Accelerated Plasma Simulations using the FDTD Method and the CUDA Architecture

Wei Meng[#1], Yufa Sun[#2]
[#]Key Lab of Intelligent Computing & Signal Processing, Ministry of Education, Anhui University
Hefei, P.R.China
[1]mengwee@sohu.com
[2]yfsun@ahu.edu.cn

***Abstract*-This letter presents the graphic processor unit (GPU) implementation of the finite-difference time-domain (FDTD) method for the solution of the two-dimensional electromagnetic fields inside dispersive media. An improved Z-transform-based finite-difference time-domain (ZTFDTD) method was presented for simulating the interaction of electromagnetic wave with unmagnetized plasma. By using the newly introduced Compute Unified Device Architecture (CUDA) technology, we illustrate the efficacy of GPU in accelerating the FDTD computations by achieving significant speedups with great ease and at no extra hardware cost. The effect of the GPU-CPU memory transfers on the speedup will be also studied.**

***Keywords*-GPU;FDTD;Z-transform;CUDA**

I. INTRODUCTION

Modelling the electromagnetic radiation can be done using a variety of approaches, however, the finite-difference time-domain (FDTD) approach is perhaps the most common [1]. The popularity of this method continues to rise due to its simplicity and robustness in solving complex electromagnetic problems [2]. Z-Transform-based finite-difference time-domain (ZTFDTD) method has universal applicability for various dispersive medium [3]. Although the method has existed for over two decades, enhancements to improve the computational time are continuously being published [4], of special interest are parallel implementations.

Present-day computers with powerful graphics processing unit (GPU) show considerable promise of increased performance for the electromagnetic model. Order of magnitude increases in computing speed have been reported for wave equation solutions, for example [5]. The increased performance can potentially be realized for a modest price without the complications of parallel processing on multi-core machines. Problem-independent parallelization is can be built into the basic model. GPU promise a relatively low cost boost which leverages the vast development driven by the game industry. Potential benefits dictate investigation of efficacy in electromagnetic plasma calculation models.

This article aims to provide empirical results into the efficacy of using a rather recent and somewhat different approach in adapting the ZTFDTD method for parallel computing. The approach makes use of the CUDA architecture. CUDA is the computing engine in NVIDIA GPU. GPU have a parallel "many-core" architecture, each core is capable of running thousands of threads simultaneously. NVIDIA CUDA GPU provide a cost-effective alternative to traditional supercomputers and cluster computers. To place this work in the context of various dispersive medium applications, a ZTFDTD program is developed to operate on the NVIDIA CUDA architecture. This program is subsequently used to model a variety of electromagnetic plasma calculation models. Experiments are conducted to quantify the speedup obtained.

The devices supporting CUDA include NVIDIA G80 series or above. In this paper, A GTX650Ti video card with a GK106 core is used to execute ZTFDTD program with CUDA. GTX650Ti GPU contains 4 multiprocessors (SMX), each of which composed by 192 streaming processors and other devices, such as 48K shared memory, instruction cache. And on the video card is 1GB memory can be used as global memory or texture in CUDA. Kepler GK106 supports the new CUDA Compute Capability 3.1 [6].

II. OVERVIEW OF THE CUDA ARCHITECTURE

A. CUDA Programming Model

CUDA is the hardware and software architecture introduced by NVIDIA in November 2006 [7] to provide developers with access to the parallel computational elements of NVIDIA GPU. The CUDA architecture enables NVIDIA GPU to execute programs written in various high-level languages such as C, Fortran, OpenCL and DirectCompute. Compared to the CPU, the GPU devotes more transistors to data processing rather than data caching and flow control. This allows GPU to specialize in mathematical-intensive, highly parallel operations compared to the CPU which serves as a multi-purpose microprocessor.

