PARALLEL FDTD-BASED RADAR CROSS SECTION SIMULATIONS

Mustafa Çakır[1], Gonca Çakır[1], Levent Sevgi[2]

[1]*Kocaeli University, Electronics and Communication Engineering Department,*
[2]*Doğuş University, Electronics and Communication Engineering Department,*
Zeamet Sok. No.21, Acıbadem/Kadıköy, İstanbul, Turkey
Email: gonca@kou.edu.tr

Key words: Parallel processing, parallel computing, computational electromagnetics, time-domain simulations, FDTD, RCS, antennas, near fields, far fields, DFT, FFT.

ABSTRACT

FDTD-based radar cross section (RCS) and antenna simulations is discussed. The main FDTD volume is left as is in the classical FDTD method, but the task of the near-to-far-field transformation (NTFFT) module is distributed. Tests with different scenarios show that a computation gain of 3 to 7 may be reached with this approach.

I. INTRODUCTION

Parallel processing in computers is based on dividing a computer code into a number of sections and distributing the task among a number of computers/processors, all of which are executed in parallel. This may be achieved on hardware-level, software level, or both. Hardware-level parallellization necessitates alternative processor designs. Software-level parallellization can be either on data level or function level, depending on the characteristics of the code.

FDTD is one of the most attractive and suitable methods for the parallelization [1]. There are many FDTD-Based parallelization studies in the literature [2-5], all of which deal with the main FDTD volume. In this study, a novel parallelization approach for FDTD-based RCS and antenna simulations is introduced. FDTD-based RCS prediction and radiation pattern plots require far field simulations. Unfortunately, FDTD simulations are performed in a discrete 3D finite volume therefore, a kind of a near-to-far-field transformation (NTFFT) is essential during the time-domain computations [6]. Most of the FDTD simulators developed for RCS and antenna problems have three main modules; the 3D FDTD module itself, the open-boundary simulators (OBS), and the NTFFT routine [7].

II. STANDARD FDTD-RCS ALGORITHM

A 3D finite physical volume is reserved for the discrete models in FDTD method and Maxwell equations are run in discrete iterative form. The standard FDTD procedure for the RCS and antenna simulations [7-10] is as follows:

- Discrete target or antenna model is located at the center of the 3D FDTD volume and initial parameters are set.
- A broadband source is injected. It is a pulsed voltage for the antenna, but a pulsed Gaussian plane wave for the RCS simulations.
- At each time step, all electric and magnetic field components inside the 3D FDTD volume is calculated using 6 iterative E/H equations.
- Far-fields are extrapolated along a number of chosen directions using surface electric and magnetic currents over this virtual surface [6].
- The NTFFT is based on the accumulation of the far zone vector potentials due to the tangential electric and magnetic fields on a virtual, closed surface surrounding the object under investigation at each time step.
- The NTFFT process is repeated for every other angular direction. For example, if an angular radiation or RCS pattern with 1° angular resolution is specified, the NTFFT procedure is repeated for 361 different directions at every time step.
- At the end, off-line DFT is applied for a given number of frequencies, and RCS or radiation patterns are obtained.

A novel FDTD-based RCS prediction virtual tool is presented in [11,12]. It contains a powerful tool which creates discrete models from graphics file images. Any kind of a target with non-penetrable boundaries can be created using basic blocks such as rectangular prism, cone, cylinder, sphere, etc. Moreover, a collection of pre-designed surface and air targets stored in 3DS (3D Studio file, visit www.autodesk.com for details) format files are also supplied. The virtual tool creates discrete FDTD and MoM models from either 3DS files or user-created objects. A typical discrete FDTD model is given in Fig. 1. Its bi-static RCS pattern is given in Fig. 2. The plot corresponds to vertical scan and curves are normalized to 30 dB (RCS_{max} is given in inset).

