Parallel Implementation of the Finite-Difference Time-Domain Method in Open Computing Language

T. P. Stefański[12] S. Benkler[3] N. Chavannes[3] N. Kuster[2]

Abstract – **In this paper we evaluate the usability and performance of Open Computing Language (OpenCL) targeted for implementation of the Finite-Difference Time-Domain (FDTD) method. The simulation speed was compared to implementations based on alternative techniques of parallel processor programming. Moreover, the portability of OpenCL FDTD code between modern computing architectures was assessed. The average speed of OpenCL FDTD simulations on a GPU was about 1.1 times lower than a comparable CUDA based solver for domains with sizes varying from 50^3 to 400^3 cells. Although OpenCL code dedicated to GPUs can be executed on multi-core CPUs, a direct porting does not provide satisfactory performance due to an application of architecture specific features in GPU code. Therefore, the OpenCL kernels of the developed FDTD code were optimized for multi-core CPUs. However, this improved OpenCL FDTD code was still about 1.5 to 2.5 times slower than the FDTD solver developed in the OpenMP parallel programming standard. The study concludes that, despite current performance drawbacks, the future potential of OpenCL is significant due to its flexibility and portability to various architectures.**

1 INTRODUCTION

A broad range of computational electromagnetics problems can be solved using the Finite-Difference Time-Domain (FDTD) method [1]. Time evolution of the electromagnetic field is calculated in this method using central-difference approximations of the partial derivatives in the Maxwell's curl equations. Due to the possibility of a wideband frequency response calculation with a single simulation run, straightforward implementation of arbitrarily shaped structures consisting of dispersive and non-linear materials and easy parallelization, the method has been successfully applied in microwave and antenna engineering, bio-electromagnetics, electromagnetic compatibility and photonics [1].

Recently, Graphics Processor Units (GPUs) became a source of a cheap computational power for the acceleration of FDTD codes [2], [3]. Moreover, the introduction of the Compute Unified Device Architecture (CUDA) parallel programming model and GPUs with enhanced computational power by Nvidia [4] delivered resources enabling high performance computing on desktop workstations. However, existing GPU accelerated FDTD codes do not fully deploy the computational power of the multi-core central processing unit (CPU), which is always present in any computer. Also, GPU codes have not normally been portable between hardware devices manufactured by different vendors. In general, due to scaling of the processor parallelism according to Moore's law, it is a real challenge to develop scientific codes that are not only portable between the very specific hardware architectures (e.g. CPUs and GPUs) available on the market, but will also transparently scale their parallelism in the future. Open Computing Language (OpenCL) [5] seems to be a remedy for overcoming these challenges as it maintains portability between hardware architectures and efficiency of the low-level programming interface. OpenCL is a framework for parallel programming of heterogeneous platforms consisting of multi-core CPUs, GPUs, and other modern processors, e.g., the Cell Broadband Engine. This standard opens the way to build heterogeneous computing systems which may simultaneously deploy the computational horsepower of multi-core CPUs and GPUs. Such mixed solutions may allow different types of processing units to be used for the tasks best suited to them. OpenCL unifies the process of code development for heterogeneous computing systems using one programming environment (compiler) to target substantially different processing elements.

This paper presents results of the OpenCL FDTD code evaluation. In this contribution we only focused on portability and efficiency tests performed on multi-core CPU machine supported by a single GPU. In Section 2, implementation of the FDTD method in OpenCL is introduced. Evaluation of the FDTD code performance is presented in Section 3, with characteristics of the simulation throughput as a function of the domain size in Section 3.1 and application examples benchmarking code implementation in Section 3.2.

2 FDTD IMPLEMENTATION IN OPENCL

The flowchart of the FDTD method is shown in Fig. 1.

[1] Integrated Systems Laboratory, Swiss Federal Institute of Technology, Gloriastrasse 35, 8092 Zurich, Switzerland.
[2] Foundation for Research on Information Technologies in Society, Zeughausstrasse 43, 8004 Zurich, Switzerland.
[3] SPEAG Software R&D, Zeughausstrasse 43, 8004 Zurich, Switzerland.
E-mails: stefanski@itis.ethz.ch, benkler@speag.com, chavanne@speag.com, kuster@itis.ethz.ch.

