Parallel FDTD Solver with Optimal Topology and Dynamic Balancing

Gleb Balykov

balykov.gleb@yandex.ru

Abstract. Finite-difference time-domain method (FDTD) is widely used for modeling of computational electrodynamics by numerically solving Maxwell's equations and finding approximate solution at each time step. The FDTD method was originally developed by K.Yee in 1966 and is still improving to fulfill the needs of researchers. Highly parallel Maxwell's equations solvers based on the FDTD method allow to model sophisticated structures on large grids with acceptable performance and required accuracy. This article describes parallel FDTD solver for different dimensions with comparison method for virtual topologies of computational nodes' grid, which allows to choose the best virtual topology for target architecture. Developed solver also incorporates dynamic balancing of computations between computational nodes. Measurements for presented algorithms are provided for IBM Blue Gene/P supercomputer. Further directions for optimizations are also discussed.

Key words: $ComputationalElectrodynamics \cdot FDTD \cdot ParallelFDTD \cdot MPI$

1 Overview

The FDTD method is widely used in electrodynamics solvers as well as its different parallelization techniques. After it had originated in 1966 [1], it had a long road from sequential algorithm to implementations of high-performance parallel versions. This happened along with development of new hardware and architectures, giving engineers opportunities to develop and evolve FDTD algorithm.

Three commonly used parallelization technologies for the FDTD method are MPI, OpenMP and Cuda. Each of them serves its own purpose: MPI is a standard for high-performance parallel computations on architectures with distributed memory, OpenMP is a standard for high-performance parallel computations on architectures with shared memory, Cuda is a parallel computing platform and API for parallel computations on Nvidia GPUs.

The most common trend in parallelization of FDTD algorithm is still a combination of MPI and OpenMP, however, interest in massive parallel computations rapidly shifts towards GPUs and computations on them and FDTD algorithm is no exception. Cuda FDTD solvers give engineers opportunities to perform electrodynamic modeling on systems varying from personal computers equipped with Nvidia GPUs to GPU clusters. But FDTD solvers developed for heterogeneous architectures with support of MPI, OpenMP and Cuda present the most interest [2][3][4].

Load balancing in parallel FDTD algorithms allows to achieve the best performance possible for current parameters of computation and characteristics of the computational system. In general, load balancing couldn't be done without characteristics of the system, on which computations are performed [5]. However, for homogeneous architectures load balancing could be performed statically before computations in some cases.

In this article, parallel FDTD solver with optimal topology and dynamic balancing is introduced [6], which incorporates algorithm of choosing of optimal virtual topologies for computational nodes' grid for homogeneous systems and dynamic balancing of computations between computational nodes. Solver has UPML and TF/SF support and supports both complex and real values with different precision. Besides, solver supports Cuda and could perform computations on GPUs. Combination of MPI and Cuda, which could be enabled separately, allows to achieve significant speed up on a wide range of target architectures and high portability of developed solver. In case of heterogeneous architectures, dynamic balancing could also be applied. Two other possible solutions for heterogeneous architectures are also discussed.

2 Parallel Algorithm Description

Electrodynamics modeling could be performed in different dimensions, i.e. one-dimensional modeling (1D), two-dimensional (2D) and three-dimensional (3D). For all dimensions Cartesian computational grid is introduced: to be specific, Ox axis is defined in case of 1D mode, Ox and Oy axes in case of 2D mode, Ox,Oy and Oz axes in case of 3D mode. Yee grid [7] for field components is then set, and all points of Yee grid are spread between all computational nodes. In case of sequential solver, all points of Yee grid remain on the one and only computational node. Thus, each point of Yee grid is assigned to one or another computational node.

In parallel FDTD algorithm, described here, points of Yee grid are spread between computational nodes in a very natural way: Yee grid is divided in rectangular chunks and each chunk is assigned to computational node. Besides, each computational node has buffer points on its borders in order to store data from neighboring computational nodes. This computational nodes' grid maps directly on MPI virtual topology, where each MPI process is launched on different computational node and virtual topologies are simply MPI virtual topologies.