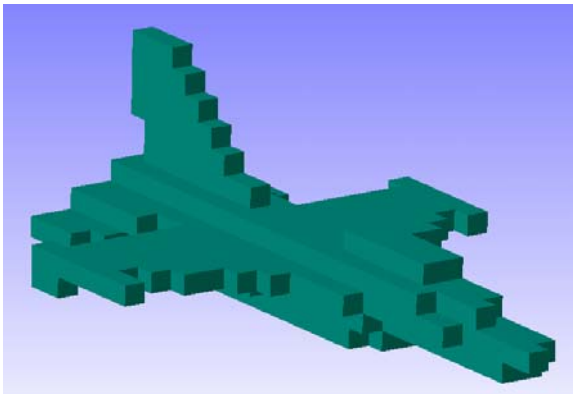

Figure 1: Discrete 15 m-long F16 model

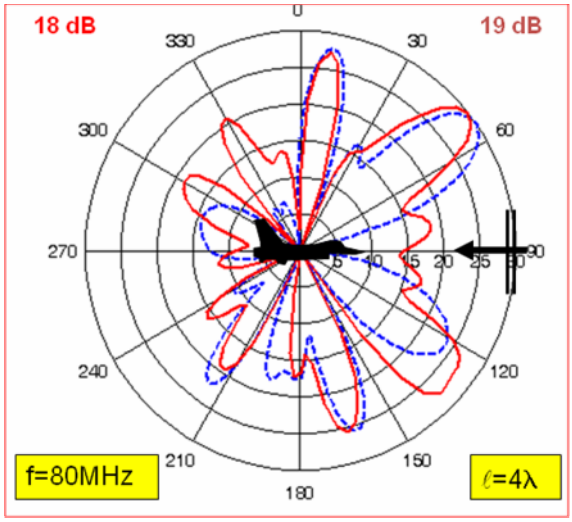

Figure 2: An angular bi-static RCS pattern (f=80 MHz, θ_i=90°, φ_i=90°, φ_s=90°, 0°≤θ_s≤360°, $\Delta\theta$=3°, Vertical scan, $\sigma_{\varphi\varphi}$-case), Solid: FDTD, Dashed: MoM

III. THE PARALLEL FDTD ALGORITHM

Function-level parallellization is used in this study [13]. The new parallel-FDTD algorithm has 7 blocks. The first block is responsible for the initializations, main 3D FDTD computations, as well as the termination. It also includes off-line DFT process and output presentations. The other 6 blocks are reserved for the parallellization of the NTFFT procedure. Each of these 6 blocks is reserved for the far field extrapolation on each of 6 facets of the virtual NTFFT closed surface. The algorithm for single program multiple data (SPMD) structure is presented in Fig. 3. On the left, the time-loop of the standard FDTD method is given. On the right, the flow chart of the novel parallel FDTD method is shown. The unique SPMD executable code is copied onto all 7 nodes. Each node executes a block having a block ID, same as the node ID. Parallellization may be achieved alternatively via multiple program multiple data (MPMD) using seven different codes for the seven nodes. Obviously, MPMD has the advantage of writing down small-sized codes without brunching with ID tags inside a unique code. Its disadvantage is that one needs to write down a different code for each processor.

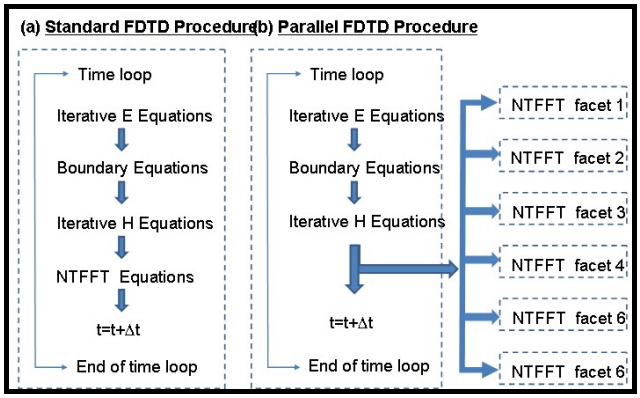

Figure 3: Flow charts of (a) classical FDTD procedure, (b) the novel parallel FDTD approach. The idea is to run FDTD and NTFFT modules simultaneously and distribute NTFFT burden to parallel processors.

The NTFFT is naturally suitable for the parallellization if its task is distributed onto 6 processors; each extrapolating far fields on a single facet of the virtual NTFFT closed surface along a number of directions separately. One aim is to increase the numerical computation performance up to an optimum level with available computer resources. Therefore, a 7-node parallellization not only speeds up the computations but also extends range of applicability. The parallel FDTD algorithm is designed in a way so that one can use computers, for example, in a student lab, or computers of roommates connected as a message-based network [13].

