Massively Parallel Two-Dimensional TLM Algorithm on Graphics Processing Units

Filippo V. Rossi[1], Poman P.M. So[1], Nikolaus Fichtner[2] and Peter Russer [2]

[1] Dept. ECE, University of Victoria, Victoria, British Columbia, Canada
[2] Institute for High Frequency Engineering, Technische Universität München, München, Germany

Abstract — Recent advances in computing technology has brought massively parallel computing power to desktop PCs. As multi-core processor technology becomes mature, a new front in parallel technology based on graphics processors has emerged. A massively parallel 2D-TLM algorithm for NVIDIA advanced graphics processors has been developed. The proposed parallel computing paradigm can be adopted straightforwardly to accelerate time-domain electromagnetic field modeling programs.

Index Terms — Time Domain Computational Electromagnetics, Parallel Algorithms, Stream Computing, FDTD, TLM, GPU, SIMD.

I. Introduction

Graphics processing unit (GPU) based parallel computing has been an important topic for the computing industry for a number of years. Macedonia addressed this topic in a computing magazine article in 2003 [1]. Most of the papers on GPU computing are related to signal and image processing [2 – 6]. Krakiwsky *et al.* applied the technique to accelerate the FDTD algorithm [7]. Takizawa *et al.* applied GPU computing to heat transfer simulation [8]. Z. Luo *et al.* and Harding *et al.* applied the paradigm to artificial neural network [9] and genetic algorithm [10], respectively. Hence, it is quite clear that parallel algorithms can be executed efficiently on GPUs. Furthermore, a cluster of GPU based computers can be created to execute grand challenge problems [11]. Researchers at Stanford [12] have been using this technique for years in protein folding computation. However, general adaptation of GPU processing for scientific computing, and hence in computational electromagnetics as well, has been quite slow due to the lack of a good software development library.

II. The Graphics Processing Unit (GPU)

The latest GPUs from NVIDIA are based on an innovative compute unified device architecture (CUDA) interface. The CUDA SDK enables programs to be developed with the SIMD paradigm using a high level programming language such as *C* or *C++*. The NVIDIA GeForce 8800 GTX GPU used in this project consists of 16 multiprocessors with 8 processors each for a total of 128 processors with 768 MB of GDDR3 global memory. Each multiprocessor contains 16k of on-chip shared memory which is much faster than global memory. There is 64k of constant cache which is used to pass instruction parameters. The single instruction multiple data (SIMD) programming model supported by the GPU is a very good software paradigm for implementing parallel FDTD and TLM algorithms.

III. Massively Parallel TLM Algorithm

This paper reports, for the first time, a massively parallel GPU based TLM algorithm for NVIDIA CUDA enabled GPUs. With even an entry-level GeForce 8800 GTX Ultra, the new program has achieved more than 10 times acceleration in speed when compare to an equivalent serial version running on a top-end DELL 690 workstation.

Fig. 1 depicts the schematic of our implementation. A TLM mesh, [13], on the host CPU is mapped to the GPU memory according to the CUDA SDK requirement, [14]. The CPU transfers its mesh data to the GPU global memory where a kernel function is executed in parallel on all 16 multiprocessors. The 16 multiprocessors partitions and transfers data from the global memory to their local shared memory before processing the TLM procedures in parallel, fig. 2a. It is important to note that the copy and write commands for transferring data between global and shared memory take more than 90 percent of time per kernel iteration. This is due to the fact that 400 – 600 clock cycles are needed for each read or write operation for accessing the global memory; this is very slow when compare to as low as 4 clock cycles for accessing on-chip memory. Hence, it is important to eliminate unnecessary data transfer between the two memory locations.

