Practical Implementation of a CPML Absorbing Boundary for GPU Accelerated FDTD Technique

[1]M. J. Inman, [1]A. Z. Elsherbeni, [2]J. G. Maloney, and [2]B. N. Baker

[1] Department of Electrical Engineering
University of Mississippi, University, MS 38677-1848, USA
atef@olemiss.edu , mjinman@olemiss.edu

[2] Georgia Tech Research Institute
Georgia Institute of Technology, Atlanta, GA 30332, USA
jim.maloney@gtri.gatech.edu , brad.baker@gtri.gatech.edu

Abstract – The use of graphical processing units (GPU) has been recently documented for the implementation of the FDTD technique; however, little has been reported about the necessary additions to three dimensional FDTD codes to make the technique more useful for fast antenna analysis and design. This paper details the addition of a convolutional perfectly matched layer absorbing boundary (CPML) to a three dimensional GPU accelerated FDTD code.

Keywords: FDTD, PML, CPML, and GPU.

I. INTRODUCTION

The use of a graphical processing unit (GPU) to accelerate the nested loops for updating the field of a three dimensional FDTD code has been documented in literature over the past few years [1-3]. What has been absent is the implementation of all the additional features of electromagnetic simulation that allow the FDTD technique to be so useful to the antenna engineer. These features include, but are not limited to, a functioning absorbing boundary, a plane wave injection method, discrete feeds for driven antennas, sub-cell models, linear and non-linear circuit element models, and near to far-field transformation for radar cross-section (RCS) and antenna pattern analysis. The goal of this paper is to add the first item in the list, a PML absorbing boundary, to a GPU-accelerated code without giving up too much of the speed advantage provided by the use of the GPUs. Since the convolutional PML [4] relies on the same triple-nested loops as the standard 3D FDTD, as well as having several other benefits [5], it seemed to be a promising candidate to implement in a GPU FDTD code. This paper addresses the benefits of implementing the CPML in a 3D FDTD code executed on a GPU. The details of how to construct a GPU FDTD code is not within the scope of this paper as this can be found in [1-3].

II. CPML FORMULATION

The CPML formulation was chosen both for its simplicity, as well as the straightforward nature of its implementation [5]. Both the standard PML [6] and its Uniaxial [7] formulation require the PML region to be updated separately from the rest of the computational domain. These formulations also possess a two-step update procedure and a complicated set of coefficients to allow general materials to be present in the PML region. The CFS-PML is favorable due to the fact that all cells in the PML are updated in the normal FDTD loop, so all general materials are handled. After the normal loop of updating the field components, the convolutional term is added to the appropriate fields for each face that has PML present on it. This is also a two step process, but the first step is simply the normal FDTD update process.

For the sake of completeness, this section will detail the formulation of CPML used in the GPU accelerated code described in this paper. The derivations for all equations, as well as a much better descriptions of both the CFS-PML and the CPML, are given in [8]. The first step in building the CPML is to set the field updating coefficients correctly. The coefficients are scaled spatially from the edge of the computational domain. All of the following equations describe a CPML that attenuates waves traveling toward the lower z boundary. The two important terms are the complex frequency shifted term, a, and the PML conductivity term, σ which are given as,

$$a_z(u) = a_{\max}\left(\frac{iPML-(u-1)}{iPML}\right)^m \qquad (1)$$

$$\sigma_z(u) = \sigma_{\max}\left(\frac{u}{iPML}\right)^m . \qquad (2)$$

In equations (1) and (2), *u* is the integer representing the location from the lower z boundary, and *iPML* is the number of PML cells (which is set to 10 for the results presented here). The term *m* is the order of the polynomial taper, which was set to 4. The polynomial tapers are applied to a maximum values for *a* and σ, which are defined as,

$$a_{\max} = 2\pi\varepsilon_o F_o / 10 \tag{3}$$

$$\sigma_{\max} = \frac{0.8(\mathrm{m}+1)}{\Delta_z \sqrt{\mu_o / \varepsilon_o}}. \tag{4}$$