A sample task progress report of the 7-block parallel FDTD code is pictured in Fig. 4. This figure belongs to a scenario labeled as S3-D1 (see Table 1). Here, 7 horizontal strips correspond to 7 processor nodes. The strip on the top corresponds to the main FDTD node, while the others to 6 NTFFT nodes. Dark and light gray portions on each strip specify busy and idle statuses, respectively. Start of the parallel FDTD computation is marked as $t = 0$. Each cycle in the FDTD time loop corresponds to Δt seconds. The first two time cycles are also marked. At the end of each cycle the FDTD node passes newly calculated E/H data to 6 NTFFT nodes. This is a forward data transfer. After the code runs for a specified number of cycles (i.e., when $t \geq N\Delta t$, N being the number of user-specified time steps) the NTFFT data from 6 nodes are backward transferred to the main FDTD node. Off-line DFT calculations are then performed and polar RCS diagrams are plotted.

Obviously, optimum performance is obtained when all 7 horizontal strips are busy (i.e., when all the nodes are running) at the same time. This means all 7 processors have almost the same computation burden for each time

cycle; nobody has to wait nobody. In Fig. 4, it is observed that the main FDTD node runs nonstop. It only waits for a while for data transfer from 6 NTFFT nodes at the end. Observe that NTFFT nodes are busy for almost 85-90 %

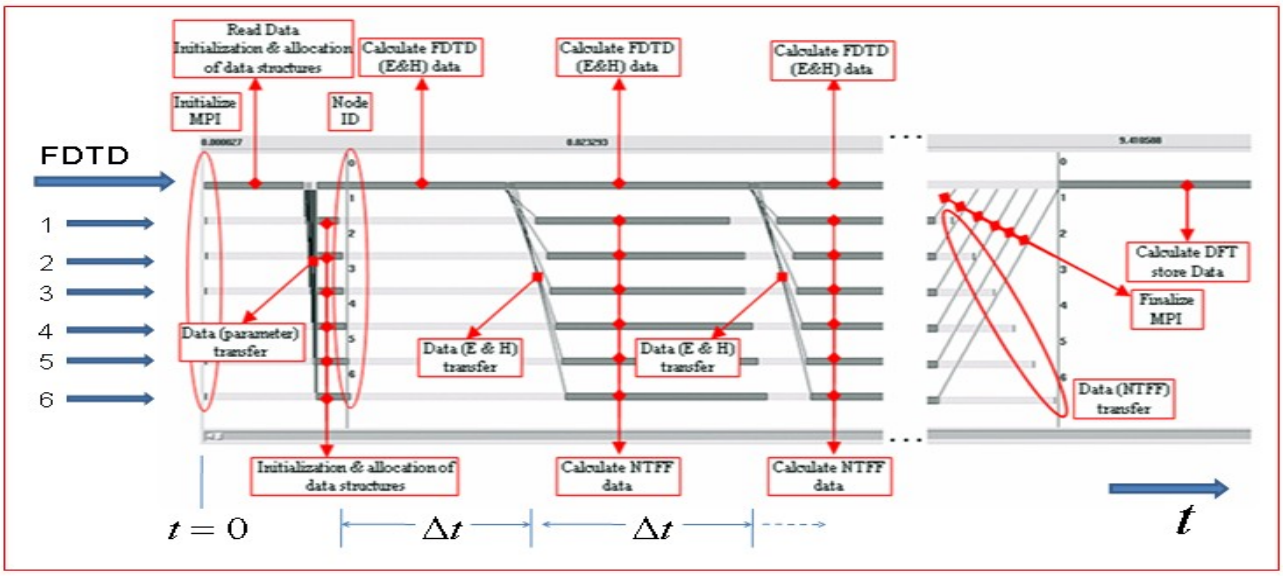

Figure 4: Task progress report of the parallel-FDTD code. Horizontal axis shows the time progress. Seven horizontal strips represent statue of seven blocks. Dark and light gray sections of the strips correspond to durations of computing and waiting, respectively. Lines represent data transfer instants from the main FDTD block to 6 NTFFT blocks.

of each cycle. Two other task progress reports are given in Fig. 5. On the top, it is shown that 6 NTFFT nodes run for a while (less that 15-20% of a time cycle) and then waits for a long time for the E/H data of the next cycle. Note also that, computation times of all 6 NTFFT nodes are different. Similar observations are valid for the task progress report at the bottom. This is the case when computation burden of the main FDTD node is much more than the NTFFT nodes.

