Interactive GPU Based FDTD Simulations for Teaching Applications

Matthew J. Inman and Atef Z. Elsherbeni

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

Abstract: Graphical processing units (GPU) has been recently documented for the implementation of the FDTD technique. The use of these specialized processors for the implementation of numerical codes has been shown to significantly speed up the execution of these codes over standard CPU based solvers. With the execution of the FDTD method being reduced to a matter of seconds, the use of these codes in teaching situations becomes possible. This paper will detail the use of a developed GPU based FDTD solvers and associated interfaces for teaching applications.

Keywords: FDTD, GPU, GUI

1. Introduction

The use of a graphical processing unit (GPU) to accelerate the nested loops for updating the field of a three dimensional finite difference time domain (FDTD) code has been documented in literature over the past few years [1-4]. Improvements in the codes applied on GPU has allowed this technique to be implemented with many useful additions such as PML absorbing boundaries and expanded domain sizes while retaining the speed gains offered over standard CPU based simulators. With the basic framework having been detailed in previous papers, the use of these GPU based solvers can be applied to a wide variety of problems such as antennas, filters, and other electromagnetic devices. The speed gains provided by the GPU based FDTD codes open the technique to widespread use in optimization and parameter exploration. Traditional execution time for a simulation using the FDTD method would take anywhere from a few minutes to many hours or more. As has been shown in previous papers by the authors, this execution time for two and three dimensional FDTD simulations can be decreased by up to 25-30 times compared to their CPU based counterparts. This paper will show that for many common simulations, the speed gained from using a GPU based code will bring the total execution time down to a reasonable amount such that in conjunction with a simple Matlab based graphical user interface (GUI), interactive simulations can be used in teaching activities. These graphical user interfaces will allow the background GPU code to be connected with an easy to use interface that allows the user to set parameters on a simple simulation and perform the simulation. With the GUI interface, it is possible to vary any of the problem parameters using Matlab generated sliding bars and to show how these parameters affect the output results interactively.

2. GPU FDTD Implementation

In order to allow the GPU to perform the calculations necessary for FDTD, an interface or library must be used that allows for the programming of the GPU. In these libraries, specific calls will be made to the shading routines provided. These “shaders” as they are called, allow for customized calculations to be performed upon the textures (which in our case represents various matrices). In the end, instead of the results being displayed to the screen, the final calculations will be returned to the user. By programming the shaders to perform the FDTD calculations, the vector processing capabilities of the GPU will provide a large increase in speed.

For this programming a language named "Brook" [5] is be used. Brook is based upon the C standard that allows for the creation of generic subroutines named Kernels to be implemented upon the GPU. These Kernels are simply special functions performed on matrices that will be implemented on the GPU. When compiled, Brook generates the necessary shader programming to interface the general C program with the GPU. This program can then be run on any computer with a compatible graphics card manufactured after 2003.

In the case of FDTD programming, the GPU will be programmed to do the updating equations using custom Kernels. The developed program first calculates all the coefficients necessary for the FDTD update equations, transfers these coefficient arrays to the graphics card, and then through the use of programmed shaders, executes the updating equations. Ancillary functions such as absorbing boundary layers, etc are also implemented for processing on the graphics card to achieve the high possible gains in speed. Brook allows the programmer to transfer the data to the graphics card and back quite easily through a single function call so that data may be examined with the independently developed Matlab GUI.

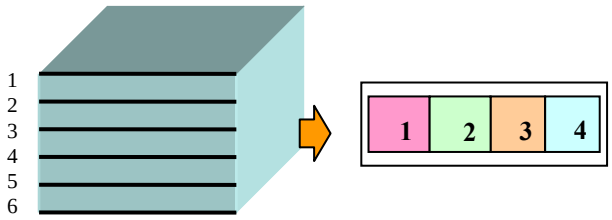

Figure 1. 3D to 2D Translation via tiling.

Unfortunately all the processing in the GPU is done on two-dimensional matrices. This presents a problem in how to program it to perform three dimensional calculations. The simplest method available is tiling, in which every layer in the 3rd dimension is represented as a tile in the 2D array as seen in Figure 1. The drawback to this is the tiling must fit within the maximum texture size for the GPU which is often 2048x2048 for older cards and 4096x4096 for newer ones. Additionally every added dimension of the tiling increases the complexity of the updating Kernels as any time data is required from another tile it must be known how to access it. In addition accessing data from outside the local area currently being performed can slow down the computations as the time to fetch data increase with the randomness contrary to sequential data (such as from neighboring cells) which can be accessed at a much faster rate. However the internal transfer rate of data inside the GPU is orders of magnitude higher than in the rest of the computer and this delay cause's minimal reduction in calculation time.

