# Реализация блочного алгоритма FDTD-метода на языке MATLAB с использованием графического процессора. Случай двумерной декомпозиции

**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 непрофессиональном графическом процессоре. В качестве решения предложен двумерный 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 10 млн. до 85 млн. узлов со средним ускорением в 7,5 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.

Можно конечно пойти путем улучшения технического обеспечения, но высокопроизводительная аппаратура ничего не даст без оптимизированного под неё алгоритма [2]. Производительность при использовании традиционного алгоритма с послойной синхронизацией упирается в пропускную способность памяти. Т.е. процессор большую часть времени простаивает, пока к нему или от него поступает новая порция данных.

Подобная проблема решается путем использования блочных алгоритмов, которые используют иерархию памяти и за счет увеличения локальности данных уменьшают простои процессора. Так, либо максимально задействуется кэш, либо графическая память. Ранее блочные алгоритмы FDTD-метода уже реализовывались на CUDA для вычислений на кластере с использованием графических процессоров [3], но нет реализации на MATLAB. Преимущество MATLAB в том, что для него не нужно ничего программировать как для CUDA. Простой синтаксис и поддержка библиотек с основными математическими методами позволяют просто и быстро

решить поставленную математическую задачу. Но существуют некоторые особенности, которые стоит учитывать при решении задач вычислительной электродинамики на MATLAB, они уже были рассмотрены одним из авторов в статье [4]. Кроме того, стоит заметить, что в реализации на CUDA не рассматривались поглощающие слои, из-за чего та статья не представляет практического интереса.

Ранее автором уже был рассмотрен блочный алгоритм при одномерной декомпозиции двумерной сеточной области [5]. Следуя правилу Фостера — от поверхности к объему [6], следующим этапом необходимо рассмотреть двумерную декомпозицию.

Итак, целью работы является исследование блочного алгоритма FDTD-метода с использованием двумерной декомпозиции сеточной области для преодоления ограничения на объём видеопамяти при организации вычислений по данному методу и его реализации на языке MATLAB.

## 2. Блочные алгоритмы

Блочные алгоритмы известны довольно давно ещё в матричных вычислениях. Такие алгоритмы предназначены для выполнения на вычислительных системах с некоторой иерархией памяти. Предполагая, что вычисления внутри блоков будут проходить с использованием более быстрой памяти, а вся сеточная область будет храниться в медленной памяти. В нашем случае вычисления будут проходить не только на более быстрой памяти (графической), но и на более быстром процессоре (GPU) за счёт большего числа вычислительных ядер.

Бывают несколько разновидностей блочных алгоритмов: DiamondTile и DiamondTorre. Каждый узел сетки на текущем временном слое имеет зависимости от нескольких узлов на предыдущем слое, определяемые дифференциальным шаблоном и влияет на узлы следующего временного слоя. Таким образом можно выделить области влияния и зависимости (рисунок 1). На пересечении этих областей получаются ромбовидные фигуры (DiamondTile). Такие ромбовидные фигуры можно сделать симметричными и тогда алгоритм называют DiamondTile, или ленточными (в виде башен) такой алгоритм называют DiamondTorre. Однако так как в DiamondTorre все блоки требуют строго последовательного выполнения, разбивать область на «башни» можно только при условии, что граничные условия исходной области не являются циклическими.

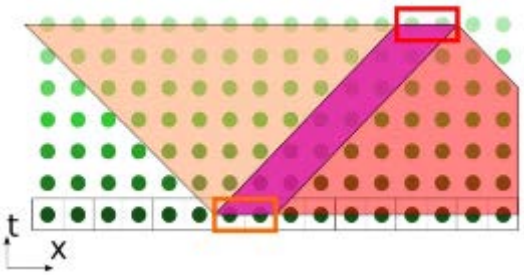


**Рисунок 1.** Пересечение области влияния (жёлтый цвет) и области зависимости (красный цвет).

Кроме этого блочные алгоритмы можно разделить на алгоритмы с одномерной декомпозицией и с двумерной декомпозицией. Блочные алгоритмы с одномерной декомпозицией, и их реализация рассмотрены в предыдущей работе одного из авторов [5]. В данной статье рассмотрим алгоритмы с двумерной декомпозицией.

Алгоритм DiamondTile предполагает разбиение сеточной области на ромбоидальные блоки. В случае двумерной декомпозиции блоки будут иметь форму октаэдра и тетраэдра, и чередоваться друг с другом как показано на рисунке 2а.