CUDA has a single-instruction multiple-thread (SIMT) execution model where multiple independent threads execute concurrently using a single instruction [8]. CUDA GPU has a hierarchy of grids, threads and blocks as shown in Fig. 1. A thread can be considered the execution of a kernel with a given thread index. Each thread uses its unique index to access elements of the dataset passed to the kernel. Thus the collection of all the threads cooperatively processes the entire dataset concurrently. A block is a group of threads while a grid is a group of blocks. Each thread has its own private memory. Shared memory is available per-block and global

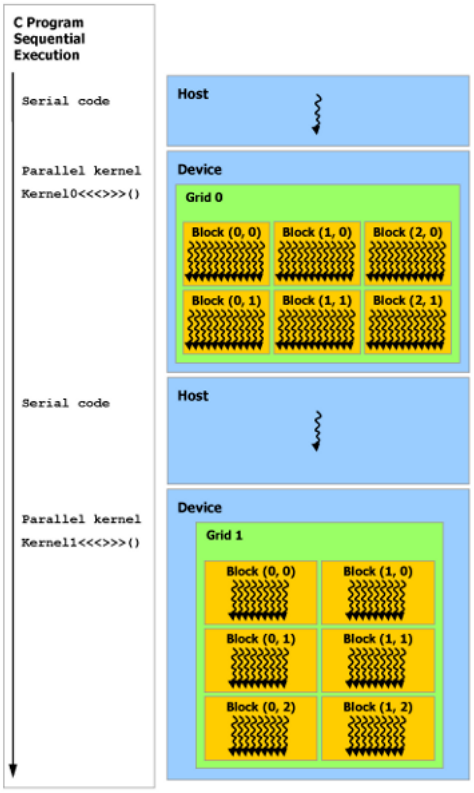

Figure 1. CUDA heterogeneous programming model.

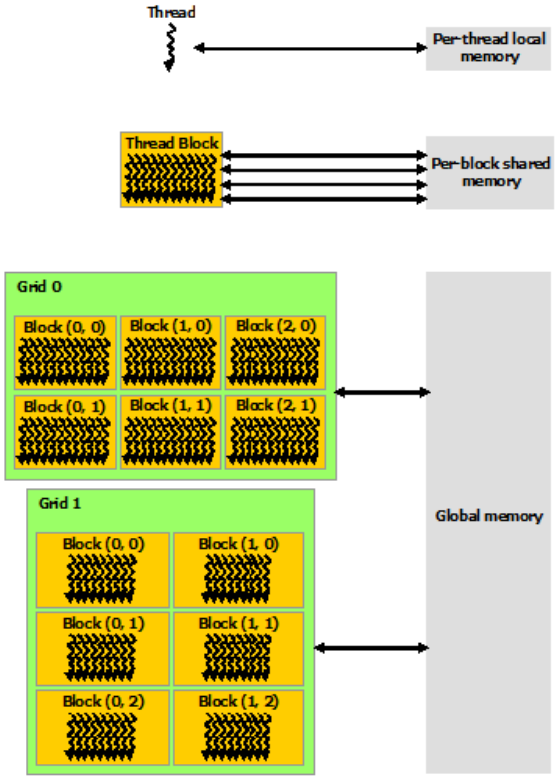

Figure 2. CUDA memory hierarchy model

memory is accessible by all threads. This multi-threaded architecture model puts focus on data calculations rather than data caching. Thus, it can sometimes be faster to recalculate rather than cache on a GPU. A CUDA program is called a kernel which is invoked by a CPU program.

B. *CUDA Memory Model*

The CUDA programming model assumes that CUDA threads execute on a physically separate device (GPU). The device is a co-processor to the host (CPU) which runs the program. CUDA also assumes that the host and device both have separate memory spaces: host memory and device memory, respectively. Because host and device both have their own separate memory spaces, there is potentially a lot of memory allocation, reallocation and data transfer between host and device as shown in Fig. 2. Thus, memory management is a key issue in general purpose GPU computing. Inefficient use of memory can significantly increase the computation time and mask the speedup obtained by the data calculations. While global memory is located off-chip and has the longest latency, there are techniques available that can reduce the amount of GPU clock cycles required to access large amounts of memory at one time. This can be done through memory coalescing. Memory coalescing refers to the alignment of threads and memory. For example, if memory access is coalesced, it takes only one memory request to read 64-bytes of data. On the other hand, if it is not coalesced, it could take up to 16 memory requests depending on the GPU's compute capability. Choice of the grid and block configurations is crucial in obtaining purely coalesced memory access.