Equation (3) is taken directly from [7], while equation (4) is chosen as a good fit for most problems. The F_o term is the center frequency of the excitation pulse in the frequency domain. For this case, a derivative of a Gaussian is used as the source waveform. The spatially scaled terms are then used to create the b_z and c_z coefficients such that,

$$b_z(u) = e^{-\Delta_t\left(\frac{\sigma_z(u)}{\varepsilon_o \kappa_z(u)} + \frac{a_z(u)}{\varepsilon_o}\right)} \tag{5}$$

$$c_z(u) = \left(\frac{\sigma_z(u)}{\sigma_z(u)\kappa_z(u) + \kappa_z(u)^2 a_z(u)}\right) b_z(u). \tag{6}$$

As mentioned previously, all cells in the computational domain, the PML cells included, are updated with the standard FDTD update equations. For the lower z boundary example, the *Ex* and *Ey* terms are then modified by a convolutional "correction" term to apply the PML. These terms are given in equations (7) and (8). They are then added into the *Ex* and *Ey* terms as given in equations (9) and (10). The *CEXH* term in equations (9) is the usual magnetic coefficient for the Ex update equation. Likewise for the *CEYH* term in equations (10). These terms are defined in equations (11) and (12) as a function of the permittivity and conductivity of the material at individual points and are separated by direction,

$$\Psi E_{xz}(i,j,u) = b_z(u)\Psi E_{xz}(i,j,u) + c_z(u)\left(Hy(i,j,u) - Hy(i,j,u-1)\right) \tag{7}$$

$$\Psi E_{yz}(i,j,u) = b_z(u)\Psi E_{yz}(i,j,u) + c_z(u)\left(Hx(i,j,u) - Hx(i,j,u-1)\right), \tag{8}$$

$$E_x(i,j,u) = E_x(i,j,u) - CEXH(i,j,u)\Psi E_{xz}(i,j,u), \tag{9}$$

$$E_y(i,j,u) = E_y(i,j,u) + CEYH(i,j,u)\Psi E_{yz}(i,j,u), \tag{10}$$

$$CEXH(i,j,u) = \frac{\Delta t}{\varepsilon_{rx}(i,j,u)\varepsilon_0 + \frac{\Delta t \sigma_{ex}(i,j,u)}{\varepsilon_0}}, \tag{11}$$

$$CEYH(i,j,u) = \frac{\Delta t}{\varepsilon_{ry}(i,j,u)\varepsilon_0 + \frac{\Delta t \sigma_{ey}(i,j,u)}{\varepsilon_0}}. \tag{12}$$

The magnetic CPML update process proceeds similarly. The one difference that should be noted is that the spatially scaled coefficients are shifted by the usual ½ cell characteristic of the Yee cell [9]. The update equations for the magnetic field are given by,

$$\Psi H_{xz}(i,j,u) = b_z(u)\Psi H_{xz}(i,j,u) + c_z(u)\left(Ey(i,j,u+1) - Ey(i,j,u)\right) \tag{13}$$

$$\Psi H_{yz}(i,j,u) = b_z(u)\Psi H_{yz}(i,j,u) + c_z(u)\left(Ex(i,j,u+1) - Ex(i,j,u)\right), \tag{14}$$

$$H_x(i,j,u) = H_x(i,j,u) - CHXE\left(\Psi H_{xz}(i,j,u)\right), \tag{15}$$

$$H_y(i,j,u) = H_y(i,j,u) + CHYE\left(\Psi H_{yz}(i,j,u)\right), \tag{16}$$

$$CHXE(i,j,u) = \frac{\Delta t}{\mu_{rx}(i,j,u)\mu_0 + \frac{\Delta t \sigma_{mx}(i,j,u)}{\mu_0}}, \tag{17}$$

$$CHYE(i,j,u) = \frac{\Delta t}{\mu_{ry}(i,j,u)\mu_0 + \frac{\Delta t \sigma_{my}(i,j,u)}{\mu_0}}. \tag{18}$$