Thread synchronization is an important issue for SIMD programming. Fig. 2a depicts three fundamental operations of a TLM process, namely impulse scattering at nodes, impulse reflection at boundaries, and impulse interchanges among neighboring nodes. Within each multiprocessor, these three operations can be synchronized easily by calling a sync function at the end of each operation. Synchronization among multiprocessors (thread blocks), on the other hand, must be handled indirectly by algorithm specific logic. Fig. 3 depicts a solution that has been implemented. Initially, a block of 16×16 nodes is copied from the global memory to the on-chip shared memory labeled VS_1. Next scattering is executed on all 256 (or 16×16) nodes in VS_1 by the thread block. The results of scattering remain in VS_1. Impulse interchange is processed next and the result is store in VS_2. Boundary conditions are handled in a similar manner. Note that in fig. 3 shared memory, VS_2, is larger than VS_1 by one dimension in all directions to accommodate the voltage data that crosses boundaries. The footprint of writing the results of VS_2 to

978-1-4244-1780-3/08/$25.00 © 2008 IEEE

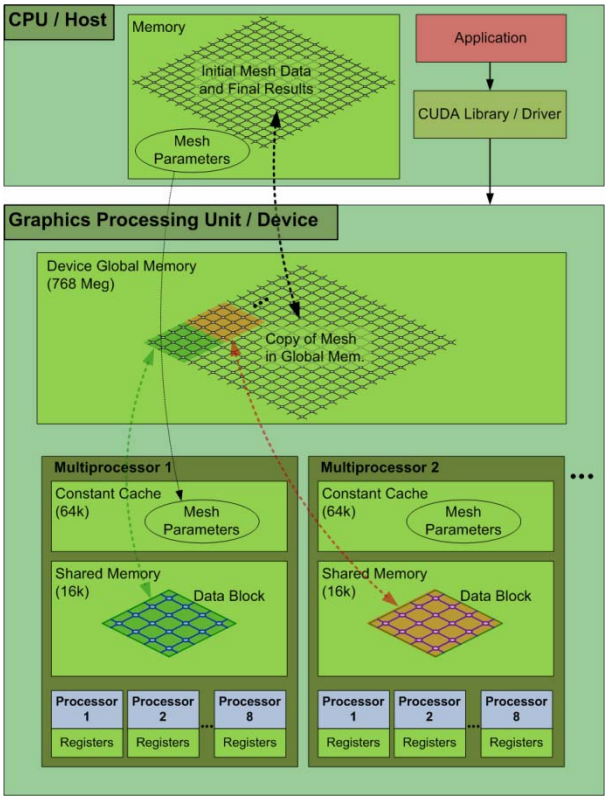


Fig. 1 The host-device relationship between a computer's CPU and GPU. Also depicted is the memory model of the NVIDIA GeForce 8800 GTX Ultra GPU used in this project.

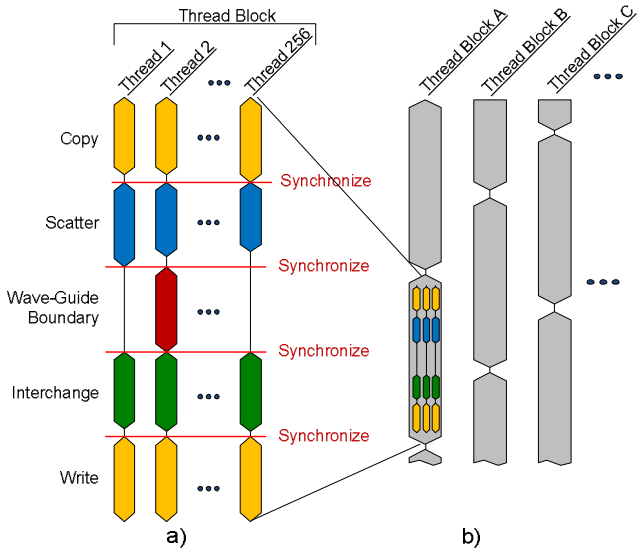


Fig. 2 a) Synchronization of computation stages within a thread block can be handled directly by a GPU sync function. b) Synchronization among thread blocks must be handled by other means.

