JOURNAL OF APPLIED
COMPUTER SCIENCE
Vol. 24 No. 2 (2016), pp. 7-16

Parallel FDTD Simulation Using Task Parallel Library (TPL)

Ivan Bolesta[1], Antonina Demchuk[2]

Ivan Franko National University of Lviv
Department of Radiophysics and Computer Technologies
Generala Tarnavskogo Str. 107, Lviv, 79017, Ukraine
[1]*bolesta@electronics.lnu.edu.ua*
[2]*demchuk.antonina@gmail.com*

Abstract. *The finite-difference time-domain (FDTD) is a numerical analysis technique used for solving computational electrodynamic problems. The nature of the FDTD method is that simulation of big and complicated electromagnetic field problems requires a vast amount of computer operational memory and runtime. Parallel-processing techniques have been broadly applied to FDTD to accelerate the simulations.*
The parallelism of the FDTD algorithm is based on a fact that the computational domain can be divided into parts (sub-domains), and each processor in a parallel system deals with one or several sub-domains. The FDTD algorithm belongs to data parallelism model and can be effectively implemented on shared memory system architecture.
The parallel FDTD method was implemented using TPL library. The Task Parallel Library (TPL) is a library for .NET that makes easy to parallelize the program using the advantages of .NET Framework. The speedup metrics of parallel FDTD algorithm were calculated and compared with Amdahl's estimated speedup.
Keywords: *FDTD, Yee cell, data parallel model, shared memory system, task, speedup, Amdahl's law.*

1. Introduction

The Finite Difference Time Domain (FDTD) method, introduced by Yee, in 1966 [1], is one of the most popular three-dimensional methods in computational electromagnetics. The applications of FDTD in electrodynamics include antenna and radar design, electronic and photonic circuit design, microwave tomography, cellular and wireless network simulation, mobile phone safety studies, and many more [2].

The FDTD technique is a direct implementation of Maxwell's time-dependent curl equations to solve the temporal variation of electromagnetic waves within a finite space that contains an object of arbitrary geometry and composition. In practice, the space is discretized by a grid mesh, and the existence of the scattering particle is defined by properly assigning the electromagnetic constants including permittivity, permeability, and conductivity over the grid points. The Maxwell curl equations are subsequently discretized by using finite-difference approximations in both time and space.

The main advantages of this method lie in several aspects. Firstly, the whole frequency band can be obtained by only one calculation in time-domain. Secondly, simplicity of explicit numerical scheme of FDTD. FDTD method can easily model complex objects. Finally, the necessary operational memory is relatively smaller than used in other low frequency numerical techniques.

Since FDTD requires the entire computational domain to be discretized by a grid mesh, the grid discretization must be granular enough to resolve both the smallest electromagnetic wavelength and the smallest geometrical feature in the model. As a result, FDTD simulations require significant computational resources both in terms of memory and time execution. Parallel-processing techniques [3] have been broadly applied to FDTD to accelerate the simulations.

FDTD method is easy to implement because the Yee scheme is explicit. At each time-step updating a Yee cell (see figure 1), requires values of the field components of a given cell and its neighboring cells in the previous time step. In a more general settings, where higher order discretization or nonlinear wave equations are involved, calculation may require field values from several previous time steps. Regardless of the complexity of the wave equations, FDTD ensures that each cell update is independent of its neighbors in the current time-step. This makes FDTD computations be well suited to parallelization [4].

The parallelism of the FDTD algorithm is based on a decomposition of the grid mesh into contiguous nonoverlapping subdomains. The computational space

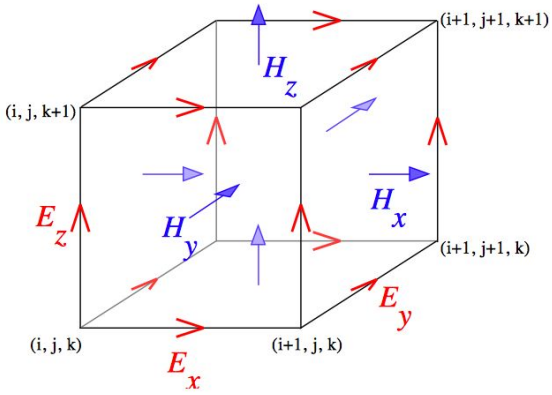

Figure 1: Yee cell with electric and magnetic field vector components distribution

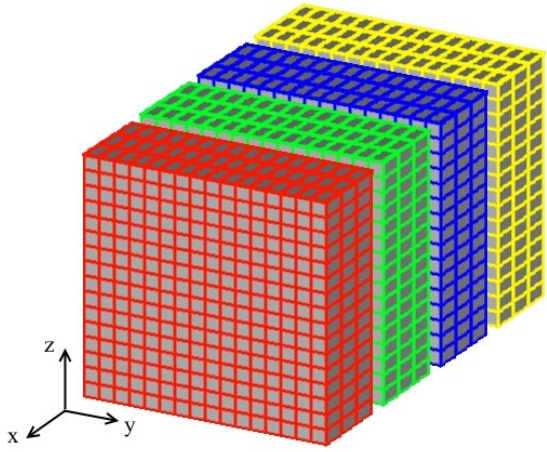

Figure 2: 1D domain decomposition example in X direction using 4 processors

can be divided into parts along the three directions, and each processor in a parallel system deals with one or several subdomains. The configuration that divides all computational mesh in X direction is shown in figure 2.