III. CUDA IMPLEMENTATION

A. *ZTFDTD Formulation*

In the following, we present the analysis of TM-polarized plane wave penetration through two-dimensional dispersive media. As such, we follow the procedure outlined in [9] for ZTFDTD for the unmagnetized plasma along with the UPML boundary condition, which is not only efficient in minimizing the memory requirements, but also the most accurate form of absorbing material. This, in turn, yields the following form of the FDTD update equation for the *x*-component of the electric field:

$$E(k)^n = \frac{(\Delta t/\varepsilon_0)\cdot\left[\nabla\times H - e^{-\nu_c\cdot\Delta t}\cdot J(k)^{n-1}\right] + E(k)^{n-1}}{1+\omega_p^2\Delta t^2} \quad (1)$$

$$J(k)^n = e^{-\nu_c\cdot\Delta t}J(k)^{n-1} + \omega_p^2\varepsilon_0\Delta t\cdot E(k)^n \quad (2)$$

where , Δt is the timestep, ε_0 is the permittivity of vacuum, ν_c is the electron collision, ω_p is the plasma angular frequency, $J(k)^n$ is the auxiliary array for $E(k)^n$.

B. *CUDA ZTFDTD Program*

In this paper, we present ZTFDTD Parallel implementation on GPU. According to the basic FDTD algorithm on CPU, we present its GPU implementation shown in Fig. 3.

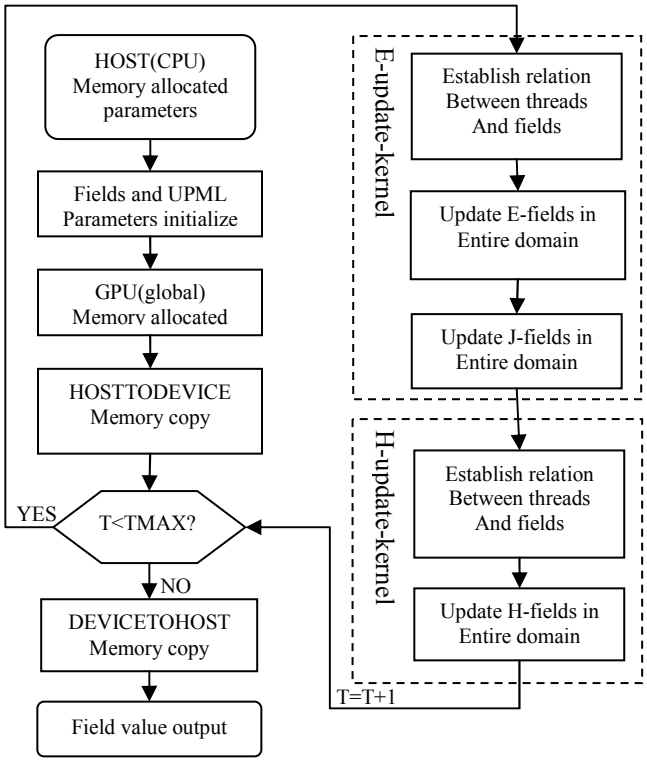

Figure 3. Flowchart of ZTFDTD algorithm on GPU.

Only procedures in the dot line frame are operated on GPU and when a kernel called a grid of threads is established. Because there is not any synchronization function for threads of whole grid while we have to use the value of former step, two kernels are launched sequentially to give a barrier for all threads. The number of threads is decided by the number of Yee cells (field size) because we make one thread to process one cell's spatial calculation.

IV. ZTFDTD Simulation Of CUDA

The ZTFDTD experiments based on CUDA are taken to research the parallel performance to solve complex computational problems of simulating the interaction of electromagnetic wave with unmagnetized plasma.

A. Specifications of the Test Platform

As mentioned before, one Geforce GTX650Ti video card, contained in a PC with Intel Core i3-2120(3.3GHz) CPU and 4GB RAM, is used for our simulation based-on CUDA. And for comparison, the same simulation with standard C code is operated on a computer with Intel Quad-Core i7-3820(3.6GHz) CPU computer and 64GB RAM (see Table I). Performance results and comparison are obtained by calculating time for specific number of time steps (iterations). And runtime of CPU program is only ZTFDTD iteration time, not including the memory allocating and initialing time. For CUDA, it also includes allocating time for global memory, time for copying memory form CPU RAM to GPU RAM and back, but these procedures execute only once. Then speedup is defined as the result GPU runtime divided by CPU runtime. The CUDA compilation is Microsoft Visual studio 2010 with CUDA v5.0 (64bit).

