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

To cite this article: N D Morunov and D L Golovashkin 2019 J. Phys.: Conf. Ser. 1368 052002

View the article online for updates and enhancements.
Implementation of the FDTD method block algorithm in the MATLAB language using a graphics processing unit. 2D-decomposition case

N D Morunov¹, D L Golovashkin¹²

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

e-mail: niknixad@mail.ru

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.

1. Introduction
A numerical solution of Maxwell's equations is quite popular. A numerical method for solving such equations was called the finite difference time domain (FDTD) method in 1980 by Alain Taflov [1]. The method is based on Yee algorithm developed in 1966, which consists in replacing the derivatives with central difference relations in computational domain. The method still finds application in the problems of electrodynamics, optics and nanophotonics [2]. However, it requires a large amount of computing resources namely time and memory. It can be solved by improving hardware, but even so, more powerful computing system requires more effective algorithm [3]. The effectiveness of the traditional FDTD algorithm with layer-by-layer synchronization is bounded by the memory bandwidth, i.e. the processor idle most of the time until a new portion of data arrives.

One of the current paper authors already tried to solve this problem using the pyramid method [4]. However, data is duplicated when calculating by the method, which negatively affects the effectiveness of the method. In the current work, a block algorithm is used, which involves splitting the entire computation domain into disjoint blocks. The block algorithm uses a memory hierarchy and reduces processor downtime by increasing the locality of the data. So, the fastest memory (CPU cache or video memory) is as much as possible involved in this hierarchy. Previously, the FDTD method
block algorithms were already implemented on CUDA for computing on a cluster using graphics processors [5], but it has not yet been implemented in MATLAB. The advantage of MATLAB is that it does not need to program anything as CUDA. Simple syntax and libraries with basic mathematical methods allow quick and easy solving of a number of mathematical problems. Nevertheless, some MATLAB features should be considered when solving problems of computational electrodynamics via GPU in the MATLAB. This issue has already been considered in articles [6, 7].

Also earlier, one of the authors has already considered a block algorithm for one-dimensional decomposition of a two-dimensional computational domain [8]. Following the Foster's rule from surface to volume [9], the next step is to consider a two-dimensional decomposition.

Therefore, the aim of the work is to design, implement and research the FDTD method block algorithm using a two-dimensional decomposition of the computational domain to overcome the limitations on the amount of video memory when organizing calculations using this method and its implementation in the MATLAB language.

2. Block algorithms

Block algorithms have been known for quite some time in matrix calculations. Such algorithms are designed to run on computing systems with a memory hierarchy. It is assumed that the intrablock calculations are performed using fast memory, and entire computational domain is stored in slow memory. In our case, the calculations not only use faster graphics memory, but also it is computed on a more efficient processor (GPU) due to a larger number of cores.

There are several types of block algorithms: DiamondTile and DiamondTorre. Each grid node on the current time layer has dependencies on several nodes of the previous time layer and affects the nodes of the next time layer. It is defined by a differential stencil. Thus, influence and dependence domain can be distinguished (Figure 1). The intersection of these domains is diamond-shaped figures. Such diamond shapes can be made symmetrical and the algorithm is called DiamondTile, or band and the algorithm is called DiamondTorre. DiamondTorre require sequential execution all of blocks, therefore, the domain can be decomposed into "torres" if the boundary conditions of the computational domain are not cyclic.

![Figure 1. Intersection of the influence (yellow) and dependence (red) domains.](image1)

In addition, block algorithms can be divided into algorithms with one-dimensional decomposition and two-dimensional decomposition. Block algorithms with one-dimensional decomposition and its implementation were considered in a previous paper by one of the authors [8]. In this article we will consider algorithms with two-dimensional decomposition.

![Figure 2. Two-dimensional decomposition of the computational domain by the algorithm: a) DiamondTile; b) DiamondTorre.](image2)
The DiamondTile algorithm consists in splitting the computational domain into diamond shaped blocks. In the case of a two-dimensional decomposition, the blocks will have octahedron and tetrahedron shapes and alternate as shown in Figure 2a.

The DiamondTorre algorithm consists in partitioning the computational domain into tower shaped blocks. In this case, it will be possible to pick out block layers, each of which blocks of the same form are processed in a certain sequence (Figure 2b). Despite the fact that DiamondTorre is more convenient to implement, it loses the DiamondTile algorithm in efficiency.