IV. TYPICAL SCENARIOS AND TESTS

A few RCS prediction scenarios are designed to test the novel parallel-FDTD code [13]. These are listed in Table 1.

Table 1: Three typical RCS scenarios

Scenario		FDTD Parameters					NTFFT	
ID	Label	# of iteration	Nx	Ny	Nz	Δ [m]	Δφ[°]	M
1	S1-D1	600	76	80	47	3	1	361
	S1-D5						5	73
2	S2-D1	600	166	180	79	1	1	361
	S2-D5						5	73
3	S3-D1	250	41	41	41	0.01	1	361
	S3-D5						5	73

Symmetric Multi-processor (SMP) Parallelization

The first analysis belongs to multitasking and is performed on a single PC, using *threads* in the Windows environment. Main block is responsible for the FDTD iterations, and final off-line DFT calculations. The *threads* 1 to 6 perform the NTFFT at each time step. *Threads* 1 and 2 perform NTFFT at *x=constant* surface (left and right yz-facets), *threads* 3 and 4 perform NTFFT at *y=constant* surface (front and back xz-facets), and *threads* 5 and 6 perform NTFFT at *z=constant* surface (top and bottom xy-facets).

Two different PCs are used in these tests; an Asus Laptop and an IBM Server. The Asus laptop has F3JC series 1 Intel *Core2Duo* processor T5600 with 1.83 GHz CPU and 1 GB RAM Memory. IBM Server has 2 *DualCore* Intel Xeon processors with 3.00 GHz CPU and 4 GB RAM memory. The results are given in Fig. 5 for the three scenarios and with both $\Delta\varphi = 5^\circ$ and $\Delta\varphi = 1^\circ$ angular resolutions. Vertical axis corresponds to the parallelization gain. Note that, gain of the standard FDTD simulations with the same parameters is "1". The number of NTFFT applications for these two cases are 73 and 361, respectively. This means the NTFFT computation burden for 1° resolution is almost 5 times more than the burden of 5° resolution test. As observed in Fig. 5, the efficiency of the parallellization for the 4-processor IBM is higher than the efficiency of the 2-processor Asus PC. Maximum gain is 3.25 and is obtained for the S1-D1 scenario. The conclusion of this first test is that, even on a single SMP platform (i.e., a laptop having almost standardized configuration of *core2duo* processor) a significant gain may be obtained via *thread*-based multitasking.

Message Passing Interface (MPI) Parallelization

Parallellization performance tests with multi computers are carried out for homogeneous and heterogeneous

computers. Fourteen computers used during these tests and their technical specs are different. The first 7 PCs (labeled as PC-1 to PC-7) are terminals in a lab, have the same technical specifications: 667 MHz CPUs and 128 MB RAM memories. PC-8 is an IBM Laptop with 2.4 GHz CPU and 512 MB RAM memory. The other PCs from PC-9 to PC-14 are the desktops of colleagues in next doors in a corridor of the EE department of Kocaeli University. The tests are performed on two different platforms which are prepared using Linux Slackware and Parallel-Knoppix versions.

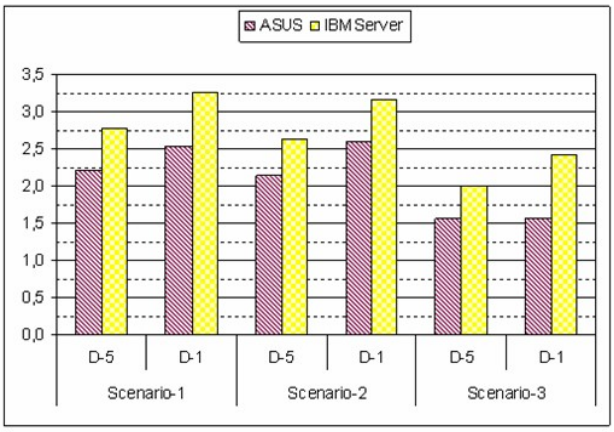

Figure 5: Parallel FDTD run using threads on a single PC in the Windows environment: Asus vs. IBM Server. Vertical axis shows the computation gain. The gain of the standard FDTD is "1".