III. GPU IMPLEMENTATION OF THE CPML

Efficient implementations of basic FDTD technique on GPU's have been documented in the past, however, including absorbing boundary conditions can present a few challenges. Certain common boundary types such as Mur or Liao may not have easy implementation due to the nature of the time and spatial dependency of their updating equations, especially for higher orders absorbing boundaries. Furthermore, this problem gets to be more complicated for three dimensional problems where the 3D to 2D translation [3] is necessary for storage inside the GPU card as seen in Fig. 1. Because of this translation, the various x, y, and z boundaries inside the domain are scattered amongst the various tiles. Thus applying the boundary conditions on the individual boundaries becomes very complicated.

On the contrary, the CMPL boundary condition can be implemented with a much easier procedure since CPML can be represented by FDTD-like arrays. In most efficient C and FORTRAN implementations of CPML, the coefficients (ψ, b, c) and the processing loops operate only on the boundary locations, however since the coefficients in non-boundary areas would be zero, the coefficients and processing loops can be extended to cover the entire domain. While unnecessary in C and FORTRAN implementations, this becomes necessary in the GPU code as the boundaries are scattered throughout the 2D translated arrays. This allows for a much simpler updating function as it can be applied over the entire domain without having to worry in the GPU section where exactly the boundary locations are. Figure 2 shows the location of the various boundaries once the 3D domain has been decomposed into a 2D tiled domain. The program will calculate and populate the various coefficients necessary to implement the CPML boundary in these regions only.

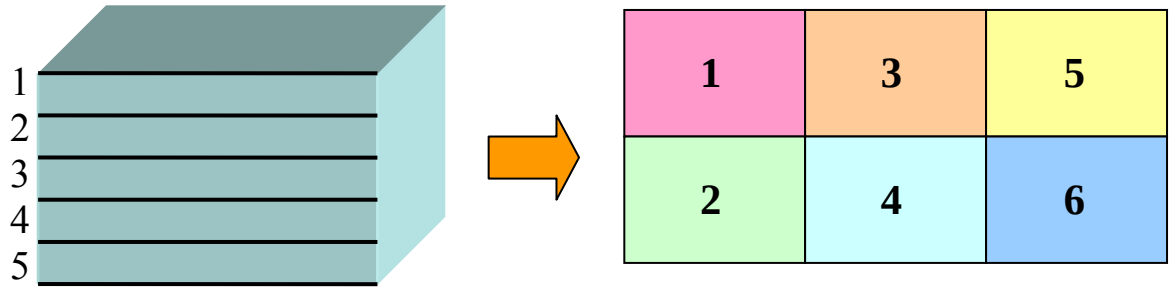


Fig. 1. 3D to 2D translation via tiling.

