A Simple Method for Static Load Balancing of Parallel FDTD Codes

Franek, Ondrej

Published in:
Electromagnetics in Advanced Applications (ICEAA), 2016 International Conference on

DOI (link to publication from Publisher):
10.1109/ICEAA.2016.7731461

Publication date:
2016

Document Version
Accepted author manuscript, peer reviewed version

A Simple Method for Static Load Balancing of Parallel FDTD Codes

O. Franek[1]

Abstract—**A static method for balancing computational loads in parallel implementations of the finite-difference time-domain method is presented. The procedure is fairly straightforward and computationally inexpensive, thus providing an attractive alternative to optimization techniques. The method is described for partitioning in a single mesh dimension, but it is shown that it can be adapted also for 2D and 3D partitioning in approximate way, with good results. It is applicable to both homogeneous and heterogeneous parallel architectures, and can also be used for balancing memory on distributed memory architectures.**

1 INTRODUCTION

For about three decades, the computational power of microprocessors had been exponentially growing, thanks to the largely self-fulfilling Moore's law. Increasing clock frequencies and memory capacities had allowed electromagnetic simulations of electrically larger objects to be finished in less time. This was changed approximately 10 years ago, when problems with excessive heat dissipation forced the chip manufacturers to start deploying multiple cores instead of increasing clock rates. Software that wants to utilize the new multi-core architecture now has to be parallelized. Thus, an efficient parallel code, something that once was characteristic of supercomputing, has become a necessity for all implementations.

The well-known Yee algorithm of the finite-difference time-domain (FDTD) method [1] can be viewed as easily parallelizable. The discretized electric and magnetic fields of the computational domain form multidimensional arrays which are updated in each time step by means of nested loops, one for each dimension. Parallelization is achieved by splitting the loops and assigning computation of portions of the field arrays to separate threads running on separate processors (cores). This is fairly straightforward for basic FDTD operations, where all field components require the same amount of operations: the loops are divided uniformly so that each thread serves a portion of the field arrays having the same size. On ideal circumstances, the speedup of such multithreaded code will be approaching the number of threads.

However, in most realistic scenarios the FDTD algorithm contains extra features such as field sources, probe readouts, and, most importantly, dispersive and non-linear materials, which typically require additional numerical operations and auxiliary variables throughout the discretization mesh. As a result, those threads that serve the non-standard parts of the field arrays are loaded more and finish their work later, while the other threads have to wait, leading to less than optimal speedup.

We have faced this problem while performing calculations on elongated domains, simulating radio wave propagation along a wind turbine blade, using our in-house parallel FDTD code. To mitigate spurious reflections from the perfectly matched layer (PML) boundaries at shallow angles, we were forced to employ up to 50 cells deep PML, which led to significant load imbalance between segments partially or entirely filled with PML and those serving the regular portions of the FDTD mesh.

Load balancing algorithms can generally be categorized as static or dynamic [2]. In a typical FDTD simulation, distribution of materials, sources, probes and other features increasing the load is known before the simulation starts and does not change during the run, which allowed us to use static load balancing in this situation. Dynamic load balancing appears to be more complex to implement on a memory distributed system with message passing between the threads, mainly due to the fact that the information on the materials in the FDTD mesh needs to be redistributed when adjusting the segment dimensions. Parallel FDTD algorithms with load balancing have been reported [3–6], but mostly utilizing optimization techniques, which we considered as unnecessary given the nature of our problem.

In this contribution, we describe how we addressed the load balancing problem for parallel FDTD algorithm with 3D mesh segmentation without resorting to optimization. Being a fairly straightforward approach, it may prove useful to anyone writing parallelized FDTD code or other code having similar characteristics to FDTD—the formulation is quite general. To our best knowledge, this approach has not yet been published.

2 METHOD DESCRIPTION

A necessary condition for implementing the proposed

[1] APNet Section, Department of Electronic Systems, Faculty of Engineering and Science, Aalborg University, Niels Jernes Vej 12, DK-9220 Aalborg Ø, Denmark, e-mail: of@es.aau.dk, tel.: +45 9940 9837, fax: +45 9815 1583.

method is that the segmentation of the computational domain follows a mesh/torus parallel architecture, either homogeneous or heterogeneous. In the following, we describe the load balancing process in detail for 1D segmentation, and then discuss its application to 2D and 3D cases.