Although MPMD programming has the disadvantage of writing a module for every node, overall, the codes are much shorter and simpler when compared to SPMD. Plus, MPMD does not need control modules. Tests performed with MPMD and SPMD programming showed that there is no significant gain difference in the parallel-FDTD simulations (at least, for the scenarios and parameter sets used in this paper).

The first MPI-based parallellization test is performed on a platform where PC-8 is reserved for the FDTD node, and, PC-9 to PC-14 for 6 NTFFT nodes. Scenario-1 with both S1-D5 and S1-D1 cases is taken into account. The results of this heterogeneous test are given in Table 2. Table 2a shows the computation times for the standard FDTD simulations. The time for the main FDTD node and 6 NTFFT nodes are also given separately. The results of parallel-FDTD tests performed on the platforms using Linux Slackware and Parallel-Knoppix are listed in Table 2b. The PCs used for the main FDTD node and 6 NTFFT nodes are listed separately in the table. As observed, there is no significant performance difference between the two platforms. It is interesting to note that although D1 NTFFT is 5 times more than D5 NTFFT calculations the difference between these two is less than 15%. But, if the values of Table 2b are compared against the values in Table 2a, it is seen that parallellization gain for the scenario S1-D1 is much higher than the scenario S1-D5.

Table 2a: Standard FDTD performances for Scenario-1

		PC 8 [s]	*PC 9-11 [s]*	*PC 12 [s]*	*PC 13 [s]*	*PC 14 [s]*
D-5	*FDTD*	76.480	74.930	69.100	53.560	67.940
	NTFF	79.090	76.910	70.100	73.660	52.310
	Total	**157.550**	**153.420**	**140.660**	**128.220**	**121.640**
D-1	*FDTD*	76.360	75.000	69.140	53.550	68.810
	NTFF	371.390	364.300	331.670	351.670	248.150
	Total	**457.210**	**446.860**	**407.780**	**410.040**	**323.380**

Table 2b: Parallel FDTD performances for Scenario-1

Scenario -1	*BProc-Slackware*			*Rsh-Knoppix*	
	PC 8: 9-14		*PC 8: 12-14*	*PC 14: 8-13*	
	SPMD [s]	*MPMD [s]*	*MPMD [s]*	*SPMD [s]*	*MPMD [s]*
D-5	98.204	97.623	97.833	80.785	80.654
D-1	108.269	107.875	108.160	101.051	101.798

It should be noted that the parallellization is applied only to the NTFFT routine and the main FDTD routine is left as it is in the standard FDTD case. Therefore, the computation time of the parallel-FDTD simulations can not be less than the time of the standard-FDTD (without NTFFT). Therefore, the computation gain comes from the NTFFT parallellization. The average NTFFT time for the S1-D5 scenario is nearly 72 s (see Table 2a). If the NTFFT burden to be distributed equally over 6 homogeneous computers the parallel computation time would be 72/6=12 s. In this case, the FDTD node dominates the total simulation time. The highest computation gain may be reached with this parallel FDTD if the computation burden of all 6 NTFFT nodes is the same and equal to the main FDTD node. In this case, parallellization gain during the numerical simulations would be 7. If for the standard FDTD simulations, the FDTD computation time is greater than the NTFFT computation time over 6, the gain reduces to 6 and total computation time is determined by the FDTD computations. If this is the case, it is best to run the main FDTD node on the highest performance PC.

The final test belongs to Scenario-3 run on 7 homogeneous PCs. Table 3 lists the computation times of both standard-FDTD and parallel-FDTD runs. The gains for S3-D5 and S3-D1 are (103.640/55.603=**1.86**) and (304.210/65.952=**4.61**), respectively. It should be noted that the scenario S1-D1 can not be run on each of the same terminals with 128 MB RAM memories with standard-FDTD because of the hardware limitations.

