International Journal of Computer Applications (0975 – 8887)
Volume 70 - No.27, May 2013

Accelerated Electromagnetic Field Simulation on Graphical Processing Unit by the Finite Difference Time Domain Method

Mayank Gupta.
Dept. of Electronics and Telecommunication.
Yeshwantrao Chavan College Of Engineering.
Nagpur, India.

Anagha Choudhhari.
Dept. of Electronics and Telecommunication.
Yeshwantrao Chavan College Of Engineering.
Nagpur, India.

N. A. Pande.
Dept. of Electronics and Telecommunication.
Yeshwantrao Chavan College Of Engineering.
Nagpur, India.

ABSTRACT

This paper details about the implementation of the Finite Difference Time Domain Method (FDTD) on a Graphical Processing Unit (GPU) and demonstrated the ability of low cost GPU's to accelerate real life problems of interest in the Electromagnetic Field Simulation domain. Source code and the achieved results for a simple simulation problem implemented in C is presented. The same problem is then discussed by implementation using Nvidia's software development platform CUDA. The methodology for implementation of some advanced FDTD simulation problems is also described, and the achieved results are presented using both, CUDA and MatLab R2011a. Finally, a comparative performance analysis of the FDTD implementation using CUDA for a 256x256x256 volume is described for 5000 time steps, by comparing the CPU implementation time with the GPU implementation time. This is a reasonably good example for judging the applicability of low cost GPU's not just for Electromagnetic Simulation, but also for real time applications.

General Terms

General Purpose Graphical Processing Unit (GPGPU) Computing, Finite Difference Time Domain (FDTD) Method.

Keywords

CUDA, Computational Electromagneticg (CEM).

1. INTRODUCTION

The demands of scientific computing are growing day by day. The domain of electromagnetic simulation is no exception to this. Issues related to suitable methodologies for electromagnetic field simulation are to be addressed not just in the industry, but also at the academic institutions. However, the commercially available electromagnetic simulation packages may not always suit these academic and industrial requirements, primarily because of two reasons. Firstly these software packages are very expensive, and secondly they tend to mask the user from the fundamental mathematics involved in simulation processes. Especially in the academic institutions, direct exposure to simulation on a commercial software packages may fail to impart this fundamental understanding, and often leads to intentional or unintentional negligence towards the basics of numerical computation. Also, at times, it may be difficult to have a customized control of simulation parameters in commercial software packages, if such need arises in an exceptional case. This brings us to the need of platforms where electromagnetic field simulation research can be implemented at its most fundamental level. A good approach to answer these issues is to start coding from scratch. Although long and tedious, but this approach is best suited to begin with academic research and in industrial projects where cost is constrained.

One of the most simple, intuitive to understand and useful means to find numerical solutions for electromagnetic simulation problems is the FDTD method. Also, the FDTD method is well suited if one is able to harvest the multi-core resources in a computing system. Thus the low cost availability of modern GPU's can be used to the advantage of scientific computing.

The FDTD method is based on the central difference approximation to a partial differential equation. Maxwell's Equations are the governing equations for all electromagnetic phenomena. The FDTD method solves for these Maxwell's equations in differential form. Being a time domain method, this is well suited for wide-band analysis, that is, analysis over a wide range of frequencies. Detailed theoretical background is well presented in [1].

A simple electromagnetics simulation is presented to describe the FDTD method from a programmer's point of view. The C code is slight modification of the bare bones simulation in [2].

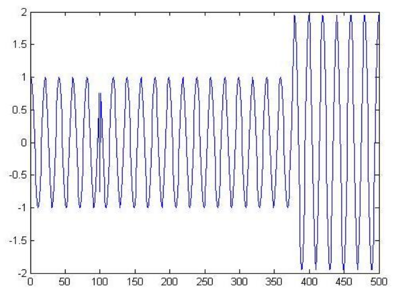


Figure 1. Output of the simple FDTD implementation.

```
int main(int argc, char **argv)

{

shrSetLogFileName(shrLogFile);

shrLog("clc; clear all; close all \n ;
");

float ez[800] = {0.}, hy[800] = {0.};
int imp0 = 377.0, temp;
for (int t = 0; t < 525; t++)
 {
 shrLog(" a = [ ", " ");
 for (int mm = 0; mm < 500 - 1; mm++)
 hy[mm]=hy[mm]+(ez[mm+1]-ez[mm])/ imp0;
 for (int mm = 1; mm < 500; mm++)
{
ez[mm]=ez[mm]+(hy[mm]-hy[mm-1])* imp0;
shrLog("%g\t",ez[mm]); }
ez[100]=sinf(((2*3.14)/20)*(t));
ez[500] = 0;
hy[0] = 0;
shrLog(" ] ; plot(a);");
shrLog(" axis([0 500 -2 2]);");
shrLog(" getframe();\n");
}
}
```