2.1 1D Segmentation

In this case, the entire computational domain is to be partitioned in S number of segments along single dimension. Let us assume that along this dimension, each slice of the FDTD mesh numbered $i = 1, \ldots, I$, with thickness of one cell is characterized by certain load density $l(i)$, corresponding to the relative computational load of all operations in the particular slice. Our objective is to find slice numbers of segment boundaries b_s, $s = 1, \ldots, S$, so that the total loads on all segments are balanced.

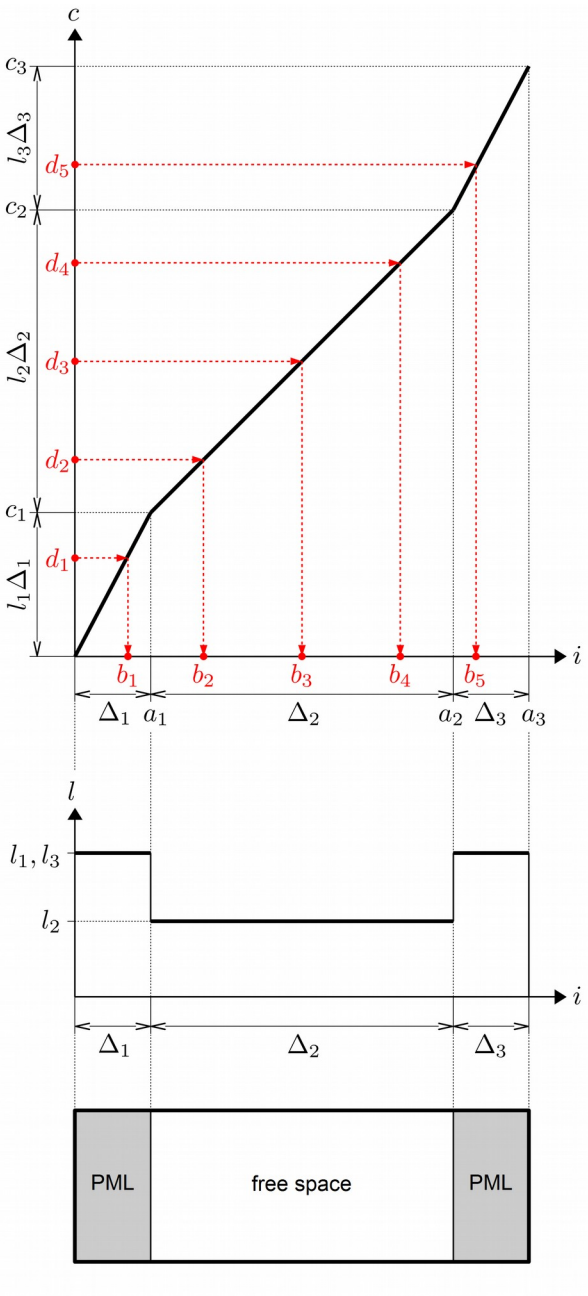

Figure 1: Example of an elongated FDTD domain with PML layers at each end, to be partitioned into $S = 6$ homogeneous segments.

Following a naive approach, we could choose

$$b_s = sS^{-1}I. \tag{1}$$

However, that would result in imbalance as the total loads of the segments

$$L_s = \sum_{j=b_{s-1}}^{b_s} l(j)$$

may have generally different values due variability in $l(i)$ along the mesh.

In order to achieve balanced loads we express the cumulative loads at each slice as a function of the i position

$$c = f(i) = \sum_{j=1}^{i} l(j)$$

so that the load between slices i_1 and i_2 can be expressed simply as

$$L(i_1, i_2) = c(i_2) - c(i_1)$$

From here it follows that if we want to partition the computational domain in terms of loads, we should do it in the cumulative load domain and then recalculate the segment boundaries in the mesh domain using the inverse function

$$i = f^{-1}(c).$$

In the common case that the cluster is homogeneous, we will divide the total load $c(I)$ equally into S segments and translate their boundaries

$$d_s = sS^{-1}c(I)$$

(compare with (1)) to the mesh coordinates

$$b_s = f^{-1}(d_s).$$

Since the cumulative load c is a function of a discrete variable i, the inversion will be accompanied by rounding to the nearest mesh coordinate.