Share operations between computational nodes are performed at each time step, so overall computational time is sum of computational time and share time for each time step. Note that only maximum sum of computational time and share time for each time step is taken into account, i.e. if one computational node performs its computations much slower than other nodes, all other nodes would have to wait for it to finish.

Each computational node performs computations on chunk of Yee grid points assigned to it and then performs share operations with all its neighboring computational nodes. Computations are the same as for sequential algorithm and could be performed either on CPU, or on GPU if computational node has one.

Share operations consist of the next steps. All directions in which share operations could be performed are considered one after another, and each computational node sends data in this directions, and also receives data from the opposite directions at the same time. In case of send operation, data in border points of node's chunk is send, in case of receive, received data is stored in buffers.

For example, for 2D mode there are 8 directions — same as number of neighbors of computational node: 1 direction — positive by Ox axis, 2 — positive by Ox and Oy axes (diagonal), 3 — positive by Oy axis, 4 — negative by Ox and positive by Oy axis (diagonal), 5 — negative by Ox axis, 6 — negative by Ox and Oy axes (diagonal), 7 — negative by Oy axis, 8 — positive by Ox and negative by Oy axis (diagonal). Fig. 1 on the left shows for computational node marked with number 0 all 8 possible send directions, Fig. 1 on the right shows send procedure in direction 1 for 9 computational nodes. Arrows show direction in which data is sent, so each computational node receives data from the opposite direction (in case it has such neighbor).

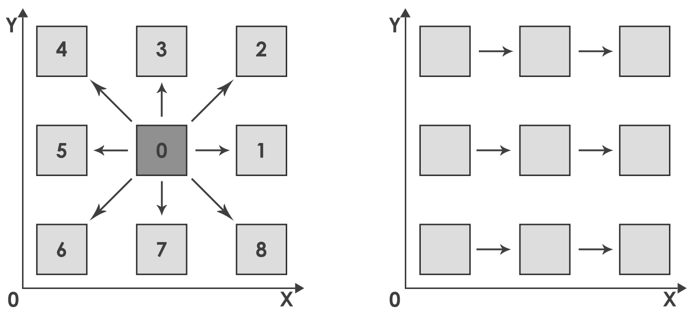

Fig. 1. 8 possible send directions for computational node marked 0 for 2D mode (on the left) and send procedure for 9 computational nodes for 2D mode (on the right). For a single computational node arrows show direction where data is sent, data is received from the opposite direction.

Division of Yee grid in rectangular chunks could be done in different ways. For 1D mode there is only one way — to divide Ox axis in chunks. In this case computational nodes would perform share operation only along Ox axis. Let's call this division of Yee grid 1D-X virtual topology. Oy and Oz axes could be divided the same way. Combining different axes divisions one can yield that for 2D mode there are 3 options: 2D-X, 2D-Y, 2D-XY virtual topologies, and for 3D mode there are 7 options: 3D-X, 3D-Y, 3D-Z, 3D-XY, 3D-YZ, 3D-XZ, 3D-XYZ virtual topologies. Fig. 2, 3, 4 show three kinds of virtual topologies for 2D mode. On the left full Yee grid is shown divided in chunks with data shown in light gray and on the right data assigned to each computational node is shown separately with required buffers shown in dark gray.

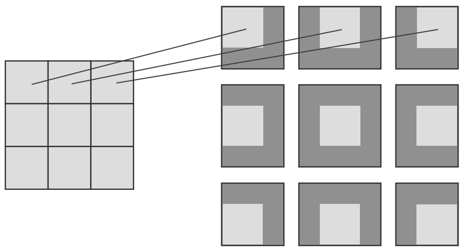

Fig. 2. 2D-XY virtual topology with full Yee grid divided in 9 chunks on the left and chunks assigned to 9 computational nodes (light gray) with required buffers (dark gray) on the right.

