Programming Finite-Difference Time-Domain for Graphics Processor Units Using Compute Unified Device Architecture

Veysel Demir* (1) and Atef Z. Elsherbeni (2)
(1) Department of Electrical Engineering, Northern Illinois University, USA
(2) Department of Electrical Engineering, University of Mississippi, USA
E-mail: demir@ceet.niu.edu, atef@olemiss.edu

Abstract

Recently graphic processing units (GPU's) have become the hardware platforms to perform high performance scientific computing them. The unavailability of high level languages to program graphics cards had prevented the widespread use of GPUs. Relatively recently Compute Unified Device Architecture (CUDA) development environment has been introduced by NVIDIA and made GPU programming much easier. This contribution presents an implementation of finite-difference time-domain (FDTD) method using CUDA. A thread-to-cell mapping algorithm is presented and performance of this algorithm is provided.

Introduction

Recent developments in the design of graphic processing units (GPU's) have been occurring at a much greater pace than with central processor units (CPU's). The computation power due to the parallelism provided by the graphics cards got the attention of communities dealing with high performance scientific computing. The computational electromagnetics community as well has started to utilize the computational power of graphics cards for computing and, in particular, several implementations of finite-difference time-domain (FDTD) method have been reported. Initially high level programming languages were not conveniently available to program graphics cards. For instance, some implementations of FDTD were based on OpenGL. Then Brook [1] has been introduced as a high level language for general programming environments, and for instance used in [2] as the programming language for FDTD. Moreover, use of High Level Shader Language (HLSL) as well is reported for coding FDTD. Relatively recently, introduction of the Compute Unified Device Architecture (CUDA) [3] development environment from NVIDIA made GPU computing much easier.

CUDA has been reported as the programming environment for implementation of FDTD in [4]-[7]. In [4] the use of CUDA for two-dimensional FDTD is presented, and its use for three-dimensional FDTD implementations is proposed. The importance of coalesced memory access and efficient use of shared memory is addressed without sufficient details. Another two-dimensional FDTD implementation using CUDA has been reported in [5] however no implementation details are provided. Some methods to improve the efficiency of FDTD using

978-1-4244-4968-2/10/$25.00 ©2010 IEEE

CUDA are presented in [6], which can be used as guidelines while programming FDTD using CUDA. The discussions are based on FDTD updating equations in its simplest form: updating equations consider only dielectric objects in the computation domain, the cell sizes are equal in *x*, *y*, and *z* directions, thus the updating equations include a single updating coefficient. The efficient use of shared memory is discussed, however the presented methods limits the number of threads per thread block to a fixed size. The coalesced memory access, which is a necessary condition for efficiency on CUDA, is inherently satisfied with the given examples; however its importance has never been mentioned.

In this current contribution a CUDA implementation of FDTD is provided. The FDTD updating equations assume more general material media and different cell sizes. A thread-to-cell mapping algorithm is presented and its performance is provided.

FDTD Using CUDA

The unified FDTD formulation [8] considered for CUDA. The problem space size is $Nx \times Ny \times Nz$, where Nx, Ny, and Nz are number of cells in *x*, *y*, and *z* directions, respectively. Thus, for instance, the updating equation that updates *x* component of magnetic field is given in [8] as

$$\begin{aligned} H_x^{n+\frac{1}{2}}(i,j,k) &= C_{hxh}(i,j,k)\, H_x^{n-\frac{1}{2}}(i,j,k) \\ &+ C_{hxey}(i,j,k)\left(E_y^n(i,j,k+1) - E_y^n(i,j,k)\right). \\ &+ C_{hxez}(i,j,k)\left(E_z^n(i,j+1,k) - E_z^n(i,j,k)\right) \end{aligned} \tag{1}$$

In order to achieve parallelism, the threads are mapped to cells to update them. For the mapping, a thread block is constructed as a one-dimensional array, as shown on the first two lines in Listing 1, and the threads in this array are mapped to cells in an *x*-*y* plane cut of the FDTD domain as illustrated in Fig. 1. In the kernel function, each thread is mapped to a cell; *thread index* is mapped to *i* and *j*. Then, each thread traverses in the *z* direction in a *for* loop by incrementing *k* index of the cells. Field values are updated for each *k*, thus the entire FDTD domain is covered.

```
block_dim_x = number_of_threads; block_dim_y = 1;
n_blocks_y = 1;
n_blocks_x = (nxx*nyy)/number_of_threads +
                              ((nxx*nyy)%number_of_threads ==0?0:1);
```

Listing 1. CUDA code to define block and grid sizes.

Unfortunately in FDTD updates the operations are dominated by memory accesses. In order to ensure high performance all global memory accesses shall be coalesced. In general an FDTD domain size would be an arbitrary number. In order to achieve coalesced memory access, the FDTD domain is extended by padded cells such that the number of cells in *x* and *y* directions is an integer

multiple of 16 as illustrated in Fig 2. The modified size of the FDTD domain becomes $Nxx \times Nyy \times Nz$, where Nxx, Nyy, and Nz are number of cells in x, y, and z directions, respectively.

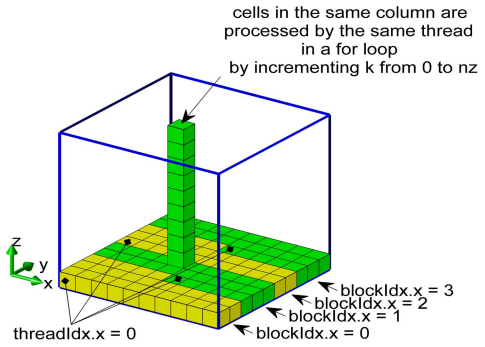

Figure 1. Mapping of threads to cells of an FDTD domain.