TABLE I
HARDWARE SPECIFICATIONS

Compute Device	i7-3820	GTX650Ti
Frequency(MHz)	3600	925
core /stream processor	4	768
floating point calculations per second(GFlops)	57.6	2131.2

B. Algorithm to Achieve

Taking the two-dimensional TM wave as an example, we simulate a plane wave impinging on a unmagnetized plasma cylinder. Incident wave is a simple Gaussian pulse source generated in the middle of the problem space and 8-point UPML as the absorbing boundaries shown in Fig. 4.

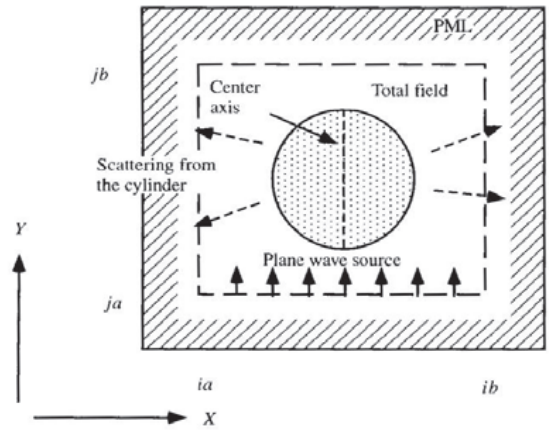

Figure 4. Diagram of the simulation of a plane wave striking a unmagnetized plasma cylinder [10].

To simulate a plane wave interacting with a unmagnetized plasma cylinder using parallel algorithm, we use each thread to calculate an electric field in Yee cell according to the parallel property of ZTFDTD. CUDA implement programs with the unit of warp. A half-warp is either the first or second half of a warp, which is an important concept for memory accesses because in order to hide the latency effectively, half of a warp should be performed at least at a time. A warp contained 32 threads. So, we make each Block has 32 * 16 Threads. When running, each thread in the block will have 512 threads to execute at the same kernel.

C. Texture memory Optimization

Optimizing the CUDA algorithm most often involves optimizing data accesses, which includes the use of the various CUDA memory spaces. Texture memory provides great capabilities including the ability to cache global memory .In order to speedup the parallel algorithm, a texture optimization algorithm was presented [11]. Texture cache is on-chip memory, which has good processing speed. Texture

memory is designed originally for dealing with graphics which have a large number of spatial localities when we access memory. According to the characteristics of texture memory that is read-only memory, we put the parameters of UPML boundary conditions and dielectric cylinder into the texture memory and realize optimization.

D. Performance Calculation

Throughout this article, speedups and throughputs are used to quantify. Speedup is defined as the improvement in simulation time with respect to the ZTFDTD program being computed using only on the CPU. Thus:

$$Speedup = \frac{CUP\ Execution\ Time}{GUP\ Execution\ Time} \tag{3}$$

Throughput is defined as number of finite difference points that are updated per second:

$$Throughput(MCell/s) = \frac{I \cdot J \cdot T}{10^6 \tau} \tag{4}$$

Where τ is the execution time in seconds and T the number of iterations. I, J are the number of cells in two directions respectively. In this instance we have used mega-cells per second.

V. Results

Fig. 5 and 6 give the measured throughput and speedup for simulating the interaction of electromagnetic wave with unmagnetized plasma. Fig. 5 shows the throughput of the CPU version of the FDTD code stays constant at approximately 2.3 MCells/s. The performance of the CUDA implementation varies from 33 to 51 MCells/s, and kept around 50 at last. Speedup of 5000 iterations for every case are illustrated in Fig. 6, which shows that 17-21 speedups can be obtained when cells number larger than 10^6.