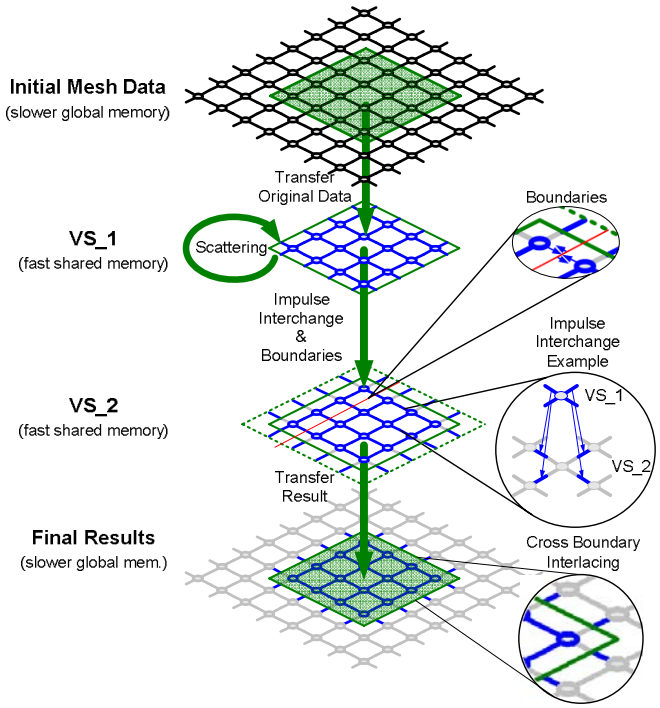


Fig. 3 Parallel computational iteration by thread block.

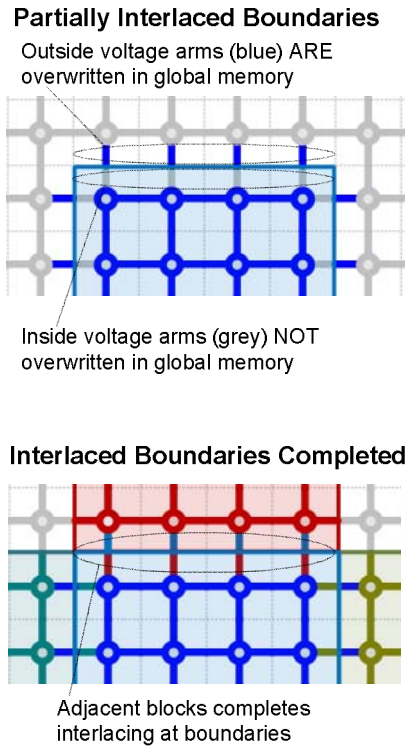


Fig. 4, Process of Inter-block dependence immunity

978-1-4244-1780-3/08/$25.00 © 2008 IEEE

global memory is depicted in fig. 4. Only the voltage arms in blue are written to global memory. Note that the voltage arms that are just inside the boundaries of the block (grey) are not written to global memory to allow interlacing. One important advantage seen by this technique is that all blocks in a mesh self-stitch, fig. 4. The overhead of this self-stitching 'radiative node method' is light compared to post kernel stitching approaches.

IV. Results

The performance of the TLM GPU routine was measured by timing the execution of increasingly larger mesh-node arrays (figure 5). As a comparison the same measurements were made for a serial CPU TLM routine and an OpenMP version that utilized 4 CPUs. The results in figure 5 indicate that at a lower number of nodes the performance of the parallel code diminishes; this can be attributed to factors such as high overhead-to-data density ratio for small mesh sizes. The GPU, therefore, offers no benefit when the mesh is smaller than a certain threshold. In this situation, the overhead in invoking the GPU code is high enough to render the GPU inefficient compared to the CPU.

As a second validation, the GPU-TLM code was used to model a WR28 waveguide band-pass filter. The field distribution, return loss and insertion loss of the filter are shown in fig. 6. The filter was also implemented in an OpenMP version of TLM code, and in MEFiSTo 2-D, [15]. Figure 7 shows a comparison of the performance results of each simulation run for this filter. MEFiSTo was configured to run using a single CPU as well as all 4 available CPUs on the Dell 690 workstation.