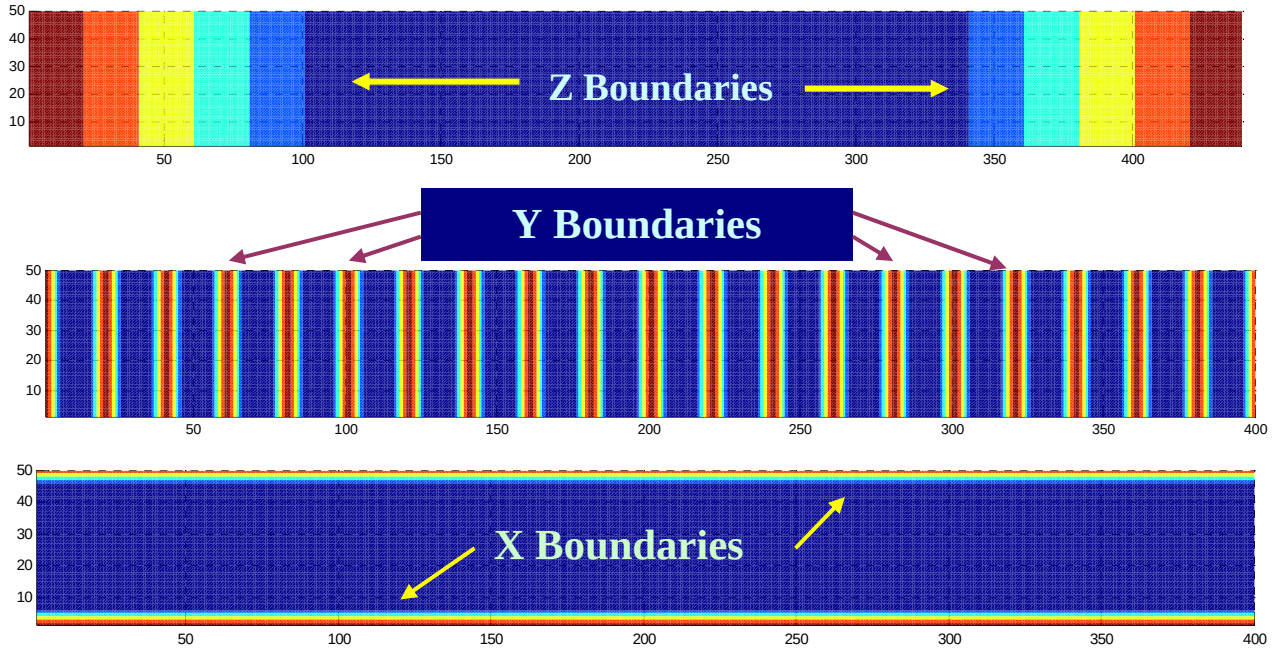


Fig. 2. Locations of the boundaries in the 2D texture.

The updating equations were implemented in GPU kernels as simple functions that would first calculate the necessary ψ terms, then apply the correcting terms to the E and H field components. The entire process is integrated easily into the GPU program with only minor changes in the precalculation of the *b* and *c* terms before the GPU performs the time steps.

IV. RESULTS

The GPU program was tested against a known FORTRAN based FDTD code with CPML to insure accuracy and proper operation. Both the GPU and CPU based codes were run on Intel Dual-Core 2.8 GHz systems with an Nvidia 8800 GTX video card with 768M of RAM. In the progression of time steps a derivative of a Gaussian waveform is injected from a point source at the center of the domain and progresses outward before being absorbed by the CPML layers and finally only very small reflections of the CPML remains. The source waveform is defined as,

$$s(t) = \frac{2}{\tau^2}\left(t_0 - t\right) e^{\frac{(t-t_0)^2}{\tau^2}} \tag{19}$$

where

$$\tau = \frac{2.1c}{40\pi \min(\Delta x, \Delta y, \Delta z)}, \tag{20}$$

$$t_0 = 4.5\tau . \tag{21}$$

Figure 3 shows the Ez field component at a plane cut containing the source point for various time steps to show proper operation of the GPU based program.

Figure 4 shows the Ez field component at an observation point 10 cells from the source point over 500 time steps. The wave is injected from approximately 40 to 150 time steps while the reflection of the CPML boundary can be seen at approximately 225 time steps into the simulation. The maximum magnitude of this reflection was calculated to be less than 0.3%. This reflection is higher than standard FORTRAN codes due to the numerical precision of the GPU.

With the PML having been verified, several test cases were run to present the full functionality and verify real simulation results. The two cases presented here are the well known microstrip patch antenna and filter [10]. Figure 5 shows the layout for the simple microstrip patch antenna.

The patch was simulated on both the GPU and CPU based systems for 3000 time steps. Figure 6 shows both the time domain and frequency domain results. The results show good comparison overall to the reference data [10] with minor difference due to the actual implementation of the codes and the numerical precision of the GPU.