In the output Figure 1, we see that the source is located at point 100, which is a hard wired sinusoidal source. The value of Ez at point 750 is defined as 0, which is imposing the boundary conditions that of a Perfect Electric Conductor (PEC). As expected, we see the wave experience reflection from this PEC boundary and superimpose additively on the incident wave. Similarly, the Hy at point 0 is defined as 0, which amounts to imposing Perfect Magnetic Conductor (PMC) boundary conditions. As expected, there is no reflection from this point. Running the generated log file in MatLab gives the output as shown in Figure 1.

Now, a parallelized version of this same program is presented, which is implemented using Nvidia CUDA 4.2. CUDA 4.2, on installation, automatically integrates with an existing Visual Studio 2010 Express Edition installation. Using the same concept as in the serialized code, we make use of , the function shrLog(). This function is used for writing a log file, and it is defined in shrUtils, which is included in CUDA 4.2 (This is however not available in the latest CUDA 5 release.) The second function is the kernel invocation __global__ void VectAdd(). This program does nothing but adds/substracts two vectors, in our case these vectors are the electric field vector Ez and the magnetic field vector Hy. The kernel invocation is done as vecAdd<<<blockPerGrid,threadPerBlock>>>()

in the main function. The kernel code for updating the H fields is given below.

```
__global__ void VecAdd
(const float* A, float* Ez, float* Hy,
int imp0, int N)
{
int i = blockDim.x * blockIdx.x +
threadIdx.x;
if (i < N)
A[i] = Hy[i] + (Ez[i-1]-Ez[i])/imp0;
// *A is the output of updated *Hy
// Similarly execute Kernel for E.
```

2. SOME ADVANCED FDTD SIMULATIONS USING MATLAB AND CUDA.

As it can be seen from the example discussed in section I, a typical FDTD simulation involves the following steps :

- Defining the computational domain, that is initializing the electric and magnetic field vectors, typically as an array initial with zero value. Also, the ε and μ vectors are defined. The size of the domain is equal to the number of nodes, or the number of elements in electric and magnetic field vectors.
- Enclosing the computational domain in suitable boundary conditions, like the PEC, PMC, Absorbing Boundary Condition (ABC) etc.
- Defining the total number of time steps for which the simulation is to be executed.
- Calculating the value of each, the electric field components and magnetic field components, for each node at each time interval.

Besides these steps, certain precautions need to be taken, like fulfilling the Courant Stability Criteria by choosing suitable step size if space/time. Figure 2 and Figure 3 shows the output from the MatLab simulation of a horn antenna and dipole antenna respectively, for which the above steps were followed. Figure 4 shows the output from the CUDA implementation of 3 dimensional FDTD implementation of a microstrip patch antenna. The result is however a sequence of numbers which needs to be converted in graphical form by some plotting function.

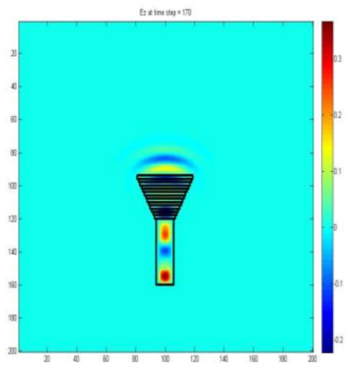


Figure 2.Matlab FDTD Simulation of Horn Antenna.