Алгоритм DiamondTorre предполагает разбиение сеточной области на блоки, по форме напоминающие башни. В таком случае можно будет выделить слои блоков, в каждом из которых в некоторой последовательности обрабатываются блоки одинаковой формы (рисунок 2б). Несмотря на то, что DiamondTorre более удобен для реализации, он проигрывает алгоритму DiamondTile в эффективности.

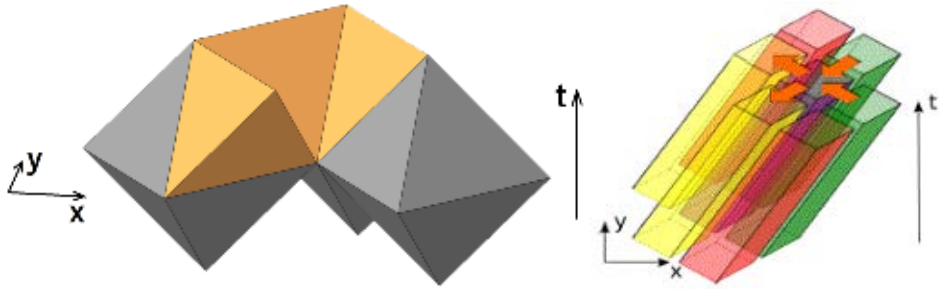


**Рисунок 2.** Двумерная декомпозиция сеточной области по алгоритму: а) DiamondTile; б) DiamondTorre.

Рассмотрим преимущества двумерной декомпозиции по сравнению с одномерной на примере алгоритма DiamondTorre. Можно заметить, что площадь проекции блока эквивалентна максимальному количеству узлов, помещающихся в блок, тогда:

$$C_{1D} = (N_x + T - 1)a \cdot N_x; \qquad T \approx \frac{C_{\max}}{a \cdot N_x} - N_x;$$

$$C_{2D} = (2D - 1)(T - 1) + 2D^2; \qquad T \approx \frac{C_{\max}}{2D} - D; \qquad (1)$$

где a – коэффициент, определяющий форму блока, $a = N_y/N_x$; $N_x$ – размер блока по оси x; $N_y$ – размер блока по оси y; T – высота блока; D – размер основания блока.

Для двумерной декомпозиции эта площадь определяется только лишь параметрами блока (D), для одномерной еще и формой блока (a). Т.е. двумерная декомпозиция предпочтительнее для сеточной области квадратной формы больших размеров (форма и размер определяются параметрами дискретизации), так как в этом случае дробление области на большее количество блоков не ограничивает максимальную высоту этих блоков.

Также стоит упомянуть про эффективность одномерной декомпозиции уменьшающуюся при высоком *a*. За критерий эффективности можно взять вычислительную интенсивность, которая определяется как отношение количества операций к количеству узлов, задействованных в этих операциях:

$$\frac{O_{1D}}{C_{1D}} = \frac{N_x T}{N_x + T}; \qquad \frac{O_{2D}}{C_{2D}} = \frac{D \cdot T}{D + T}.$$

Как было уже замечено выше в формуле (1), *D* определяется доступным объемом видеопамяти, и как показывает практика на порядок превосходит *T*. Однако $N_x$ зависит от формы блока (от параметра *a*), чем больше параметр *a*, тем меньше $N_x$, и меньше производительность.

### 3. Реализация блочного алгоритма на языке MATLAB

Пример разбиения на блоки двухмерной по пространству сеточной области представлен на рисунке 3. Ранее в работе [5] оговаривалось то, что использование индексов в MATLAB сильно уменьшает время вычислений, поэтому следует использовать функцию arrayfun(), а для неё обрамлять блок в рамки (рисунок 4). В таком случае реализация алгоритма DiamondTile на языке MATLAB будет не эффективна, так как объем октаэдра в 6 раз (для тетраэдра в 20 раз) меньше чем объем куба в который он вписан, соответственно и число полезных операций в 6 раз (в 20 раз) меньше по сравнению с общим количеством операций.

В случае с алгоритмом DiamondTorre в зависимости от высоты блока число операций в 2-4 раза меньше чем общее количество операций. Как видно несмотря на большую эффективность алгоритмов с двумерной декомпозицией эффективность их реализаций на MATLAB ниже чем у одномерной.

Кроме этого в случае с двумерной декомпозицией сложно выделить крайние блоки, что необходимо для введения поглощающих слоёв. От этого вычисления в блоках покрываются ветками условий, что незначительно, но плохо скажется на производительности.

**Рисунок 3.** Пример разбиения на блоки в пространстве, разными цветами помечены разные блоки, шар — Ey, пирамида — Ex, куб — Hz.