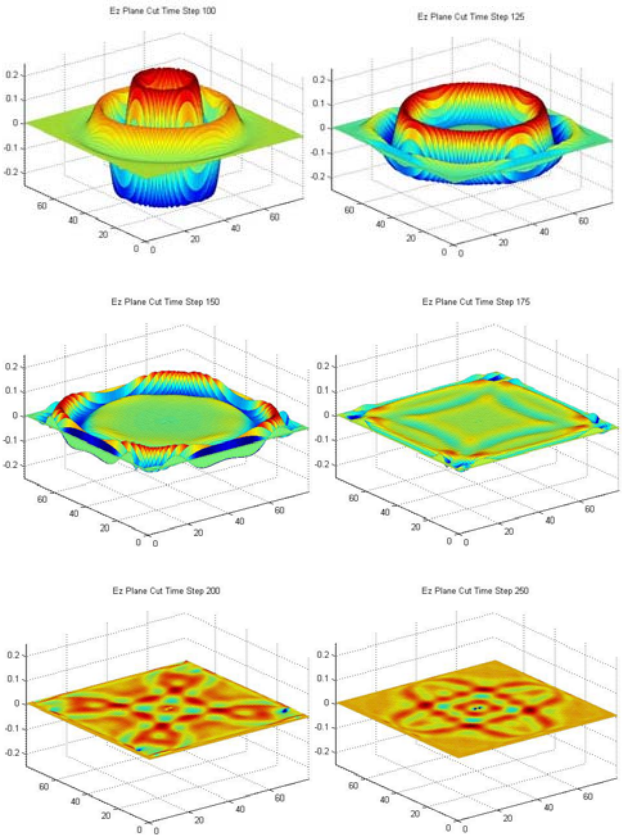


Fig. 3. Ez plane cuts at various time steps.

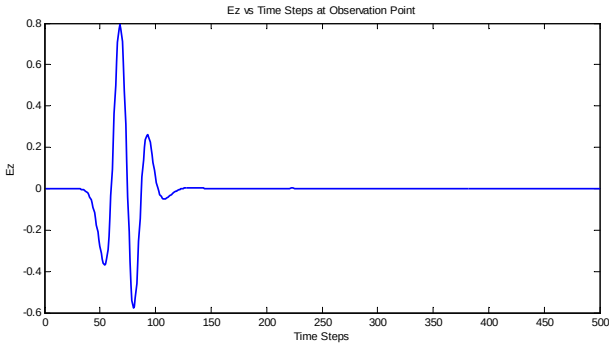


Fig. 4. The Ez field component at the observation point over 500 time steps.

The second test case simulates a microstrip filter. The simulation was also run on the GPU systems for 3000 time steps. Figure 7 shows the layout of the simple microstrip filter while Fig. 8 shows the results from this filter. Again good agreement is shown between the GPU results and the reference data [10].