For better illustration of the procedure, let us now assume a special, yet very common case of an FDTD domain with M continuous intervals of lengths Δ_m with constant load densities l_m throughout the intervals, where $m = 1, \ldots, M$. These intervals may represent volumes filled with PML layers, dispersive, or other media requiring more calculations per cell than the regular FDTD update equations. Our objective is to distribute portions of the mesh equally among S segments to achieve balanced loads. The

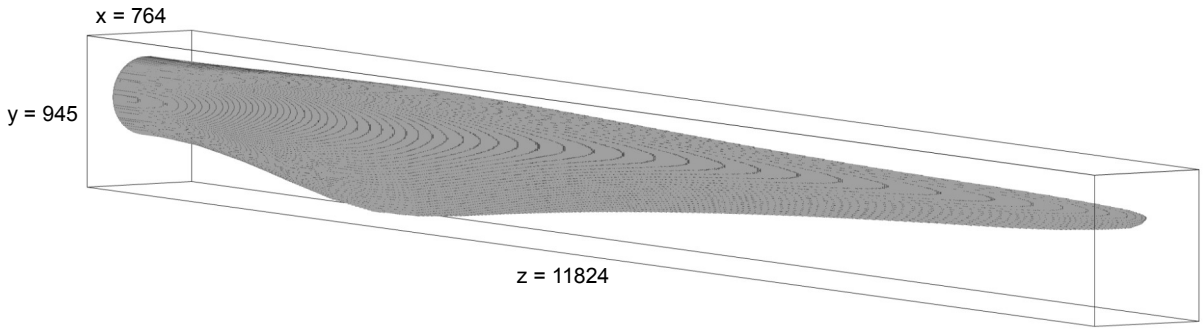

Figure 2: Elongated FDTD domain for simulation of wind turbine blade.

procedure is depicted in Fig. 1 for $M = 3$ volumes (regular FDTD mesh, $l_2 = 1$, with two PML layers at opposite ends, $l_1 = l_3 = 1.86$) to be partitioned into $S = 6$ homogeneous segments.

Given the definitions of the media lengths and load densities above, we can express the mesh positions of the media boundaries as

$$a_m = \sum_{j=1}^{m} \Delta_j$$

and the cumulative loads at the media boundaries can be calculated as

$$c_m = \sum_{j=1}^{m} l_i \Delta_i.$$

The next step is to divide the loads equally into segments, and so the cumulative loads at the segment boundaries will be

$$d_s = sS^{-1}c_M,$$

where c_M is the total load of the entire domain. After we find mapping from the segment indices s to the media indices m, $s \to m$, such that condition

$$c_{m-1} < d_s \leq c_m$$

is satisfied (assuming $c_0 = 0$), we can determine the mesh positions b_s between the segments with balanced loads from

$$b_s = l_m^{-1}(d_s - c_m) + a_m.$$

If any segment covers only one media volume with constant load density, then the size of that segment can be expressed by

$$b_s - b_{s-1} = l_m^{-1} S^{-1} c_M,$$

but this simple rule does unfortunately not apply to segments containing two or more media.

Comparing the naive and the balanced scheme, the best case of load reduction per segment between the two schemes is given by the ratio between the average and maximum load densities

$$r = \frac{l_{\text{avg}}}{\max l_m} = \frac{c_M}{a_M \max l_m},$$

where a_M is the total number of cells along the mesh. This is indeed only the best case, which occurs when a segment in the naive scheme is fully occupied by medium with the highest load density. Note that the speedup of the proposed method is reciprocal to the load reduction r.

2.2 2D and 3D Segmentations

If the FDTD code is intended to be deployed on distributed memory machine with message passing, it is advisable to partition the domain in all 3 spatial dimensions of the mesh, as this is expected to minimize the communication between the segments. Unfortunately, in the case of 2D or 3D mesh/torus arrangement of segments it is not generally possible to achieve perfect load balance between the segments and it might be necessary to resort to optimization techniques to find the minimum load.