3. Sample Configuration

To demonstrate the utility of GPU based solvers for optimizing three dimensional electromagnetics problems, a sample configuration of a patch antenna was chosen. The example configuration is the standard patch antenna defined in [6]. This is a simple offset fed patch antenna on a known dielectric substrate of $\varepsilon_r = 2.2$ and after a 10 cell air gap there is a CMPL boundary [7,8] as seen in Figure 2. Since the characteristics are well know for this configuration it will serve as both an example and as a verification of proper operation of the GPU based FDTD codes.

For this example configuration there are four parts which have been designated as parameters to be explored as well as the permittivity and thickness of the substrate. The parts of the patch that can be varied are shown in Figure 2. and they are denoted as A, B, C, and D and for the left width, feed width, right width, and total height of the patch, respectively. To achieve this, a GPU based code was written in Brook to allow command line parameters for A, B, C, D, substrate thickness, and permittivity to be

imported into the program. The program will then construct the proper geometry based upon these dimensions to simulate the appropriately size patch antenna. The program will then export the resulting data from the port into a text file which will be read from the controlling program and analyzed appropriately.

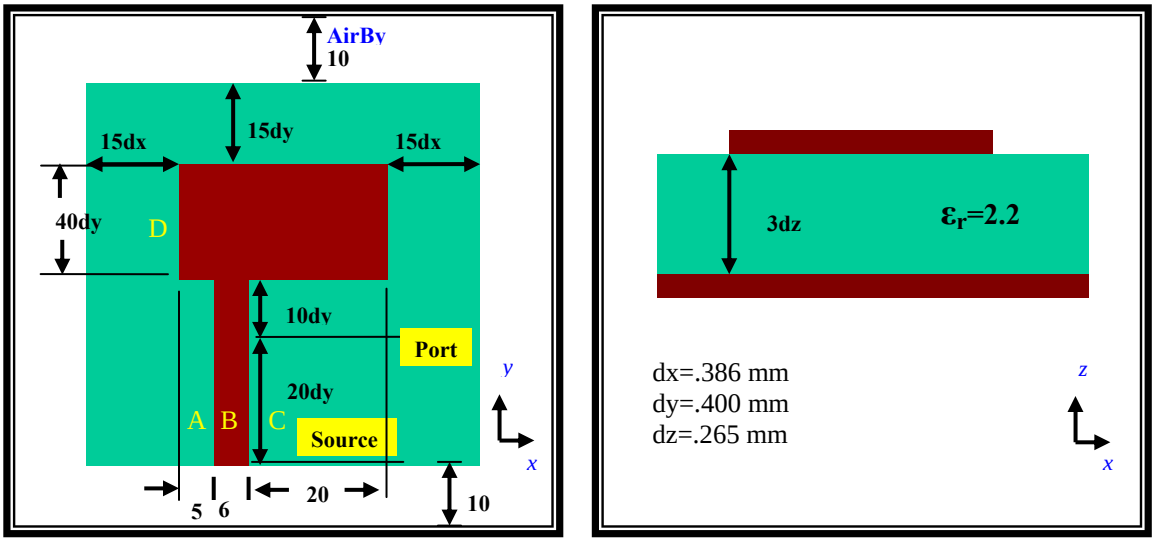

Figure 2. Sample patch antenna layout.

4. Graphical Interface for Microstrip Patch Antenna

When teaching the proper design of the microstrip patch antenna or other electromagnetic devices, understanding how the different parameters of the device effect its operation can be of the utmost importance. For example in the microstrip patch antenna, changing the feed line width and patch size can dramatically change the operation of the antenna. Being able to interactively vary these parameters and having the results shown in a matter of seconds is very useful in teaching how this antenna operates. To this end a simple GUI front end to a GPU based FDTD solver can be implemented to facilitate this point. Figure 3 shows a simple GUI that was created to run the GPU solver for the patch example.