978-1-4244-7368-7/10/$26.00 ©2010 IEEE

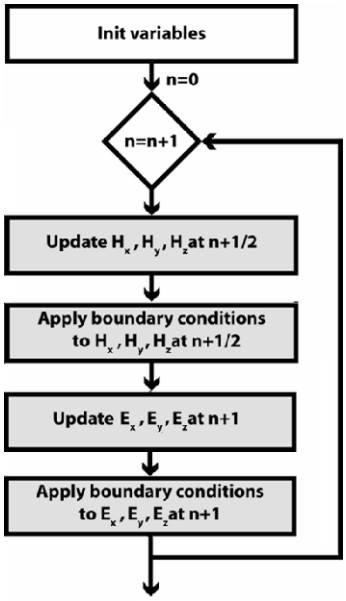

Figure 1: Flowchart of the FDTD method.

Each step of the electromagnetic field update consists of (i) H-field components update, (ii) application of boundary conditions to the H-field, (iii) E-field components update, and (iv) application of boundary conditions to the E-field. The FDTD update equations [1] are omitted here for the sake of brevity.

In the case of OpenCL implementation on a GPU, electromagnetic field and material data are stored in the global memory of the GPU as arrays. The *m*-th element in the data array is associated with *(i,j,k)* cell in the discretized *(x,y,z)* Cartesian space by the formula:

$$m = \left[i \cdot size(y) + j \right] \cdot size(z) + k$$

The *x*-direction is called the fastest direction and the *z*-direction is called the slowest direction. Transfer of data between GPU and CPU memory spaces is solely related to an acquisition of simulation results during time-marching. To minimize overhead of communication between the global memory and the processing unit on the graphics card, reuse of data resulting from the spatial locality of the FDTD method was employed (previous value and local derivatives of the E (H) field are required to update single H (E) field component values). Therefore, updates are performed in the loop along the slowest direction Rectangular *zy*-tiles of the field values, being of the same size as blocks of threads, are transferred to the low-latency local (shared) memory using coalesced memory access [4]. A single thread in a rectangular block of threads updates a single cell per one loop step, using two *zy*-tiles of data recently downloaded into the local memory.

In the case of OpenCL implementation on a CPU, the strategy described above cannot be used because the CPU does not have shared memory. Even though it was possible to execute the OpenCL GPU code described above on a CPU, the obtained simulation throughput was very low. Therefore, update kernels were optimized to avoid usage of the local memory. In this improved implementation every thread in the cuboidal group of threads updates a single cell directly using data stored in the global memory.

3 NUMERICAL RESULTS

OpenCL FDTD code was developed based on in-house written CUDA FDTD code whose performance is similar to existing commercial GPU solvers, e.g. [6]. However, OpenCL implementation does not use texture memory on a GPU like our CUDA implementation (using *cudaBindTexture* and *tex1Dfetch* functions) since OpenCL handles it using 2D and 3D image objects and texture memory space implementation would require address conversions. Data that were stored in texture memory in the CUDA implementation are stored in constant memory in our OpenCL implementation.

The code developed allows the mesh to be terminated with electric wall, magnetic wall, Mur 1st order absorbing boundary condition (Mur ABC), and periodic boundary conditions. The test simulations presented below were run on a personal computer with an Intel Core i7 920 multi-core CPU and a Nvidia C1060 GPU. Reference CPU solutions, presented for the sake of comparison, were developed in the OpenMP parallel programming standard and compiled with Visual Studio 2008.

3.1 Efficiency as a function of the domain size

Simulation results were obtained for the vacuum bounded by Mur ABCs and excited by a dipole antenna located in its centre. The size of the cubic domain was varied in the range 50^3-400^3 mesh cells. Peak simulation speeds as a function of the domain size are shown in Fig. 2 for the OpenCL FDTD code executed on a CPU and a GPU, and for OpenMP (CPU) and CUDA (GPU) implementations. The relative speedup factors, calculated as ratios of the simulation speeds, are shown as a function of the domain size in Fig. 3.

Averaged peak speed of the optimized OpenCL CPU FDTD solver in this test was equal to 49 Mcells/sec whereas the OpenMP FDTD solver gave 95 Mcells/sec. This represents a 1.9 times average speedup of the OpenMP vs. OpenCL CPU solver.

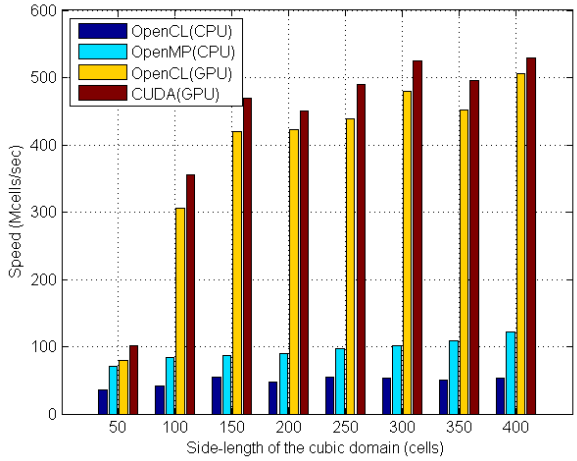

Figure 2: Speed of the FDTD OpenCL solver on a CPU and a GPU as a function of the side-length of the cubic domain; results of the CUDA (GPU) and OpenMP (CPU) implementation are presented for comparison.