However, we have experienced that applying the 1D procedure described above independently on each mesh dimension leads to segmentation that is not very far from the result obtained by optimization. For each dimension of the mesh, I will be the number of FDTD cells, and S will be the number of segments, along that particular dimension. Then, we can utilize the same procedure as described in section 2.1 and achieve sufficient speedup, even in the 2D or 3D case, while avoiding potentially expensive optimization.

3 EXAMPLE

Our simulation of ultrawideband signal propagation along a wind turbine blade (more details about the topic can be found in [7]) required a computational domain with size 764 × 945 × 11824 cells (see Fig. 2), plus 50 cells of PML in each direction, exceeding a total of 10 billion mesh cells. The domain was partitioned in 2 × 3 × 48 = 288 separate processes with distributed memory, each process having approximately cubic-shaped array partition to serve for better efficiency of message passing between the processes. For reasons of simplified addressing, the PML layers in our FDTD code are

positioned at the upper end of all three coordinates, with wraparound (periodic) boundaries between them. We measured the time required to calculate one time step as the minimum across ten first time steps and three independent attempts, to even out fluctuations caused by the computing nodes.

With uniform partitioning, the last segment had dimensions 432 × 348 × 248 cells, and one time step took 5.71 s. The proposed load balancing method gave 389 × 291 × 164 as the size of the last segment, with the remaining domain partitioned uniformly, which was exactly the same result as obtained from optimization. The time for calculating one step was then reduced to 4.47 s, ie. by 22 %.

Since the simulation required 21000 time steps, the original total simulation time of approx. 33 hours has been cut by more than 7 hours. This clearly demonstrated usefulness of the proposed method.

4 CONCLUSION

It has been shown that the proposed method of balancing the load for parallel FDTD method gives significant improvement in running times and its results compare well with optimization, while being straightforward and computationally inexpensive procedure. Although tested only on homogeneous architecture, the formulation is general enough to be equally applicable to heterogeneous architectures. Finally, it should also be noted that the method is not limited to balancing the computational load—it may as well be used to balance memory requirements on distributed memory parallel architectures.

Acknowledgment

This work was supported by the Innovation Fund Denmark (InnovationsFonden) project of "Intelligent Rotor for Wind Energy Cost Reduction" (project code 34-2013-2). The author also gratefully acknowledges the support from the Danish e-Infrastructure Cooperation (DeIC) for the Linux cluster "Fyrkat" at Aalborg University, Denmark.

References

[1] A. Taflove, and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, Artech House, 3rd ed., 2005.

[2] C. Xu, and F.C. Lau, *Load Balancing in Parallel Computers: Theory and Practice*, Springer Science & Business Media, 1996.

[3] S.A. Seguin, M.A. Cracraft, and J.L. Drewniak, "Static and quasi-dynamic load balancing in parallel FDTD codes for signal integrity, power integrity, and packaging applications," in *IEEE International Symposium on Electromagnetic Compatibility*, vol. 1, Aug. 2004, pp. 107–112.

[4] H. Wang, A. Trakic, L. Xia, F. Liu, and S. Crozier, "An MRI-dedicated parallel FDTD scheme," *Concepts in Magnetic Resonance Part B: Magnetic Resonance Engineering*, vol. 31, no. 3, 2007, pp. 147–161.

[5] R. Shams, and P. Sadeghi, "On optimization of finite-difference time-domain (FDTD) computation on heterogeneous and GPU clusters," *Journal of Parallel and Distributed Computing*, vol. 71, no. 4, 2011, pp. 584–593.

[6] N. Zakaria, A.J. Pal, and S.N.M. Shah, "Stagewise optimization of distributed clustered finite difference time domain (FDTD) using genetic algorithm," *International Journal of Innovative Computing, Information and Control*, vol. 9, no. 6, June 2013, pp. 2303–2326.

[7] O. Franek, S. Zhang, T.L. Jensen, P.C.F. Eggers, K. Olesen, C. Byskov, and G.F. Pedersen, "Wind Turbine Blade Deflection Sensing System Based on UWB Technology," in *International Conference on Electromagnetics in Advanced Applications (ICEAA)*, 2016. (submitted)