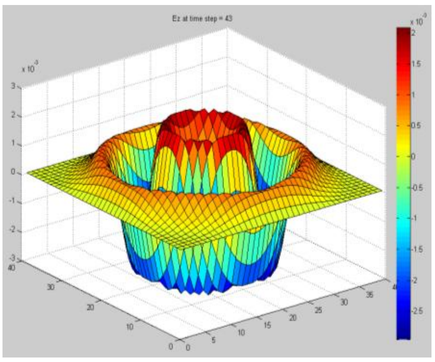
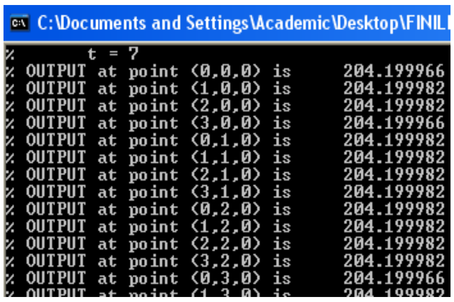


Figure 3.Matlab Simulation of the Dipole Antenna.



```
C:\Documents and Settings\Academic\Desktop\FINIL
%        t = 7
% OUTPUT at point (0,0,0) is      204.199966
% OUTPUT at point (1,0,0) is      204.199982
% OUTPUT at point (2,0,0) is      204.199982
% OUTPUT at point (3,0,0) is      204.199966
% OUTPUT at point (0,1,0) is      204.199982
% OUTPUT at point (1,1,0) is      204.199982
% OUTPUT at point (2,1,0) is      204.199982
% OUTPUT at point (3,1,0) is      204.199982
% OUTPUT at point (0,2,0) is      204.199982
% OUTPUT at point (1,2,0) is      204.199982
% OUTPUT at point (2,2,0) is      204.199982
% OUTPUT at point (3,2,0) is      204.199982
% OUTPUT at point (0,3,0) is      204.199966
% OUTPUT at point (1,3,0) is      204.199982
```

Figure 4 : CUDA output for Microstrip Patch FDTD.

3. TYPESET TEXT

The Table 1 shows the achieved speedups obtained by comparing the simulation on Intel Pentium E5300 Dual Core 2.6 GHz CPU against Nvidia GT520, 48 CUDA Cores, 1.4 GHz clock GPU. As it can be clearly seen, the speedup achieved is around an order of magnitude higher in case of GPU implementation for a 256x256x256 problem size in single precision. Computation time for 5000 time steps is presented. The results are measured using Nvidia Visual Profiler Tool in CUDS 4.2.

Problem Size	**CPU Runtime (sec)**	**GPU Runtime (sec)**	**Speedup**
128x128x128	990	231	4.29
256x256x256	4095	459	8.92

.

4. CONCLUSION

The low cost GPU certainly do possesses tremendous processing power, which can be harnessed with suitable parallelized modifications in the conventional algorithms, such that they become suitable for GPU implementation. Although not all algorithms would possesses this inherent parallelization ability, but those which do possesses this scope of parallelization can be further exploited with the use of GPU's.

5. ACKNOWLEDGMENTS

The authors acknowledge the continued support from Dept. of Electronics and Telecommunication, YCCE, Nagpur for providing the research facilities. The authors also acknowledge the continued motivation of their Head Of the Dept, Dr. P. L. Zade for encouraging research in the field of antenna design, electromagnetic simulation and microwave engineering.

6. REFERENCES

[1] Atef Z. Elsherbeni, Chapter 7, Handbook of Antennas In Wireless Communication, CRC Press, 2002

[2] John. B Schneider, Understanding the Finite-Difference Time-Domain Method, www. eecs. wsu. edu/~schneidj/ufdtd, 2010.

[3] Matthew Livesey et. al., "Development of a CUDA Implementation of the 3D FDTD Method" IEEE Antenna and Propogation Magazine, Vol 54, No 5, October 2012.

[4] Danilo De Donno et. al., "Introduction to GPU Computing and CUDA Programing : A Case Study on FDTD" IEEE Antenna and Propogation Magazine, Vol 52, No 3, June 2010.

[5] C. A. Balanis. Advanced Engineering Electromagnetics. John Wiley & Sons, New York, 1989.

[6] A V Choudhari, N A Pande and M R Gupta. Article: Feasibility Analysis of Low Cost Graphical Processing Units for Electromagnetic Field Simulations by Finite Difference Time Domain Method. *International Journal of Computer Applications* 67(24):30-33, April 2013.

[7] A. Taflove, Computational Electrodynamics: The Finite-Difference Time-Domain Method. , 1995 :Artech House.