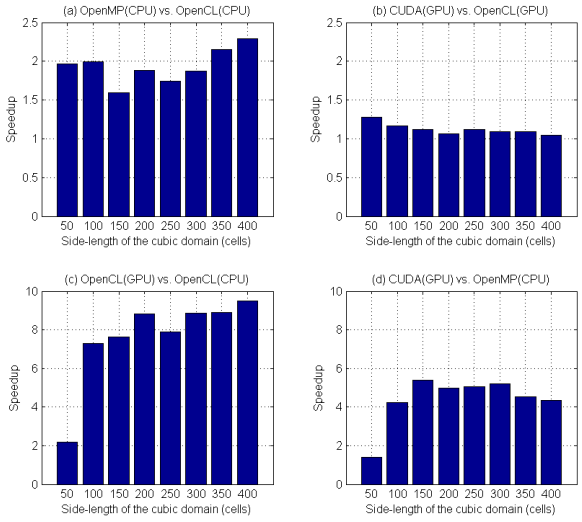

Figure 3: Speedup as a function of the side-length of the cubic domain: (a) OpenMP referenced to OpenCL - both executed on a CPU, (b) CUDA referenced to OpenCL - both executed on a GPU, (c) OpenCL (GPU) referenced to OpenCL (CPU), (d) CUDA (GPU) referenced to OpenMP (CPU).

However, OpenCL CPU code performance results from runtime compiler efficiency and optimization level (OpenCL kernel executables are compiled and built online). As is seen in Fig. 3(a), the relative performance of the CPU codes differs significantly, i.e. 1.5 - 2.5 times, as a function of the domain size.

The average peak performance of the OpenCL FDTD code executed on a GPU was equal to 388 Mcells/sec whereas CUDA FDTD code performance was equal to 427 Mcells/sec (the average peak speed of the recent version of the commercial GPU solver [6] was equal to 422 Mcells/sec in this test). This shows relative speedup of the CUDA FDTD vs. OpenCL GPU solver equal to about 1.1 times, see Fig. 3(b). It is noticeable that the GPU simulations ran significantly faster as the domain size increased. For relatively small domains, the advantages of GPU acceleration are not so visible.

As seen in Figs 3(c-d), relative speedups of OpenCL (GPU) vs. OpenCL (CPU) and CUDA vs. OpenMP are in the order of 7-10 and 4-6 times, respectively, for larger domains. This stems from the fact that OpenMP code is compiled by a commercial compiler whose performance is higher than the CPU compiler available in the OpenCL package. However, OpenCL code development requires only one programming environment for CPU and GPU targeting.

3.2 Application benchmark

The graphics user interface of the SEMCAD X commercial simulator [7] was employed for further tests.

3.2.1 IEEE SCC 34

This benchmark relies on simulation of a dipole antenna radiating a 835 MHz signal positioned next to a bowl filled with head imitating liquid (see Fig. 4 for the H-field cross-sectional plot). The computational domain was terminated by Mur ABCs and its size was equal to 72x82x96 cells. Simulations executed 7532 iterations. A comparison of the total simulation runtimes is presented in Table 1.

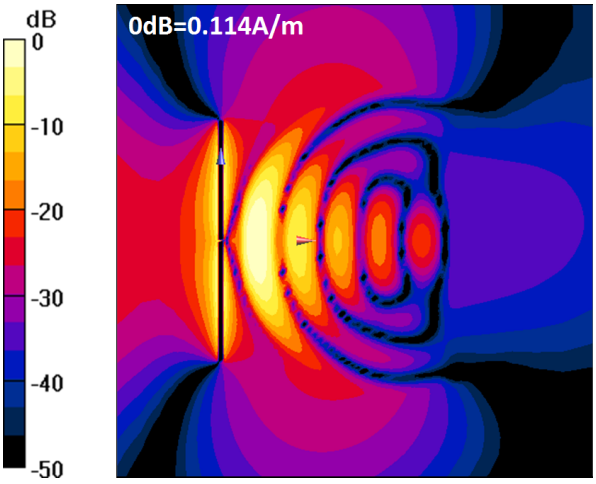

Figure 4: H-field cross-section obtained in the IEEE SCC 34 benchmark of the OpenCL (GPU) code, a dipole antenna is positioned next to a bowl filled with head simulating liquid.