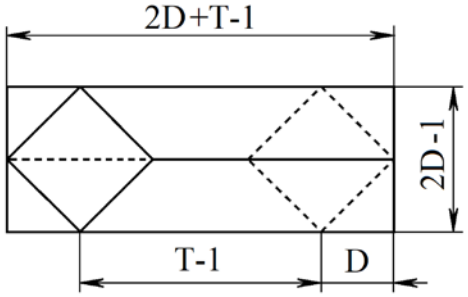


**Рисунок 4.** Обрамление блока в виде сверху, D – размер основания блока, T – высота блока.

Далее будет рассмотрен расчёт границ блоков. В алгоритме сперва задаётся сетка разбивающая область задачи на одинаковые ромбы, таким образом, чтобы каждая вершина в этой сетке обладала уникальным индексом (номером блока). Эти вершины будут центрами оснований «башен». Далее рассчитываются границы рамки для каждого блока. Ниже представлен фрагмент алгоритма на языке MATLAB, отвечающего за расчёт границ блоков для реализации двумерной декомпозиции:

```
kN = ceil(N/D); kM = ceil(M/D); D_ = floor(M/kM); k=1;
for i=0:kN+1+ceil(T/D_)-1
for j=0:kM
if (mod(i+j,2)==0)
Nd(k) = D_*i+1; Md(k) = D_*j+1; k=k+1;
end end end
Nu = Nd-T+1; Mu = Md;
Nbl = max(Nu-D_,1); Mbl = max(Mu-D_,1);
Ntr = min(Nd+D_,N); Mtr = min(Md+D_,M);
```

**4. Экспериментальное исследование**

Далее был проведен эксперимент, целью которого было исследовать эффективность блочного алгоритма двумерного случая FDTD-метода на языке MATLAB с использованием двумерной декомпозиции сеточной области.

Учитывая все особенности реализации была написана программа на языке MATLAB. Программа запускалась на пользовательском ноутбуке с четырёхъядерным центральным процессором Intel Core i5-6300HQ с базовой тактовой частотой 2,3 ГГц и максимальной – 3,2 ГГц, и с набором микросхем HM170. Также на ноутбуке было установлено 8 ГБ оперативной памяти типа DDR4-2133, и графическая карта NVIDIA GeForce GTX 960M, подключенная через шину PCI Express x16, с объёмом внутренней оперативной памяти GDDR5 – 2 ГБ и числом CUDA-совместимых шейдерных процессоров – 640 с тактовой частотой 1,1 ГГц.

Для того чтобы исследовать эффективность реализации рассмотренного блочного алгоритма необходимо построить график зависимости производительности вычислений от общего количества узлов в одном временном слое, при вычислениях в каждом блоке максимально задействовать графическую память. Было определено, что максимально задействованной видеопамяти соответствует максимальное количество узлов — 8,7 млн. узлов.

В программе в качестве параметров задаются размер основания блока (D), высота блока (T) и общее количество узлов (C). Тогда размер основания блока соответствующий количеству узлов в одном блоке ($C_1$) можно определить следующим образом:

$$C_1 = (2D-1)(D+T-1); \qquad D = \frac{\sqrt{(T-2)^2 + 4(C_1+T-1)} - (T-2)}{4}.$$

Также необходимо рассмотреть зависимость времени выполнения алгоритма от высоты блока. Время выполнения алгоритма складывается из времени выполнения вычислений и времени выполнения коммуникаций:

$$t = n \cdot t_{выч} + \frac{n}{T} \cdot t_{ком}. \quad (2)$$

В результате работы профилирующих скриптов были получены: график зависимости производительности вычислений от количества узлов для двумерного случая FDTD и двумерной декомпозиции, график и таблица зависимости времени вычислений блочного алгоритма двумерного случая FDTD с двумерной декомпозицией блоки, размер которых соответствует эффективному размеру сеточной области.

На рисунке 5 можно увидеть среднее значение производительности вычислений для эффективного объёма памяти – 72 (ускорение в 7 раз) и для максимального – 60 (ускорение в 6 раз). График зависимости времени выполнения от высоты блока имеет форму гиперболы (рисунок 6).