2. Parallel computing paradigm

The parallelism model is a strategy for distributing the data among processors and applying operations to reduce interactions. There are two strategies for partitioning work among processors: data parallelism and task parallelism [5].

In data parallel model, program performs the same operation to different items of the data set at the same time. The data set is typically organized as an multi-dimensional array. The primary characteristic of data-parallel model problems is that the intensity of data parallelism increases with the size of the problem, which makes possible to use more processors to solve larger problems.

Task parallelism is achieved when each processor executes a different operation on the same or different data. The processors may execute the same or different code. In the general case, different execution processors communicate with one another as they work. Communications usually take place by passing data from one processor to the next as part of a workflow.

From the description of FDTD algorithm, we deal with gridded computational domain, that is represented as multi-dimensional arrays of data. At every time step we need to update fields based on the formula, so the most suitable parallel model for FDTD is data parallel model.

Parallel computing [6] also depends on how the processors are connected to the memory. The way of such connection can be classified into shared or distributed memory systems.

In shared memory system, a single address space exists, within it every memory location is given a unique address and the data stored in memory are accessible to all processing cores. Therefore, in order to enforce consistency, it is necessary to use synchronization.

In distributed memory system, each processor has its own memory and can only access its local memory. The processors are connected with other processors via high-speed communication links. The processors communicate with others through passing messages using MPI (Message Passing Interface) standard [3].

The memory access efficiency is one of the key components contributing to the performance of the FDTD method on high performance computing platforms [7]. The data required by the FDTD process is supplied either from memory directly or from the communication with other FDTD threads.

FDTD algorithm require a small amount of shared memory access when implemented in parallel [8]. On a multi-core processor system, this communication can be done very quickly, since each core has direct access to the same memory.

3. Parallel constructs in .NET

The .NET Framework has several libraries to support parallel programming. The first versions .NET Framework provide the Threading library which contains low-level synchronization constructs for building concurrent programs. Thread is the main class for implementing concurrent computations, and ThreadPool allows to reuse threads in efficient way.

In the .NET Framework 4.0 the new introduced Task Parallel library (TPL) is based on the high-level concept of a task. Task represents a single operation that usually executes asynchronously. Using tasks instead of threads has many benefits [9] - not only tasks are more efficient, they also abstract away from the underlying hardware and the OS specific thread scheduler.

The TPL has a static Parallel class that supports parallel loops with For and ForEach methods, and structured fork-join tasks with Invoke method [10]. The most basic parallel loop requires invoking Parallel.For with three arguments.The first two arguments specify the start inclusive and end exclusive index of iteration domain, and the third argument is a C# lambda function called for each index in the domain.

$$Parallel.For(from, to, index => ProcessTask(index)); \quad (1)$$

.NET 4.0 also provides the Parallel Language-Integrated Query (PLINQ) library, which supports parallel execution of LINQ to Objects expressions. PLINQ queries operate on IEnumerable objects by calling AsParallel(). After the AsParallel, the data is partitioned to worker threads, and each worker thread executes in parallel the following Where, Select, and ForAll expressions.

In comparison with other parallel libraries, TPL has many advantages: (1) parallel code is easy to debug, (2) library has many synchronization primitives to support concurrency, (3) parallel library and .NET code itself supports handling exceptions, (4) parallel settings are easy to adjust (e.g. to set maximum number of parallel threads), (5) possibility to break parallel execution and proceed with next task, (6) effective usage of low-level threads.

The disadvantages of library are: (1) need to rewrite program to apply parallel execution, (2) program execution in general can be a little slower in comparison to the same program written in a low-level programming language, (3) need for .NET framework on the running machine.

Based on the specifics of FDTD algorithm, especially the fact that computa-

tional domain is presented as a set of multi-dimensional arrays and low number of reduce operations, usage of Parallel class with Parallel.For method is efficient [11].

4. Performance metrics

To analyze parallel performance of the program, the speedup metrics are used. The speedup is defined as the ratio of time that is taken by a sequential algorithm to time that is taken by a parallelized version of the same algorithm. The speedup of a program using multiple processors in parallel computing is also limited by memory performance, communication time between processors and the time needed for the sequential fraction of the program.