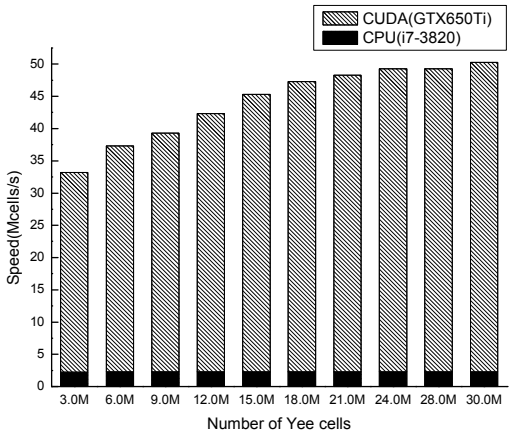

Figure 5. Throughput of CPU/CUDA ZTFDTD program.

VI. Conclusion

Using GTX650Ti video card, CUDA based code has shown good performance compared with C code operated on i7-3820 CPU in 2D FDTD simulations. The speedups are archived 14

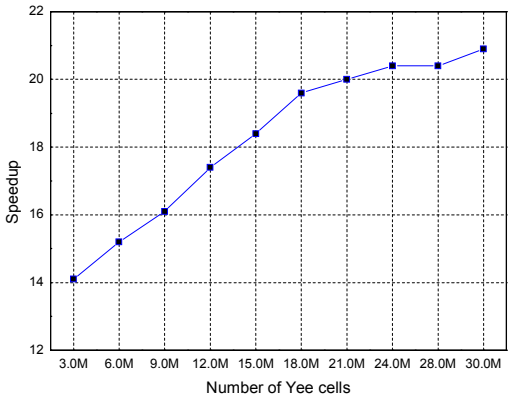

Figure 6. Obtained speedup of the CUDA program.

in all the cases we have tested. And for more common cases (number of Yee cells is larger than 10^7), it is up to 21 at last. The card is displaying for operating system at the same time of FDTD simulation. So it is expected that better performance would be obtained using Tesla C2070 GPU without video output port [12], which is designed for scientific computation with CUDA.

Acknowledgment

This work was financed by the National Natural Science Foundation of China (Grant No.61172020), the 211 Project of Anhui University (Grant No.ZYGG201202) and the academic innovation Foundation of Anhui University (Grant No.01001770-10117700481).

References

[1] Ge Debiao and Yan Yubo, *Finite-Difference Time-Domain Method for Electromagnetic Waves*, 3rd ed., Xi'an: Xidian University Press, 2011.

[2] A. Taflove and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 3rd ed. Norwood, MA: Artech House, 2005.

[3] Sullivan D M, "Frequency-dependent FDTD methods using Z transforms," IEEE Trans. Antennas Propagat, pp.1223-1230, 1992.

[4] Liu Shaobin, Liu Song and Hong Wei, *The Frequency-Dependent Finite-Difference Time-Domain Method*, Beijing: Science Press, 2010.

[5] P. Micikevicius, "3D Finite Difference Computation on GPUs using CUDA", ACM International Conference report, 2009.

[6] NVIDIA Corporation Technical staff, *Kepler GK110 Architecture Whitepaper*, NVIDIA Corporation, 2012.

[7] D. B. Davidson, "Development of a CUDA Implementation of the 3D FDTD Method," IEEE Antennas and Propagation Magazine, Vol. 54, No. 5, Oct. 2012.

[8] NVIDIA Corporation Technical staff, *NVIDIA CUDA C Programming Guide 5.0*, NVIDIA Corporation, 2012.

[9] Liu Jianxiao，Yin Zhihui "Improved Z-FDTD Method Analysis of 3D Target with Non-Magnetized Plasma," Journal of Microwaves ,Vol.28 No.2 Apr.2012.

[10] Sullivan D M. *Electromagnetic Simulation Using the FDTD Method.* New York: IEEE Press, 2000, pp.49-62.

[11] J. Sanders and E. Kandrot, *CUDA By Example an Introduction to General-Purpose GPU Programming*. American: Pearson Education, 2010, pp.84-101.

[12] Tomoaki Nagaoka and Soichi Watanabe, "Accelerating three-dimensional FDTD calculations on GPU clusters for electromagnetic field simulation," EMBC, 2012, pp.5691 – 5694.