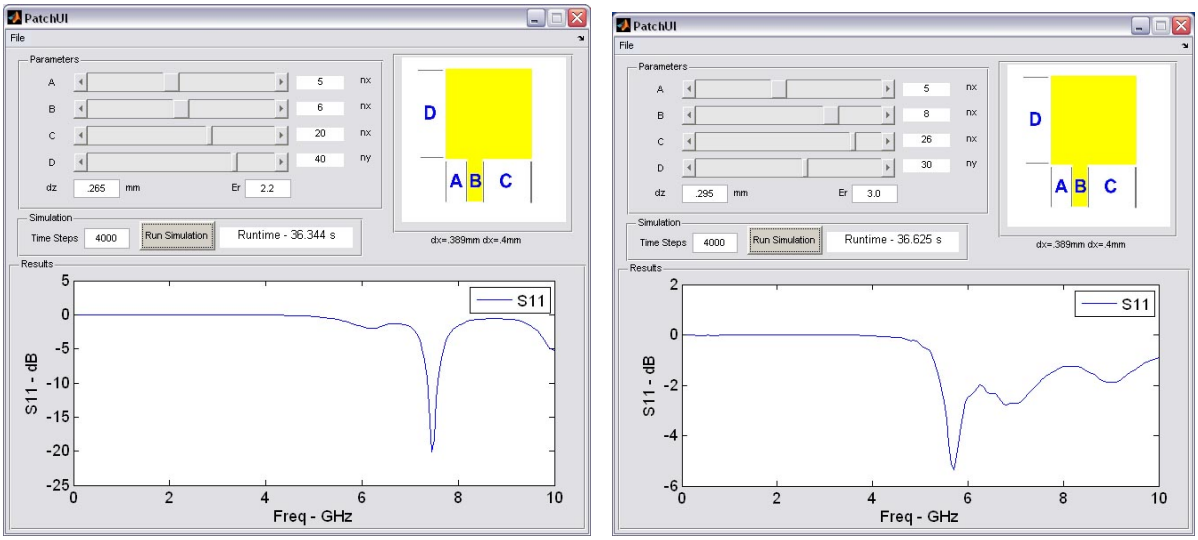

Figure 3. Example GUI for patch antenna.

For example in the case of the patch antenna it is known that parameter B which is the feed line width will have a major effect on its operation. If this feed line is widened, the size of the patch is changed, and the substrate parameters have been modified, the operational frequency of the antenna will change accordingly. This will be reflected in the shown graphical results as both a shift in the resonant frequency and a change in the return loss bandwidth for the system as seen in Figure 3.

Likewise the other parameters can interactively be changed to examine their effects on the operation of the patch antenna. Since the run time of the simulations have been reduced to seconds in the case of the current generation graphics card, the utility of these programs has been greatly increased. With such shortened execution times it can be used extensively in teaching to show interactively how changes in the various parameters will effect the operation of any device. All that is required is to set up a GPU based solver for the intended simulation.

5. Conclusion

In order to properly teach and understand a wide variety of electromagnetic devices, it is often necessary to perform a number of simulations to achieve the final design goals. With the advent of GPU based solvers and the speed gains available with them, the use of the FDTD method as a tool for teaching and parameter exploration becomes increasingly more useful. Whereas in the past the limiting factor in FDTD simulations has often been the speed of its execution, this is no longer the case. GPU based FDTD solvers have shown that they are capable of performing the same simulations in a fraction of the time as their CPU brethren. For parameter explorations, the speed of the GPU based codes is ideal for interactively learning the effects of each parameter in device design. Such techniques are extremely useful in reducing the complexity of large optimization problems as well as in teaching the design and operation of devices.

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, Issue 6, pp. 71-78, 2005.

[4] M. J. Inman, A. Z. Elsherbeni, B. N. Baker, and J. Maloney, "Practical Implementation of a CPML Absorbing Boundary for GPU Accelerated FDTD Technique," IEEE APS Symposium, Honolulu, Hawaii, 2006.

[5] Ian Buck, "Brook Spec v0.2", Stanford University. 2003.

[6] D. M. Sheen, S. M. Ali, M. D. Abouzahra, and J. Au Kong, "Application of the Three-Dimensional Finite-Difference Time-Domain Method to the Analysis of Planar Microstrip Circuits", IEEE Trans.on Microwave Theory and Tech. Vol. 38, No. 7, July 1990.

[7] J. A. Roden, 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.

[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.