Table 3: Standard vs. Parallel FDTD on homogeneous PCs for Scenario-3

Scenario-3		*Standard FDTD [s]*	*Parallel FDTD [s]*
D-5	*total*	**103.640**	**55.603**
	FDTD	50.970	
	NTFF	49.620	
D-1	*total*	**304.210**	**65.952**
	FDTD	50.980	
	NTFF	238.940	

V. CONCLUSION

A novel parallel-FDTD approach for RCS and antenna simulations is discussed. The performance of the parallel-FDTD run on heterogeneous computers may be optimized using non-cubical 3D-FDTD volume. In this case, the number of cells on the facets of the virtual NTFFT closed surface is different. The largest/smallest facets may be assigned to the fastest/slowest PC. Even for the cubical 3D-FDTD volume one still can specify a non-cubical virtual NTFFT closed surface and run parallel FDTD code. Alternatively, if the available PCs are homogeneous it is best to choose a cubical virtual NTFFT closed surface so that each NTFFT node has exactly the same computation burden.

REFERENCES

[1] K. S. Yee, "Numerical Solution of Initial Boundary Value Problems Involving Maxwell Equations," IEEE Trans. Antennas and Propagat., V-14, No.3, pp.302-307, May 1966.

[2] Vineet Ahuja, Lyle N. Long, "A Parallel Finite-Volume Runge-Kutta Algorithm for EM Scattering," Journal of Computational Physics 137, 299-300 (1997), Article No. CP975802

[3] C. Guiffaut, K. Mahdjoubi, "A Parallel FDTD Algorithm Using the MP Library," IEEE Antennas and Propagation Magazine, Vol. 43, No. 2, April 2001

[4] Wang Huiling, XUE Zhenghui, YANG Shimming, GAO Benqing, "Near-field Scattering Analysis with Parallel FDTD Algorithm," IEEE International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications Proceedings 2005

[5] Wang Chen, Panos Kosmas, Miriam Leeser, Carey Rappaport, "An FPGA Implementation of the Two-Dimensional Finite Difference Time Domain (FDTD) Algorithm," Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field Programmable Gate Arrays, Pages 213-222 Monterey, California, USA 2004

[6] R. J. Luebbers et. al. "A Finite-Difference Time-Domain Near Zone to Far Zone Transformation," IEEE Trans. Antennas and Propagat., Vol.39, No.4, pp.429-433, 1991.

[7] L. Sevgi, Complex Electromagnetic Problems and Numerical Simulation Approaches, IEEE Press – John Wiley & Sons, Piscataway, New Jersey, 2003.

[8] L. Sevgi, S. Paker, "FDTD Based RCS Calculations and Antenna Simulations," AEU, International J. of Electronics and Commun., Vol.52, No.2, pp.65-75, March 1998

[9] L. Sevgi, "Target Reflectivity and RCS Interaction in Integrated Maritime Surveillance Systems Based on Surface Wave HF Radar Radars," IEEE Antennas and Propagation Magazine, pp. 36-51, Feb. 2001

[10] L. Sevgi, "Numerical Simulation Approaches for Phased Array Design", ACES Journal of Applied Computational Electromagnetic Society, Special issue on Phased Array Design, (invited tutorial), Vol. 21, No.3, pp. 206-217, Nov 2006

[11] Ç. Uluışık, , G. Çakır, M. Çakır, L. Sevgi, "Radar Cross section (RCS) Modeling and Simulation: Part I – A Tutorial Review of Definitions, Strategies, and Canonical Examples," IEEE Antennas and Propagation Magazine, (under review) Jun 2007

[12] G. Çakır, M. Çakır, L. Sevgi, "Radar Cross section (RCS) Modeling and Simulation: Part II – A Novel FDTD-Based RCS Prediction Virtual Tool for the Resonance Regime," IEEE Antennas and Propagation Magazine, (under review) Jun 2007

[13] M. Çakır, G. Çakır, L. Sevgi, "A Novel Parallellization Approach for Radar Cross section (RCS) and Antenna Simulations", IEEE Transactions on Antennas and Propagation, (submitted) Jun 2007