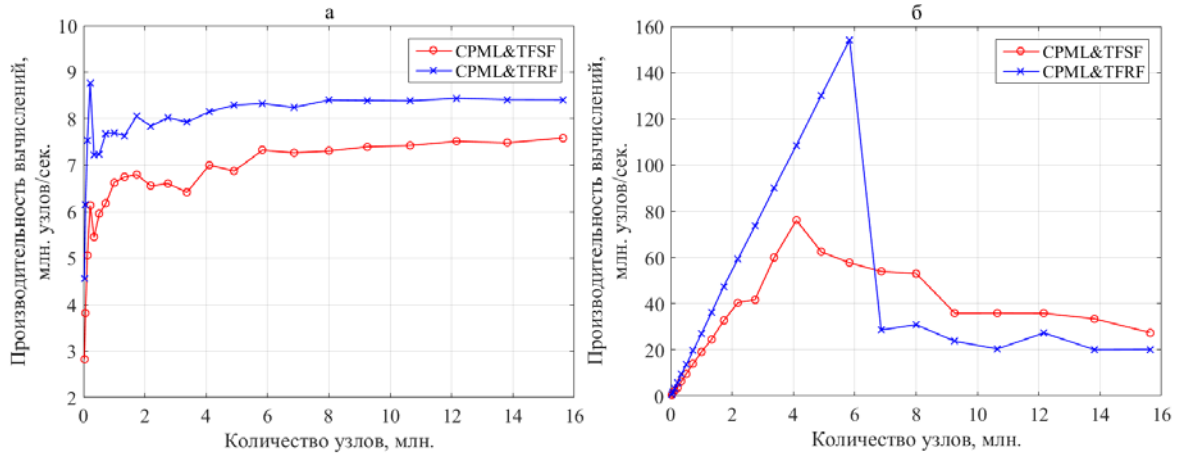

Рисунок 2. Графики зависимости производительности вычислений (млн. узлов/сек.) от количества обработанных узлов(млн.) в случае с CPML: а) на CPU; б) на GPU.

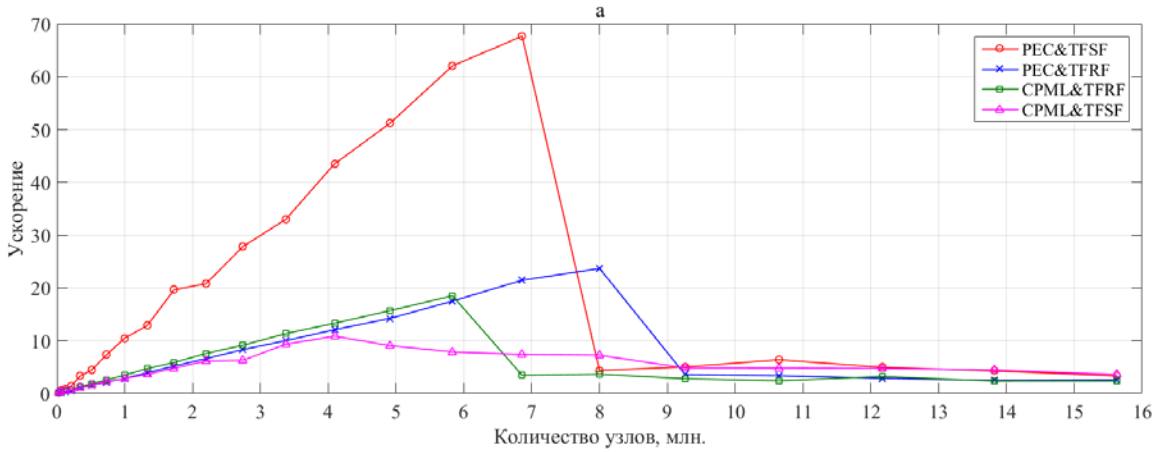

Рисунок 3. Графики зависимости ускорения от количества обработанных узлов (млн).

В обоих случаях с TFRF (рисунки 1 и 2) увеличение производительности вычислений при использовании GPU приемлемое, однако в случаях с TFSF наблюдается спад производительности при использовании CPML. Такое поведение можно объяснить следующим: граничные условия в TFSF (идеальная электрическая стенка) задаются легко без добавления лишних векторов, однако CPML накладывается на каждую из 6-ти граней, в TFRF задаются периодические граничные условия, из-за чего увеличивается количество векторов, а CPML накладывается только на 2 грани. Использование в расчётах CPML порождает тем большее количество малых векторов, чем больше размерность поля, на которое эти идеальные слои накладываются. Соответственно из-за большого количества отдельных векторов уменьшается объём параллельных вычислений, а вместе с тем падает производительность.

4. Заключение

Подводя итоги анализа, следует отметить, что, если требуется решать вычислительную задачу электродинамики, где в качестве источника используется плоская волна, предпочтительнее использовать метод TFRF. Основываясь на результатах, полученных в данной работе, (рисунок 3) можно утверждать, что на недорогих видеокартах можно выполнять расчеты вычислительных задач электродинамики с ускорением до 20 раз с использованием CPML и до 60 раз без него. Однако встает другая проблема: ограниченность объёма графической памяти по

сравнению с оперативной. Решить данную проблему можно путём декомпозиции, разбиением области значений на блоки такого объёма, при котором вычислительный процесс имеет максимальное ускорение.

5. Литература

[1] Головашкин, Д.Л. Разностный метод решения уравнений Максвелла: учебное пособие / Д.Л. Головашкин, Н.Л. Казанский. – Самара: Издательство Самарского государственного аэрокосмического университета, 2007. – 160 с.
[2] Сойфер, В.А. Дифракционная нанофотоника / В.А. Сойфер. – М.: Физматлит, 2011. – 680 с.
[3] Taflove, A. Computational Electrodynamics: the finite-difference time-domain method / A. Taflove. – London: Artech House, 1995. – 599 p.
[4] Elsherbeni, A.Z. The finite-difference time-domain method for electromagnetics with MATLAB simulations / A.Z. Elsherbeni, V. Demir. – Raleigh: SciTech Publishing, 2008. – 426 p.

Implementation of the finite-difference method for solving Maxwell`s equations in MATLAB language on a GPU

N.D. Morunov[1]

[1]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086

Abstract. The FDTD method of solving the Maxwell equations and methods for specifying a plane wave for it are considered. In addition, the features of implementation of calculations in MATLAB via graphics processing unit were considered. The comparative analysis of the difference solution of the Maxwell equations in the language MATLAB on the GPU for different ways of specifying a plane wave using an inexpensive graphics card is proposed. As a result, in the language MATLAB via the user video card NVIDIA GTX 960m you can speedup calculations to 60 times.

Keywords: maxwell's equations, FDTD, GPGPU, MATLAB.