CUDA FDTD implementation on a GPU was significantly faster than the other tested codes. Higher runtimes of the OpenCL FDTD solvers can partially be attributed to the online compilation of the OpenCL kernel sources during code execution.

OCL (CPU)	OMP (CPU)	OCL (GPU)	CUDA (GPU)	[6]
1:59	1:10	0:26	0:20	0:34

Table 1: Comparison of the total simulation runtimes in the IEEE SCC 34 benchmark for tested codes (min:sec).

3.2.2 PCB board level EMI

This benchmark relies on simulation of a cross-talk effect on unshielded PCB board (see Fig. 5). The computational domain was terminated by Mur ABCs and its size was equal to 536x1192x65 cells. Simulations on a GPU executed 223773 iterations. A comparison of the total simulation runtimes is presented in Table 2. It can be seen that OpenCL and CUDA runtimes on a GPU are similar in this benchmark.

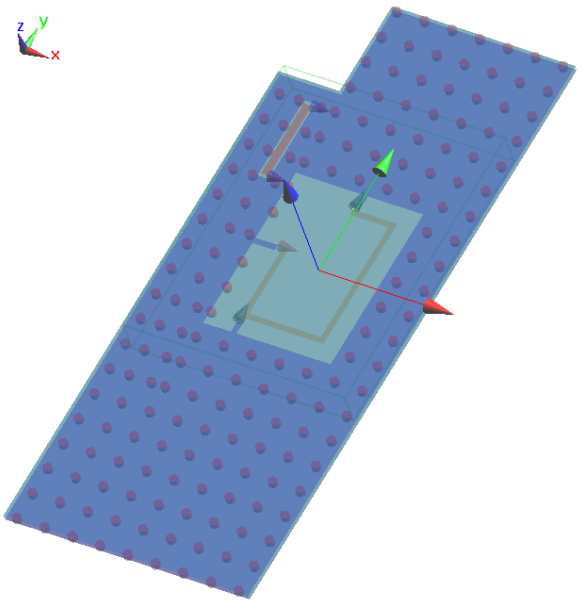

Figure 5: PCB used for EMI benchmark.

OCL (GPU)	CUDA (GPU)	[6]
7:06:22	7:00:16	8:39:16

Table 2: Comparison of the total simulation runtimes in the PCB board level EMI benchmark for tested GPU codes (hour:min:sec).

4 CONCLUSIONS

A usability and performance evaluation of OpenCL for implementation into the FDTD method has been presented. The developed GPU dedicated OpenCL FDTD code can be executed on multi-core CPUs, but satisfactory performance is only obtained after specific code optimization. Although the OpenCL FDTD simulations still perform at a lower speed than native CUDA or OpenMP implementations, it can be anticipated that the OpenCL framework will increase in popularity in coming years, and might become the standard with respect to parallel programming. Matured versions of this technology may result in a partial separation of the actual code development from the hardware specifications. Subsequently, the effort of tailoring code development to new hardware will be transferred to the developers of OpenCL compilers and hardware manufacturers, who have the required specific expertise thorough knowledge of the hardware architecture.

References

[1] A. Taflove and S. C. Hangess, Computational Electrodynamics: The Finite-Difference Time-Domain Method, 3rd ed. Boston, MA: Artech House, 2005.

[2] S. E. Krakiwsky, L. E. Turner, M. M. Okoniewski, "Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU)," 2004 IEEE MTT-S Int. Microwave Symp. Dig., pp. 1033-1036.

[3] M. J. Inman, A. Z. Elsherbeni, "Programming video cards for computational electromagnetics applications," IEEE Antennas & Propag. Magazine, vol. 47, no. 6, pp. 71-78, December 2005.

[4] NVIDIA CUDA Programming Guide, ver. 2.3.1, electronic file available at: http://www.nvidia.com/object/cuda_develop.html, 2009.

[5] The OpenCL Specification, ver. 1.0, Khronos OpenCL Working Group, electronic file available at: http://www.khronos.org/registry/cl/specs/opencl-1.0.48.pdf, 2009.

[6] Acceleware [Online], available at: http://www.acceleware.com, 2010

[7] Schmid & Partner Engineering AG [Online], available at: http://www.speag.com/speag/, 2010