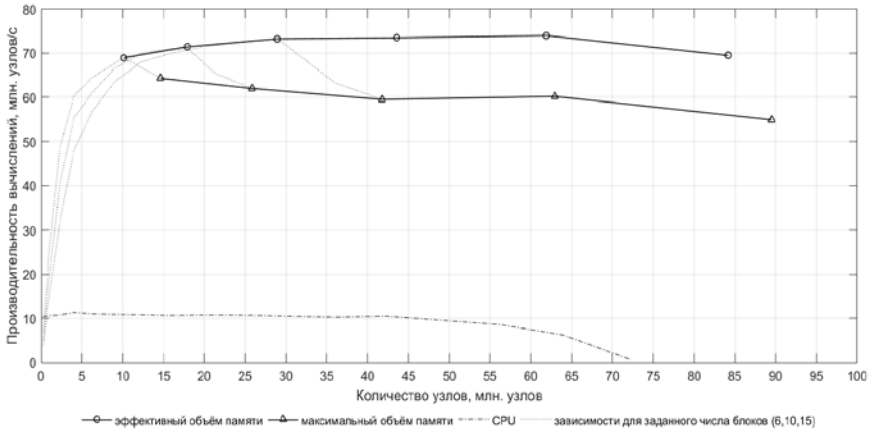


**Рисунок 5.** Графики зависимости производительности вычислений от количества узлов (млн. узлов) для двумерного случая FDTD и двумерной декомпозиции.

Стоит отметить, что по гиперболе можно определить время выполнения вычислений и время коммуникаций. Так если минимальное значение, к которому стремится график, разделить на количество временных слоев (1024) то можно получить время вычислений, а отношение разности максимального значения времени и минимального к количеству временных слоев есть время выполнения коммуникаций (2). Тогда получаем время вычислений в одном временном слое — 0,4 с., а время коммуникаций в одном слое блоков — 5,6 с.

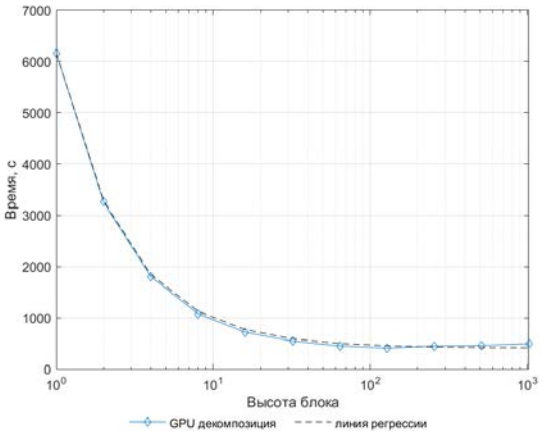


**Рисунок 6.** График зависимости времени вычислений блочного алгоритма двумерного случая FDTD с двумерной декомпозицией на блоки, размер которых соответствует эффективному размеру сеточной области.

Сравнивая результаты с результатами одномерной декомпозиции [5] можно сделать вывод, что эффективность реализации двумерной декомпозиции в 2 раза меньше чем у одномерной. Для того, чтобы её повысить 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.

## 5. Заключение

Блочный алгоритма FDTD-метода с использованием двумерной декомпозиции сеточной области для преодоления ограничения на объём видеопамяти при организации вычислений по данному методу, и его реализация на языке MATLAB исследованы. Выбран наиболее подходящий блочный алгоритм для реализации на MATLAB – DiamondTorre. Рассмотрены особенности его реализации на языке MATLAB. В результате было установлено, что: максимальная производительность которой удалось достичь таким способом — 75 млн. узлов / с., что при данных аппаратных средствах соответствует ускорению в 7,5 раз; максимально возможная дискретизация сетки при вычислениях на GPU была расширена с 10 млн. до 85 млн. узлов. У двумерной декомпозиции есть свои преимущества над одномерной. Так, например, при использовании в расчётах достаточно большой сеточной области в алгоритме с одномерной декомпозицией возникнет ситуация, когда высоту блока нельзя сделать больше одного. В случае с двумерной декомпозицией такой проблемы нет.

## 6. Литература

[1] Taflove, A. Computational Electrodynamics: the finite-difference time-domain method / A. Taflove. – London: Artech House, 1995. – 866 p.

[2] 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.

[3] 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.

[4] Morunov, N.D. Implementation of the finite-difference method for solving Maxwell`s equations in MATLAB language on a GPU // CEUR Workshop Proceedings. – 2018. – Vol. 2212. – P. 146-151.

[5] Морунов, Н.Д. Реализация блочного алгоритма FDTD-метода на языке MATLAB с использованием графичского процессора // Материалы Международной научно-технической конференции «Аналитические и численные методы моделирования естественно-научных и социальных проблем» (АЧМ). – Пенза: Издательство ПГУ, 2018. – С. 44-50.

[6] Foster, I. Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. – Addison-Wesley Longman Publishing Co, 1995. – 370 p.

## Благодарности

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

# Implementation of the FDTD method block algorithm in the MATLAB language using a graphics processing unit. 2D-decomposition case

**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 a non-professional graphics processor was considered in this article. As a solution, a block algorithm of the FDTD method with 2-D decomposition 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 10 million to 85 million nodes with an average acceleration of 7,5 times for the two-dimensional case of the FDTD method.