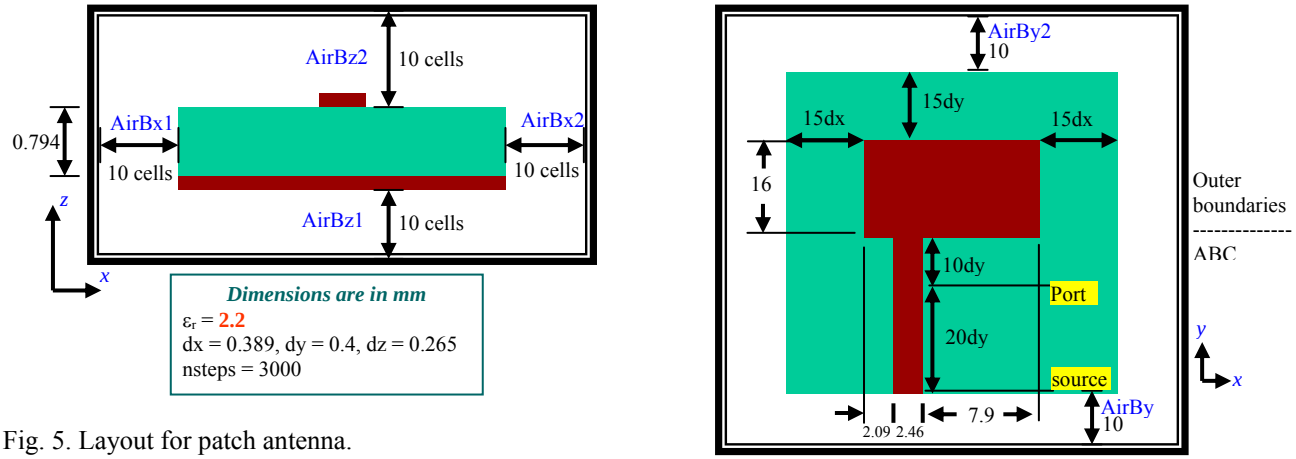


Fig. 5. Layout for patch antenna.

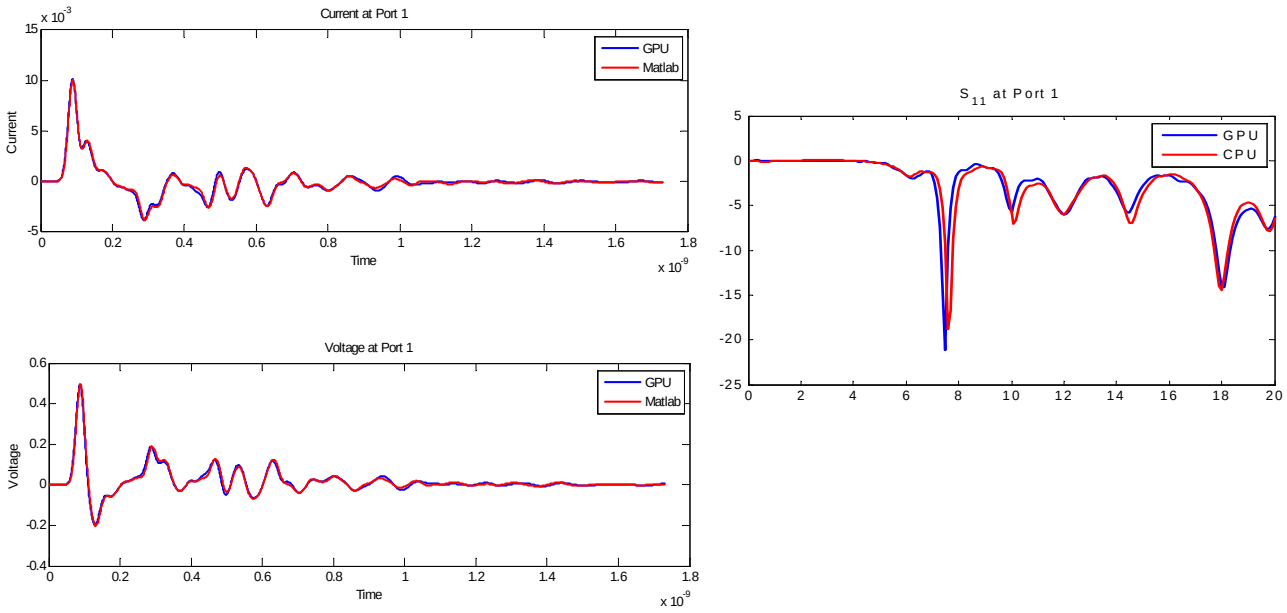


Fig. 6. Patch antenna verification results.