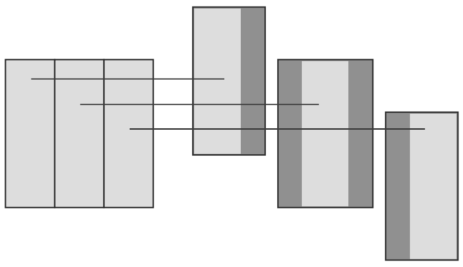

Fig. 3. 2D-X virtual topology with full Yee grid divided in 3 chunks on the left and chunks assigned to 3 computational nodes (light gray) with required buffers (dark gray) on the right.

There are two main cases for computational nodes and communicational network: all computational nodes are the same by performance and share time for all nodes is the same (homogeneous computing system), computational nodes are not the same by performance or share time for nodes is not the same (heterogeneous computing system).

Let N be the number of computational nodes used in computations. Let's consider it being determined somehow for now (e.g. by user of the solver).

First, let's consider case of homogeneous computing system for 2D and 3D modes (for 1D mode there is only one kind of virtual topology).

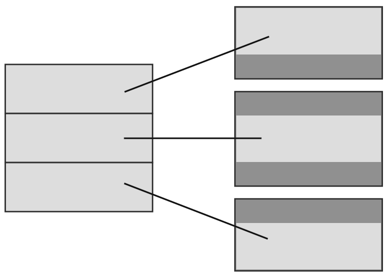

Fig. 4. 2D-Y virtual topology with full Yee grid divided in 3 chunks on the left and chunks assigned to 3 computational nodes (light gray) with required buffers (dark gray) on the right.

2.1 2D Mode for Homogeneous Computing System

Let $a > 0$ be the size of Yee grid by Ox axis and $b > 0$ be the size of Yee grid by Oy axis. Goal is to identify virtual topology to use for computations so that the overall computational time is minimal.

Let virtual topology have size of $n > 0$ computational nodes by Ox axis and $m > 0$ computational nodes by Oy axis, $N = n * m$. 2D-X virtual topology will be used in case $n = N$ and $m = 1$, 2D-Y topology in case $n = 1$ and $m = N$, otherwise 2D-XY virtual topology will be used, where $n \neq 1$ and $m \neq 1$.

Then, single computational node will have $a_1 = \lfloor a/n \rfloor$ grid points by Ox axis in the chunk assigned to it and $b_1 = \lfloor b/m \rfloor$ grid points by Oy axis. Total size of chunk assigned to computational node is $a_1 * b_1 = \lfloor a/n \rfloor * \lfloor b/m \rfloor$ grid points. Let's consider only cases where $a \bmod n = 0$ and $b \bmod m = 0$, which leads to the size of chunk being equal to

$$a_1 * b_1 = \frac{a}{n} * \frac{b}{m} = \frac{a * b}{N} \tag{1}$$

Computational time on single time step is proportional to number of Yee grid points in chunk of computational node $a_1 * b_1$ and share time on a single time step is proportional to the surface area of chunk $a_1 + b_1$. Number of Yee grid points in chunk is the same for all computational nodes, which means that computational time should also be the same. This is accurate in case each computational node performs same amount of computations, for example, this is not accurate if there is a point wave source with sophisticated wave function calculated only on one computational node and not calculated on others. In cases when nodes perform different amounts of computations on the same amount of grid points, dynamic information could be used and methods of solving such tasks are the same as for heterogeneous systems and are discussed later.

Thus, the minimal overall computational time could be achieved by minimizing share time on a single time step:

$$a_1 + b_1 = \frac{a}{n} + \frac{b}{m} = \frac{a}{n} + \frac{b * n}{N} = f(n) \tag{2}$$

Function $f(n)$ has only one extremum for $n > 0$ — global minimum:

$$n_0 = \sqrt{\frac{a * N}{b}} \tag{3}$$

$$m_0 = \frac{N}{n_0} = \frac{N}{\sqrt{\frac{a*N}{b}}} = \sqrt{\frac{b * N}{a}} \tag{4}$$

However, obtained values n_0 and m_0 might not be integer or might not be dividers of N, and a and b correspondingly. Values of n and m, which satisfy this conditions, have to be found.