Amdahl's law serves as a way of determining whether an algorithm is a good candidate for parallelization.

Amdahl's law is an approximation of the maximum theoretical speed up that parallelized program can achieve on a multiprocessor system. The speedup is measured the following way. An algorithm is assumed to consist of two components: one that will operate in serial form and one that can be parallelized. Even if the parallel component of the algorithm could speed up to take close to no time at all, the program execution would still need to execute the finite serial component.

According to Amdahl's law [12], the estimated speedup in an improved sequential program, where some part was speed up p times (through processing on p cores) is limited by inequality:

$$S_{max} \leq \frac{p}{1 + f(p-1)}, \tag{2}$$

where f is the fraction of time spent for executing serial part of a program (part that cannot be parallelized) in total time spent for executing not improved program.

As the number of processing cores used in the parallelization increases to be much larger than one, the Amdahl formulation above indicates that the parallel process achieves very little speedup. This is reflected by the near horizontal component of the speedup curve shown in the theoretical processing time.

In practical terms, Amdahl's metrics are also limited by the effects of computing hardware used to parallelize the algorithm. Amdahl's law does not cover two issues: it does not take into account the problem size and, as result, memory size;

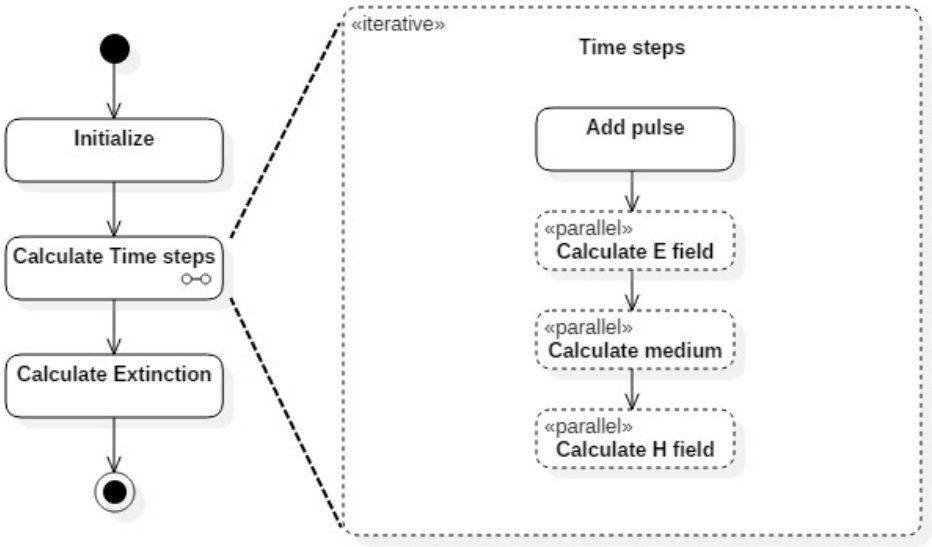

Figure 3: Activity diagram of parallelized FDTD algorithm

it ignores the cost of communication/synchronization operations associated to the introduction of parallelism into an application.

5. Experimental results

The diagram of the parallel FDTD algorithm using TPL is shown on figure 3. Three parallel loops presented in the algorithm calculate electric field values based on the adjacent magnetic fields and vice versa. Each iteration performs at least one electric and then one magnetic calculation as required by the FDTD domain, but adjacent values remains the same within parallel loop.

The disadvantage of such computation is the time need to wait all task completion before proceed with the next parallel loop, that can be time consuming especially if domain partitions do not execute for nearly the same time intervals due to different numbers of medium points in each partition. Therefore, the main advantage of this version of parallel algorithm is that there is no need to use synchronization primitives and wait for access to field values [13].

The figure 4 shows estimated speedup, calculated using Amdahl's law, with serial fraction of f, and actual speedup of parallelized FDTD algorithm using Task