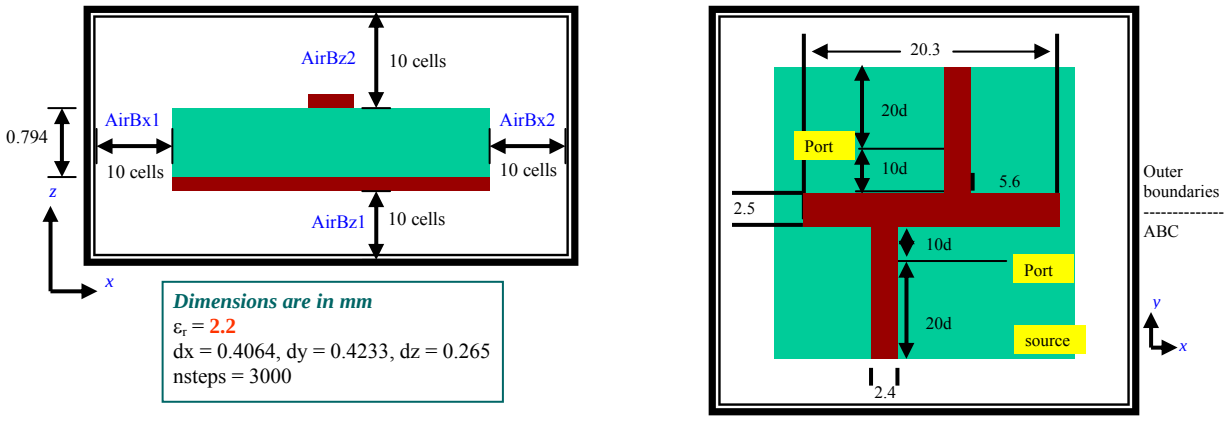


Fig. 7. Microstrip filter layout.

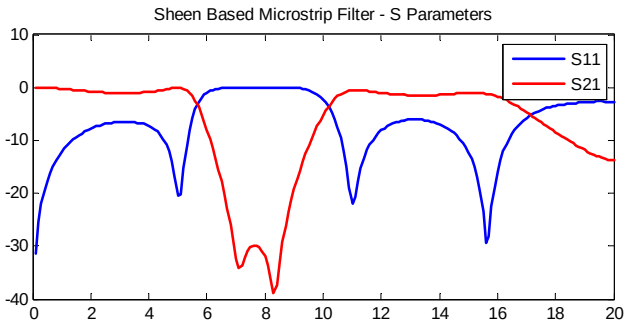
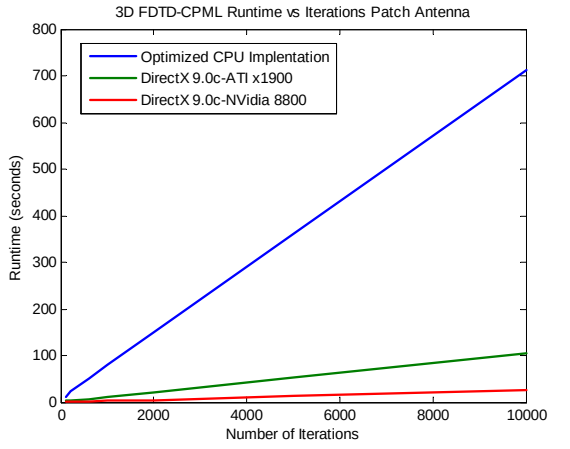
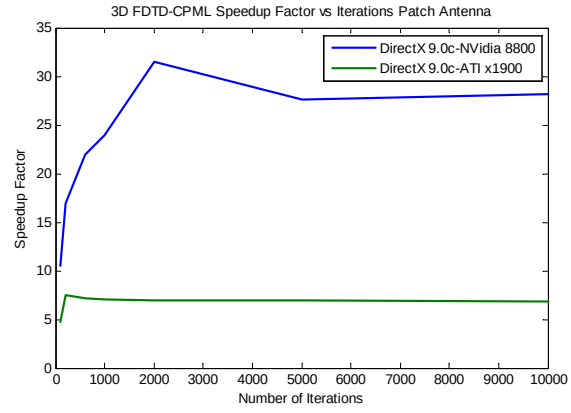


Fig. 8. Microstrip filter results.