First, all pairs of n and m which satisfy next conditions have to be found: n and m are dividers of N, and a and b correspondingly, and $N = n * m$. These pairs define set, which contains possible optimal values of n and m. In order to find these pairs, n has to be set equal to all dividers of $GCD(a, N)$, including 1 and $GCD(a, N)$ itself, and only those pairs have to be chosen for which corresponding $m = N/n$ is divider of b.

After that, two pairs (n'_0, m'_0) and (n''_0, m''_0) have to be found, for which for n the next conditions are satisfied: $n'_0 < n_0$ and $n''_0 \geq n_0$, and there are no pairs in the range $(n'_0; n''_0)$ (i.e., values n'_0 and n''_0 are the closest possible to n_0 from different directions). From two pairs (n'_0, m'_0) and (n''_0, m''_0) one has to be chosen, for which the value of $f(n)$ is smaller. Chosen pair describes the optimal virtual topology.

So, algorithm for choosing optimal virtual topology for homogeneous computational system for defined a, b, N consists of the next steps:

- Identify all pairs (n, m), for which n is divider of a, m is divider of b, $n * m = N$.
- Find n_0 using relation (3).
- Choose from the pairs found on the first step two (n'_0, m'_0) and (n''_0, m''_0), for which $n'_0 < n_0$ and $n''_0 \geq n_0$, and there are no pairs in the range $(n'_0; n''_0)$. From this two pairs one has to be chosen as optimal, for which the value of $f(n)$ is smaller.

There could be a case, when no appropriate pair is found (e.g., Yee grid could not be divided in chunks for N computational nodes). In this case N should be increased or decreased. Cases when $a \bmod n \neq 0$ or $b \bmod m \neq 0$ are not discussed here because they require dynamic information about computational system.

2.2 3D Mode for Homogeneous Computing System

Let $a > 0$ be the size of Yee grid by Ox axis, $b > 0$ be the size of Yee grid by Oy axis and $c > 0$ be the size of Yee grid by Oz axis.

Let virtual topology have size of $n > 0$ computational nodes by Ox axis, $m > 0$ computational nodes by Oy axis, $k > 0$ computational nodes by Oz axis,

$N = n * m * k$. For example, 3D-X virtual topology will be used in case $n = N$, $m = 1$, $k = 1$.

Similarly to 2D mode, number of grid points in chunk assigned to computational node $a_1 * b_1 * c_1$ is constant (in case $a \bmod n = 0$, $b \bmod m = 0$ and $c \bmod k = 0$). In order to minimize overall computational time, the next function $f(n, m)$ has to be minimized (share time on a single time step is proportional to $f(n, m)$):

$$f(n,m) = \frac{a * b}{n * m} + \frac{b * c * n}{N} + \frac{a * c * m}{N} + 4 * (\frac{a}{n} + \frac{b}{m} + \frac{c * n * m}{N}) \tag{5}$$

Again, all triples of (n, m, k) have to be identified, for which n is divider of a, m is divider of b, k is divider of c, $n * m * k = N$. One of these triples describes the optimal virtual topology. In order to find these triples n has to be set equal to all dividers of $GCD(a, N)$, including 1 and $GCD(a, N)$ itself, then m has to be set equal to all dividers of $GCD(b, N)$, including 1 and $GCD(b, N)$ itself, and only those triples have to be chosen for which corresponding $k = N/(n * m)$ will be a divider of c.

Then for all m from found triples the minimum of function $f(n, m)$ has to be found. This leads to the next formulas for chosen m:

$$n_0(m) = \sqrt{\frac{a * N}{m * c}} \tag{6}$$

$$k_0(m) = \frac{N}{n_0(m) * m} \tag{7}$$

Similarly to 2D mode, obtained values n_0 and k_0 might not be integer or might not be dividers of N, and a and c correspondingly. n and k, which satisfy this conditions, have to be found. Two triples (n_0', m, k_0') and (n_0'', m, k_0'') have to be found, for which for n the next conditions are satisfied: $n_0' < n_0$ and $n_0'' \geq n_0$, and there are no triples in the range $(n_0'; n_0'')$ (i.e., values n_0' and n_0'' are the closest possible to n_0 from different directions). From two triples (n_0', m, k_0') and (n_0'', m, k_0'') one has to be chosen, for which the value of $f(n, m)$ is smaller. Chosen triple describes the optimal virtual topology for specified m. After all values of m are handled, triples, found for each m, have to be compared and one triple with smallest $f(n, m)$ has to be chosen as the optimal virtual topology.