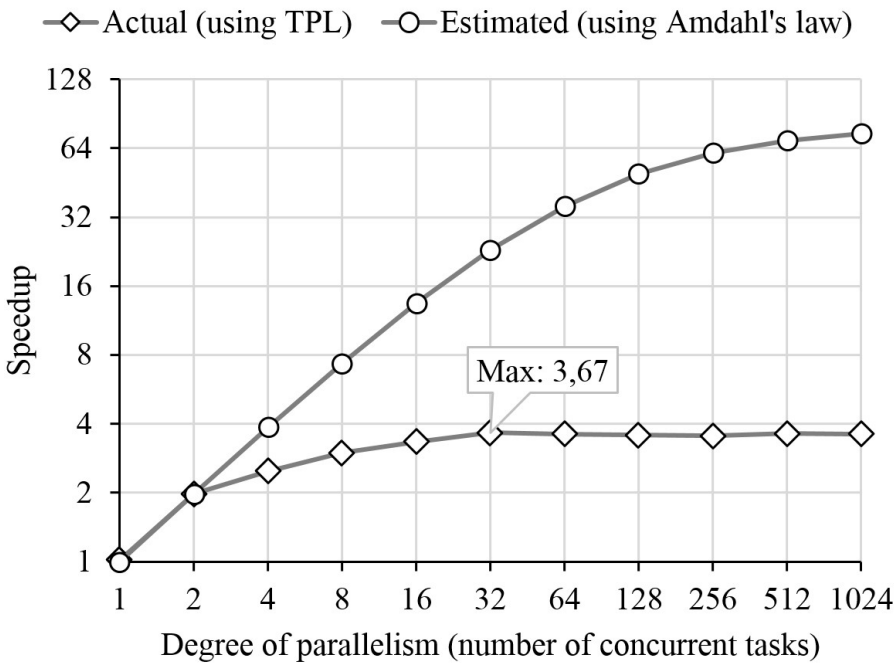

Figure 4: Comparison of estimated and actual speedup of FDTD algorithm

Parallel Library. The problem size of the FDTD simulation include number of time steps (100), the size of computational space (50*50*50 cells) and number of frequencies of interest (100). The experiment was held on Intel® Core™ i5-4590 processor with four cores without HyperThreading.

The actual speedup corresponds to estimated at degree of parallelism equals 2. The following speedup rates differ from the estimated and continue to increase up to the maximum value of 3.67 for 32 number of tasks. The subsequent increment of parallelism degree does not give a rise in speedup.

The difference between estimated and actual speedup can be explained by the Amdahl's law constraints, especially communication and wait operations, non-uniform computation time within the 3D grid, the limit number of cores (4 cores for the experiment), memory performance (e.g. reading and writing from shared arrays, automatic garbage collecting) etc.

6. Conclusions

The most suitable parallel model for FDTD method is data parallel model based on a shared-memory system architecture. The .NET framework gives a possibility to execute algorithm in parallel and have maximum control in maintenance and performance diagnostic of a program. The actual speedup of parallel FDTD algorithm is limited to the number of processor cores, but corresponds to the estimated speedup by Amdahl's law in qualitative characteristics.

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, 1966, pp. 302–307.

[2] Taflove, A. and Hagness, S., *Computational Electrodynamics: The Finite-Difference Time Domain Method*, Artech House, Norwood, 2006.

[3] He, Z. L., Huang, K., Zhang, Y., Yan, Y., and Liang, C. H., *Study on High Performance of MPI-Based Parallel FDTD from WorkStation to Super Computer Platform*, International Journal of Antennas and Propagation, Vol. 2012, 2012, Article ID 659509, 7 pages.

[4] Guo, X. M., Guo, Q. X., Zhao, W., and Yu, W. H., *Parallel FDTD simulation using NUMA acceleration technique*, Progress In Electromagnetics Research Letters, Vol. 28, 2012, pp. 1–8.

[5] Sen, R., *Developing Parallel Programs*, Advances in Computer Science: an International Journal, Vol. 1, No. 1, 2012, pp. 18–27, Available at: http://www.acsij.org/acsij/article/view/321.

[6] Barney, B., *Introduction to Parallel Computing*, Lawrence Livermore National Laboratory, 2015, Available at: https://computing.llnl.gov/tutorials/parallel_comp/.

[7] Demir, V. and Elsherbeni, A. Z., *Compute Unified Device Architecture (CUDA) based Finite-Difference Time-Domain (FDTD) implementation*, ACES Journal, Vol. 25, No. 4, 2010, pp. 303–314.

[8] Ujaldon, M., *Using GPUs for accelerating electromagnetic simulations*, ACES Journal, Vol. 25, No. 4, 2010, pp. 294–302.

[9] *Task Parallel Library (TPL)*, 2012, Available at: https://msdn.microsoft.com/en-us/library/dd460717(v=vs.110).aspx.

[10] Leijen, D., Schulte, W., and Burckhardt, S., *The Design of a Task Parallel Library*, In: Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA '09, ACM, 2009, pp. 227–242.

[11] Vištica, M., Haseljić, H., Maksumić, A., and Nosović, N., *Comparison of speedups for computing π using .NET TPL and OpenMP parallelization techonologies*, In: X International Symposium on Telecommunications (BIHTEL), 2014.

[12] Amdahl, G. M., *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*, In: Proceedings of the April 18-20, 1967, Spring Joint Computer Conference, AFIPS '67 (Spring), ACM, 1967, pp. 483–485.

[13] Zhang, K. D., Shao, W., and Wang, B. Z., *Parallelized ADI-FDTD Algorithm for Attenuation Constant Extraction by Using OpenMP Library*, In: International Conference on Microwave and Millimeter Wave Technology, IEEE, 2010, pp. 786–788.