(a)



(b)

Fig. 9. Fortran CMPL vs. GPU CPML implementation results for various time steps with 500000 cell configuration, (a) run time; (b) speed up factor.

Figure 9 shows runtime and speedup results for both an optimized CPU and GPU code for the patch antenna test configuration of approximately 500000 cells. The GPU code was run on both an ATI x1900 and NVidia 8800 GPUs, while the CPU code was run on a Dual Core Pentium 2.8 GHz processor. The runtime results show a near linear trend for this case as the number of time steps is increased. The speedup factors for this case show that as the number of time steps is increased the maximum speedup factor asymptotically approaches a limit of 26 for the GPU on the NVidia 8800 and 6.4 for the ATI x1900.

V. CONCLUSIONS

The GPU based code outlined in this paper has shown good performance compared to a known FORTRAN code in implementing a three dimensional FDTD simulation with a CPML boundary condition. While the speedup factors gained in this GPU code is less than that has been shown without a boundary condition, it still offers a significant gain in speed over purely CPU based FDTD solvers. As the domain size is increased, the speedup factor slightly decreases due to the fact that the CPML coefficients and updates has to be implemented over the entire domain rather than in sections as it is in the FORTRAN code. It is expected that the current implementation would yield higher speed factor when new generation of GPUs are used.

REFERENCES

[1] M. J. Inman, A. Z. Elsherbeni, and C. E. Smith "GPU programming for FDTD calculations," *The Applied Computational Electromagnetics Society (ACES) Conference*, Honolulu, Hawaii, 2005.

[2] M. J. Inman and A. Z. Elsherbeni, "3D FDTD acceleration using graphical processing units," *The Applied Computational Electromagnetics Society (ACES) Conference*, Miami, Florida, 2006.

[3] M. J. Inman and A. Z. Elsherbeni, "Programming video cards for computational electromagnetics applications," *IEEE Antennas Propagation Mag*., vol. 47, no. 6, pp. 71-78, 2005.

[4] J. A. Roden, and S. D. Gedney, "Convolutional PML (CPML): An efficient FDTD implementation of the CFS-PML for arbitrary media," *Microwave Optical Tech. Let*., vol. 27, pp. 334-339, 2000.

[5] S. D. Gedney, "Scaled CFS-PML: It is more accurate, more efficient, and simple to implement. why aren't you using it?," *IEEE Antennas and Propagation Society International Symposium*, vol. 4B, pp. 364-367, July 2005.

[6] J. P. Berenger, "A perfectly matched layer for the absorption of electromagnetic waves," *Journal of Computational Physics*, vol. 114, pp. 195-200, 1994.

[7] S. D. Gedney, "The perfectly matched layer absorbing medium," *Advances in Computational Electrodynamics: The Finite Difference Time Domain*, A. Taflove, Editor, Artech House, New York, pp. 263-340, 1998.

[8] S. D. Gedney, "Perfectly Matched Layer Absorbing Boundary Conditions," in *Computational Electrodynamics: The Finite Difference Time Domain*, third edition, A. Taflove, and Susan C. Hagness, Editors, Artech House, Norwood, MA, pp. 295-313, 2005.

[9] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Trans. Antenna Propagation*, vol. AP-14, pp. 302-307, May 1966.

[10] D. M. Sheen, S. M. Ali, M. D. Abouzahra, and J. A. Kong, "Application of three-dimensional finite-difference time-domain method to the analysis of planar microstrip circuits," *IEEE Trans. Microwave Theory and Techniques*, vol. 37, no. 7, pp. 849-857, July 1990.

Matthew Joseph Inman received his B.S. in Electrical Engineering in 2000 and his Masters in Electromagnetics in 2003 from the University of Mississippi. He currently is currently pursuing Ph. D. studies in electromagnetics there. He is currently employed at the University as a research assistant and graduate instructor. His interests involve electromagnetic theories, numerical techniques, antenna design and visualization, as well as teaching a number of undergraduate courses.

Atef Z. Elsherbeni – Biography can be found on page 61