So, algorithm for choosing optimal virtual topology for homogeneous computational system for defined a, b, c, N consists of the next steps:

- Identify all triples (n, m, k), for which n is divider of a, m is divider of b, k is divider of c, $n * m * k = N$.
- For all allowed values for m find $n_0(m)$ using (6).
- Choose from the triples found on the first step two (n_0', m_0', k_0') and (n_0'', m_0'', k_0''), for which $n_0' < n_0$ and $n_0'' \geq n_0$, and there are no triples in the range $(n_0'; n_0'')$. From two triples one has to be chosen as optimal for specified m, for which the value of $f(n, m)$ is smaller.

– Virtual topologies found for each m have to be compared and one triple with smallest $f(n, m)$ has to be chosen as the optimal virtual topology.

2.3 Non-Specified Number of Computational Nodes for Homogeneous Computing System

In case when number of computational nodes N is not specified, there are two possible directions for optimization. First is to optimize by memory, i.e. choose the smallest number of computational nodes, memory of which is capable to store Yee grid. Second is to optimize by performance, i.e. choose the number of computational nodes in such a way that time of computations is minimal. The second problem requires dynamic information and methods of solving such tasks are the same as for heterogeneous systems and are discussed later.

Let's consider the first problem with optimization by memory for 2D mode. Problem is to choose N so that for specified a and b the number of Yee grid points per node $\frac{ab}{N}$ is maximal.

First the size of memory of a single computational node S has to be identified. The amount of memory occupied on a single computational node is $F(a, b, N)$:

$$F(a, b, n, m) = k_1 * (\frac{a * b}{n * m}) + k_2 * (\frac{a}{n} + \frac{b}{m}) + k_3 \tag{8}$$

where s is number of Yee grid points and k_1, k_2, k_3 — constants, which are defined by parameters of computation statically (during compilation of solver). Let's perform transformation in order to remove dependency on virtual topology.

$$k_1 * (\frac{a * b}{n * m}) + k_2 * (\frac{a}{n} + \frac{b}{m}) + k_3 < k_1 * (\frac{a * b}{n * m}) + k_2 * (a + b) + k_3 \tag{9}$$

Let's consider $F_1(a, b, N)$:

$$F_1(a, b, N) = k_1 * (\frac{a * b}{N}) + k_2 * (a + b) + k_3 \tag{10}$$

N_s has to be found, for which $F_1(a, b, N)$ is equal to S for specified a and b:

$$N_s = \frac{k_1 * a * b}{S - k_3 - k_2 * (a + b)} \tag{11}$$

An answer N_0 is the first divider of $a * b$ in ascending order, which will be greater or equal to N_s: $N_0 \geq N_s$ and $F_1(a, b, N_0) \leq F_1(a, b, N_s)$. Different choices of virtual topology will not affect the found answer, because it was taken into account. For 3D mode N_0 could be obtained in a similar way.

2.4 Heterogeneous Computing System

In case of heterogeneous systems, the algorithms described above couldn't be applied because either computational time on different computational nodes is proportional to number of grid points in chunk with different proportional coefficient, or share time has different proportional coefficients for different computational nodes. In such cases more sophisticated methods should be applied. Each one of the methods described below could give results, but the best option would be to use their combination.

Dynamic Balancing (Dynamic Redistribution). Dynamic redistribution of Yee grid points between computational nodes will allow to assign chunks of different sizes for a single computational node during computations (opposed to previously described distribution before computations). This process could be triggered based on some dynamic information, i.e. computation time and share time of each computational node. Thus, computational nodes, which have higher performance, could be dynamically assigned larger chunks and computational nodes with less performance — smaller chunks. One drawback of this method is that redistribution process takes time and will affect computational time. However, benefit of this method is the ability to distribute Yee grid points between computational nodes more efficiently without the need to identify optimal virtual topology before computations.