Consider the advantages of two-dimensional decomposition compared to one-dimensional one on the example of the DiamondTorre algorithm. Note that the projected area of the block is equivalent to the maximum number of nodes filled in the block, then:

\[
C_{1D} = (N_x + T - 1)a \cdot N_x; \quad T = \frac{C_{max}}{a \cdot N_x} - N_x;
\]

\[
C_{2D} = (2D - 1)(T - 1) + 2D^2; \quad T = \frac{C_{max}}{2D} - D;
\]

where \(a\) is block shape factor, \(a = N_y/N_x\); \(N_x\) is x-axis block size; \(N_y\) is y-axis block size; \(T\) is block height; \(D\) is block base size.

This area is only determined by block size \((D)\) for two-dimensional decomposition and by block shape \((a)\) for one-dimensional as well. Therefore, two-dimensional decomposition is more preferable for a computational domain with square shape and large dimensions (the shape and size are determined by the discretization parameters), since in this case splitting the domain into more blocks does not bound the maximum block height.

It is worth mentioning the efficiency of one-dimensional decomposition decreases with a high value of \(a\). The calculation intensity was taken as a performance criterion, that is defined as the ratio of the operation count to the number of nodes in one-time layer:

\[
\frac{O_{1D}}{C_{1D}} = \frac{N_x T}{N_x + T}; \quad \frac{O_{2D}}{C_{2D}} = \frac{D \cdot T}{D + T}.
\]

As already noted above in formula (1), \(D\) is determined by the available video memory size, and as practice shows it is an order of magnitude higher than \(T\). However, \(N_x\) depends on the block shape \((a)\), the larger the parameter \(a\), the smaller \(N_x\), and the lower the performance.

3. Implementation of the block algorithm in MATLAB

An example of block division of a two-dimensional grid domain in space is shown in Figure 3. It was claimed that the use of indices in MATLAB greatly reduces runtime in earlier report [8]. So, we should use the arrayfun() function and frame the block rectangular. In this case, the implementation of the DiamondTile algorithm in the MATLAB language will not be effective, since the volume of octahedron is 6 times (20 times for tetrahedron) less than the volume of the cube in which it is inscribed. Therefore, the number of useful operations is 6 times (20 times) less compared to the total number of operations.

![Figure 3. Example of decomposition into blocks in space, different blocks are marked with different colors, ball is EY, pyramid is EX, cube is HZ.]

![Figure 4. Framing the block in the top view, D is the block base size, T is the block height.]
When implementing the DiamondTorre algorithm, the number of operations is 2–4 times less than the total number of operations depending on the block height. As can be seen, despite the high efficiency of algorithms with two-dimensional decomposition, the efficiency of their implementation on MATLAB is lower than that of algorithms with one-dimensional decomposition.

In addition, in the case of a two-dimensional decomposition, it is difficult to isolate boundary blocks that are necessary for the introduction of absorbing layers. Because of this, the calculations in the blocks acquire conditional branches, which slightly affect the performance.