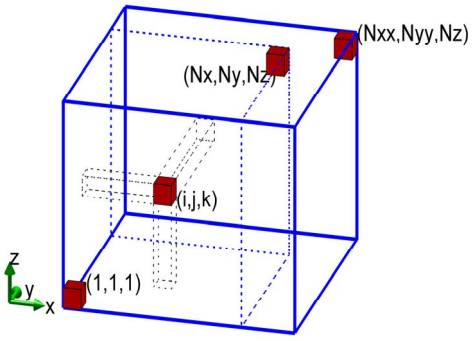

Figure 2. An FDTD problem space padded with additional cells.

```
__global__ void update_magnetic_fields_on_kernel(...)
{
extern __shared__ float sEz[];
int ci = blockIdx.x * blockDim.x + threadIdx.x;
int j  = ci/nxx;
int i  = ci - j*nxx;
int si = threadIdx.x;
int cizp;
float ex, exzp, ey, eyzp;
ey = Ey[ci];
ex = Ex[ci];
for (int k=0;k<nz;k++)
{
       cizp   = ci + nxx*nyy;
       exzp   = Ex[cizp];
       eyzp   = Ey[cizp];
       sEz[si]= Ez[ci];
       if (threadIdx.x<16)
              Ez[blockDim.x+threadIdx.x] = Ez[ci+blockDim.x];
       __syncthreads();

       Hx[ci] = Chxh[ci]*Hx[ci]+ Chxey[ci]*(eyzp-ey)
                             + Chxez[ci]*(Ez[ci+nxx]-sEz[si]);
       Hy[ci] = Chyh[ci]*Hy[ci]+ Chyez[ci] * (sEz[si+1]-sEz[si])
                             + Chyex[ci] * (exzp-ex);
       ...
       ci = cizp;
       ey = eyzp;
       ex = exzp;
       }
}
```

Listing 2. A section of CUDA code to update magnetic field components.

After ensuring the coalesced memory access, data reuse in a *for* loop in z direction and appropriate use of shared memory a CUDA code is developed. A section of this code is shown in Listing 2. The developed algorithm is tested on an

NVIDIA® Tesla™ C1060 Computing card. Size of a cubic FDTD problem domain has been swept and the number of million cells per second processed is calculated as a measure of the performance of the CUDA program. The result of the analysis is shown in Fig. 3. It can be observed that the code processes about 450 million cells per second on the average.

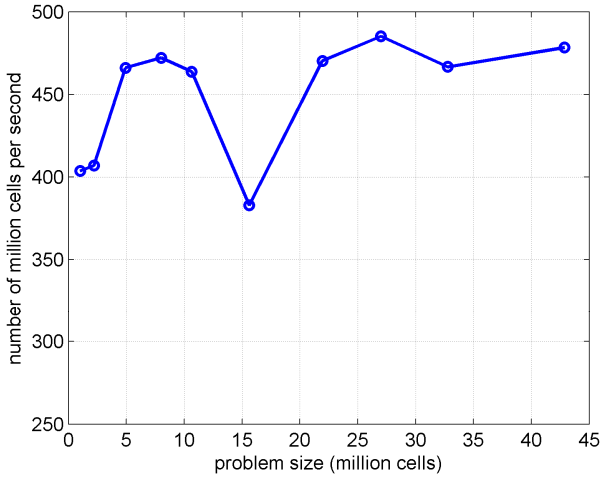

Figure 3. FDTD algorithm speed versus problem size.

References

[1] Buck, *Brook Spec v0.2* Stanford, CT: Stanford Univ. Press, 2003.

[2] M. J. Inman and A. Z. Elsherbeni, "Programming Video Cards for Computational Electromagnetics Applications," *IEEE Antennas and Propagation Magazine*, vol. 47, no. 6, pp. 71–78, December 2005.

[3] NVIDIA CUDA ZONE, http://www.nvidia.com/object/cuda_home.html.

[4] N. Takada, T. Shimobaba, N. Masuda, and T. Ito, "High-speed FDTD Simulation Algorithm for GPU with Compute Unified Device Architecture," *IEEE International Symposium on Antennas & Propagation & USNC/URSI National Radio Science Meeting*, *2009*, North Charleston, SC, United States, p. 4, 2009.

[5] Valcarce, G. De La Roche, A. Jüttner, D. López-Pérez, and J. Zhang,"Applying FDTD to the coverage prediction of WiMAX femtocells," *EURASIP Journal on Wireless Communications and Networking*, February 2009.

[6] P. Sypek, A. Dziekonski, and M. Mrozowski, "How to Render FDTD Computations More Effective Using a Graphics Accelerator," *IEEE Transactions on Magnetics*, vol. 45, no. 3, pp. 1324–1327, 2009.

[7] Ong, M. Weldon, D. Cyca, and M. Okoniewski, "Acceleration of Large-Scale FDTD Simulations on High Performance GPU Clusters," *2009 IEEE International Symposium on Antennas & Propagation & USNC/URSI National Radio Science Meeting*, North Charleston, SC, United states, 2009.

[8] Atef Elsherbeni and Veysel Demir, "The Finite Difference Time Domain Method for Electromagnetics: With MATLAB Simulations," SciTech Publishing, 2009.