In developed solver dynamic balancing was implemented for 1D-X, 2D-X, 2D-Y, 3D-X, 3D-Y and 3D-Z virtual topologies. Before start of computations some virtual topology has to be chosen (machine learning or saved dynamic profile could be used to choose initial virtual topology in future). Then after M time steps, during which computational time T was gathered, redistribution is performed. Let's consider 2D-X case. Total size of grid is $a * b$, chunk of grid points, assigned to i $(i \geq 0, i < N)$ computational node has size $S_i = a_i * b$ and

$$\sum_i a_i = a \tag{12}$$

Performance of i computational node is calculated like this

$$perf_i = \frac{a_i * b * M}{T_i} \tag{13}$$

where T_i is computational time of i computational node for M time steps. Then, new a'_i is calculated

$$a'_i = \left\lfloor \frac{a * perf_i}{\sum_i perf_i} \right\rfloor \tag{14}$$

In case $a' = a - \sum_i a'_i > 0$, a' is spread between all computational nodes. After this procedure computational nodes will have chunks with new sizes, and distribution of computations between computational nodes will be better in terms of reduction of total computational time. This procedure could be repeated later to further improve distribution of computations.

Saving the Profiling Data. During first computation on the computational system profiling data could be saved for each computational node as a recommendation for virtual topology chooser. On the second launch, this saved data could be used to identify optimal virtual topology. Drawback of this method is the need to perform first computation with gathering of dynamic information in order to save it later. This process takes time and will affect computational time. However, starting from the second computation launch virtual topology will be chosen more optimally. Besides, even the second computation launch could also gather dynamic information and update the saved one. So, the benefit of this method is that after K computation launches with profiling $K+1$ computation launch will use the most optimal virtual topology from all, which could be chosen according to saved dynamic information. Besides, $K+1$ computation launch could be performed without profiling, thus, without performance degradation.

Some modification of this method is to perform benchmarking of computational system before computations, when profiling data is gathered not on some random computations but on one which is optimized to save more relevant dynamic data. Such a benchmark then has to be found. This method is to be discussed in detail in further work.

Machine Learning. Neural network could be used in this method, trained on characteristics of different computational systems and optimal virtual topologies. Then, before computation on some computational system optimal virtual topology will be found using trained neural network. Benefit of this method is the lack of need to do additional activities during computations because optimal virtual topology is identified before first computation on the system. However, drawback is that identified virtual topology could be not the most optimal even for well trained neural network. This method is to be discussed in detail in further work.

Table 1. Measurements for 2D mode for 4 computational nodes for Yee grid with size $a = 256$ and $b = 256$ and 10000 time steps.

Virtual topology	Value of $f(n)$	Execution time, seconds
2D-X with $n = 4, m = 1$	320	3015.16
2D-XY with $n = 2, m = 2$	256	3001.69

Table 2. Measurements for 2D mode for 4 computational nodes for Yee grid with size $a = 8192$ and $b = 8$ and 10000 time steps.

Virtual topology	Value of $f(n)$	Execution time, seconds
2D-X with $n = 4, m = 1$	2056	2605.45
2D-XY with $n = 2, m = 2$	4100	3192.83

Table 3. Measurements for 3D mode for 8 computational nodes for Yee grid with size $a = 64$, $b = 64$ and $c = 64$ and 1000 time steps.

Virtual topology	Value of $f(n, m)$	Execution time, seconds
3D-X with $n = 8, m = 1, k = 1$	5664	2106.54
3D-XY with $n = 4, m = 2, k = 1$	4032	2090.29
3D-XYZ with $n = 8, m = 1, k = 1$	3456	2069.23

Table 4. Measurements for 3D mode for 8 computational nodes for Yee grid with size $a = 4096$, $b = 8$ and $c = 8$ and 1000 time steps.