Next, we consider the calculation of the boundaries of the blocks. First, the algorithm sets a grid that divides the computational domain into identical rhombuses, so that each vertex in this grid has a unique index (block number). These peaks are the centers of the bases of the "torres". Next, the frame boundaries for each block are calculated. A fragment of the algorithm is given below in the MATLAB language, which is responsible for calculating the block boundaries for the implementation of two-dimensional decomposition:

```matlab
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. Experimental research

Next, an experiment was conducted, the purpose of which was to investigate the effectiveness of the 2D FDTD block algorithm in the MATLAB language via GPU using a two-dimensional decomposition of the computational domain.

Taking into account all the features of the GPU implementation, a program was written in the MATLAB language. The program was run on a custom laptop with a four-core Intel Core i5-6300HQ CPU with a clock speed from 2.3 GHz to 3.2 GHz, and with the HM170 chipset. Also on the laptop was installed 8 GB of DDR4-2133 RAM, and an NVIDIA GeForce GTX 960M graphics card connected via the PCI Express x16 bus, with 2 GB of internal GDDR5 RAM and the number of CUDA-compatible cores is 640 with clock speed 1.1 GHz.

In order to investigate the efficiency of the implementation of the considered block algorithm, it is necessary to plot of the computing performance versus the total number of nodes in one-time layer, when calculating in each block use the graphic memory as much as possible. It was determined that the maximum number of nodes is 8.7 million nodes that corresponds to the maximum video memory used.

In the program, the block base size (D), the block height (T) and the total number of nodes (C) are set as parameters. Then the block base size corresponding to the number of nodes in one block (C1) can be determined as follows:

\[ C_1 = (2D - 1)(D + T - 1); \]

\[ D = \sqrt{(T - 2)^2 + 4(C_1 + T - 1) - (T - 2)} / 4. \]

It is also necessary to consider the dependence of the algorithm runtime on the block height. The algorithm runtime is the sum of calculations and communications runtimes:

\[ t = n \cdot t_{calc} + \frac{n}{T} \cdot t_{comm}. \] (2)

As a result of the profiling scripts work, we obtained: a plot of the computing performance versus the number of nodes for the two-dimensional FDTD case and two-dimensional decomposition; a plot and table of the block algorithm runtime versus the block height for the two-dimensional FDTD case.
with two-dimensional decomposition, and with the block size corresponds to the effective size of the computational domain.

From the Figure 5, it can be seen that the average value of the computing performance is 72 (speedup 7 times) for an effective memory size and 60 (speedup 6 times) for maximum one. The plot of runtime versus block height is in the form of a hyperbola (Figure 6).

![Figure 5](image1.png)

**Figure 5.** Plots of the computing performance versus the number of nodes (million nodes) for the 2D FDTD and two-dimensional decomposition.

![Figure 6](image2.png)

**Figure 6.** Plot of the block algorithm runtime versus the height block for 2D FDTD with two-dimensional decomposition into blocks, and with the block size corresponds to the effective size of the computational domain.

It should be noted that with the help of hyperbola could be determined the time to perform calculations and the time of communications. So, the calculation runtime is defined as the ratio of the minimum value is tended by the plot to the number of time layers (1024). And the ratio of difference between maximum and minimum time to number of time layers is the communications runtime (2). Then, the calculation time is obtained in one-time layer is 0.4 s, and the communication time in a single block layer is 5.6 s.

Comparing the obtained results with the results of one-dimensional decomposition [8], we can conclude that the effectiveness of the implementation of two-dimensional decomposition is 2 times...
less than that of the one-dimensional. It is necessary somehow remove idle operations from the calculations to increase performance. The nodes making the largest contribution to the additional elapsed time are located at the block base. They form a square from a rhombus doubling the calculation time.

5. Conclusion
The block algorithm of the FDTD method using a two-dimensional decomposition of the computational domain to overcome the limitations on the video memory size when organizing computations using this method and its implementation in the MATLAB language are investigated. The most suitable kind of block algorithm was chosen for implementation in MATLAB namely DiamondTorre. The features of its GPU implementation in the MATLAB language are considered. As a result, it was found that: the maximum achieved performance in this way is 75 million knots / s., that with the given hardware corresponds to 7.5x speedup; the maximum possible discretization of the grid domain in GPU calculations has been expanded from 10 million to 85 million nodes. However, the two-dimensional decomposition has its advantages over the one-dimensional. So, for example, a situation, in which the block height cannot be made more than one, will arise when using high discretization domain in algorithm with one-dimensional decomposition. There is no such problem in the two-dimensional decomposition case.

6. References
[1] Taflove A 1995 Computational Electrodynamics: the finite-difference time-domain method (London: Artech House) p 866
[2] Siraji A A, Zhao Y 2019 Design and analysis of thin optical lens composed of low-index subwavelength structures Applied Optics 58(17) 4654-4664
[3] Williams S, Waterman A, Patterson D 2009 Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures Communications of the ACM 52 65-76
[4] Malysheva S A, Golovashkin D L 2016 Implementation of the FDTD algorithm on GPU using a pyramid method Computer Optics 40(2) 179-187 DOI: 10.18287/2412-6179-2016-40-2-179-187
[5] Zakirov A V, Levchenko V D, Perepelkina A Yu and Zempo Y 2016 Diamond Torre algorithm and high-performance implementation of the FDTD method for supercomputers with graphics accelerators Proceedings of “Supercomputer days in Russia” 80-94
[6] Morunov N D 2018 Implementation of the finite-difference method for solving Maxwell’s equations in MATLAB language on a GPU CEUR Workshop Proceedings 2212 146-151
[7] Diener J E, Elsherbeni A Z 2017 FDTD Acceleration using MATLAB Parallel Computing Toolbox and GPU Applied Computational Electromagnetics Society Journal 32(4) 283-288
[8] Morunov N D 2018 Implementation of the FDTD method block algorithm in the MATLAB language using a graphics processing unit Proceedings of the International Scientific and Technical Conference "Analytical and numerical methods for modeling natural science and social problems" 44-50
[9] Foster I 1995 Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering (Addison-Wesley Longman Publishing Co) p 370

Acknowledgment
This work was supported by the Russian Foundation for Basic Research grant 19-07-00423-A and by the Ministry of Science and Higher Education within the State assignment to the FSRC “Crystallography and Photonics” RAS No. 007-GZ/Ch3363/26.