Virtual topology	Value of $f(n, m)$	Execution time, seconds
3D-X with $n = 8, m = 1, k = 1$	10368	1587.31
3D-XY with $n = 4, m = 2, k = 1$	16464	1815.29
3D-XYZ with $n = 2, m = 2, k = 2$	24624	2069.98

3 Measurements

All measurements were performed on IBM Blue Gene/P supercomputer for different virtual topologies. IBM Blue Gene/P is a massively parallel computational system. It contains 8192 calculation cores (2048 calculation nodes, 4 core each) with peak performance at 27.9 tflops. It supports both MPI and OpenMP technologies. Single calculation core is a PowerPC 450 with frequency at 850 MHz having 4GB of RAM. Communicational network is a three-dimensional torus and unites all the nodes. Single node has 6 bidirectional connections with 6 neighbors and throughput of each of these 12 connections is 425 MB/s. Blue Gene/P has GCC 4.2 compiler.

Basic FDTD computation was chosen as a benchmark (no PML, no TF/SF, point wave source for each computational node). In each computation virtual topology was mapped on computational nodes of Blue Gene/P in such a way that virtual topology matches physical topology, so, computational nodes, which are neighbors in virtual topology, will be neighbors in physical topology too, and no additional share expenses arise.

As tables 1, 2, 3 and 4 show, the smallest computational time is achieved with the virtual topology that is optimal for current grid size, and variation of computational times for different virtual topologies could be significant, varying from 0.5% to 18.4% for 2D mode and from 1.8% to 23.3% for 3D mode for different Yee grid sizes, which could be significant for long running tasks. These results depend on the size of Yee grid and variation could be even higher for larger grids. Besides, obtained results depend heavily on the target architecture, and for architectures where share operations are heavy in terms of time, results could be even more significant.

Measurements for dynamic balancing for two computational nodes for 2D mode and 2D-X virtual topology for Yee grid with size $a = 1000$ and $b = 1000$ were performed for two cases: 0 computational node has point wave source, nodes don't have point wave sources, i.e. the difference between two cases is the calculation of wave function for 0 node. After some execution time Yee grid appeared to be divided in chunks in the next way. In the first case, 0 node had chunk with size of 47% of total Yee grid size and 1 node had chunk with size of 53% of total Yee grid size. In the second case computational nodes had chunks of the same size. This proves that dynamic balancing allows to spread computations optimally even for homogeneous architectures, if each node has to perform different amount of computations.

4 Conclusion

Developed FDTD solver provides features for optimal computations distribution between computational nodes. Measurements prove described algorithm of choosing of optimal virtual topology for homogeneous architectures and that there is no one "silver bullet" virtual topology to choose for different Yee grid sizes. This allows solver to be more efficient in terms of computational time. Besides, dynamic balancing was shown to spread computations optimally through all computational nodes for homogeneous target architectures in case of different amount of computations on each computational node. In further work dynamic balancing would be improved for both homogeneous and heterogeneous target architectures and other dynamic methods would be described in detail.

References

1. Yee, K.S.: Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media. IEEE Transactions on Antennas and Propagation. Vol.14, No.3, pp. 303–307 (1966)
2. Zunoubi, M. R., Payne, J., Roach, W. P.: CUDA-MPI-FDTD implementation of Maxwell's equations in general dispersive media. Proceedings of the SPIE, Vol. 8221, id. 822115 (2012)
3. Zakirov, A.V., Levchenko, V.D., Perepelkina, A.Yu., Zempo, Y.: High performance FDTD code implementation for GPGPU supercomputers. Keldysh Institute Preprints, No.44 (2016)
4. He, B., Tang, L., Xie, J., Wang, X., Song, A.: Parallel numerical simulations of three-dimensional electromagnetic radiation with MPI-CUDA paradigms. Mathematical Problems in Engineering. Vol.2015, Article ID 823426 (2015)
5. Shams, R., Sadeghi, P.: On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters. Journal of Parallel and Distributed Computing, vol. 71, no. 4, pp. 584–593 (2011)
6. Parallel FDTD solver, `https://github.com/zer011b/fdtd3d`
7. Taflove, A., Hagness S. C.: Computational Electrodynamics: The Finite-difference Timedomain Method. Artech House, 3rd ed. (2000)