Our first attempt to port the two-dimensional shunt TLM algorithm to the GPU environment was very successful. It was recognized, however, that further improvements to speed of

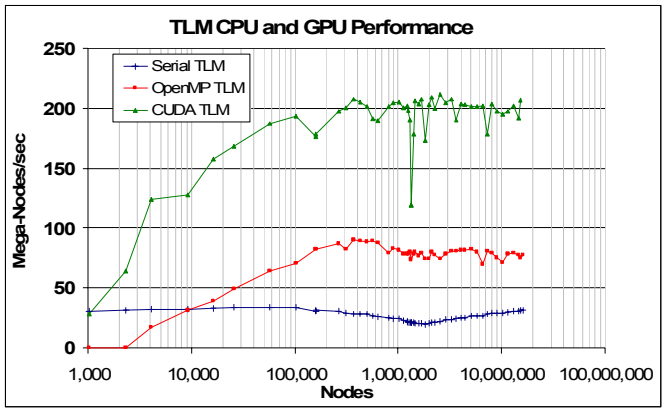
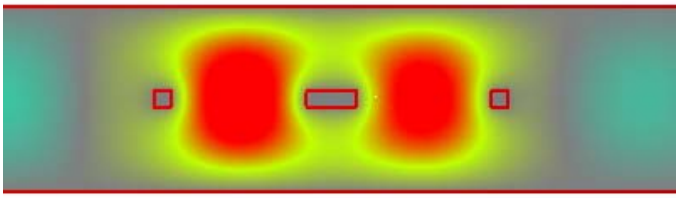
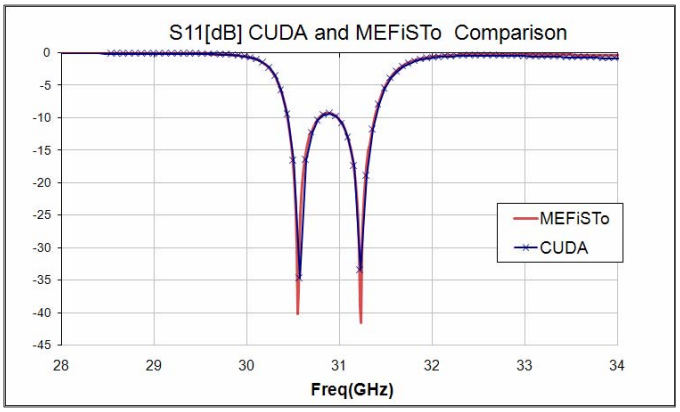


Fig. 5 Performance comparison between the traditional CPU and GPU TLM algorithms.





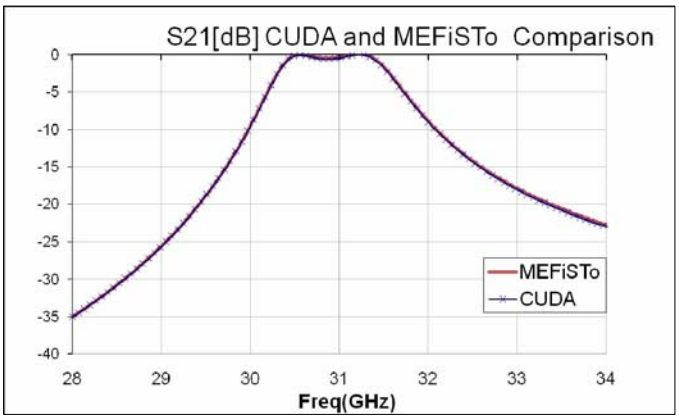


Fig. 6 The field distribution, return loss and insertion loss of a bandpass filter in a WR28 waveguide. The dimensions of the inductive obstacles are 0.6465mm × 0.6465mm and 0.6465mm × 1.9340mm. The spacing between the obstacles are 5.172mm.

execution can be achieved by applying additional optimization techniques, particularly contiguous memory organization. We will employ the techniques learned during this project to port the SCN TLM engine in Yatsim, [16, 17], to the NVIDIA GPU platform as soon as possible.

V. Conclusions

We have successfully designed and implemented a massively parallel 2D-TLM algorithm for the NIVIDIA CUDA enabled GPU. It is found that the SIMD software paradigm is very suitable for implementing time-domain computational electromagnetic methods such as TLM and FDTD. Our initial implementation has achieved more than 10 times improvement in speed.

978-1-4244-1780-3/08/$25.00 © 2008 IEEE

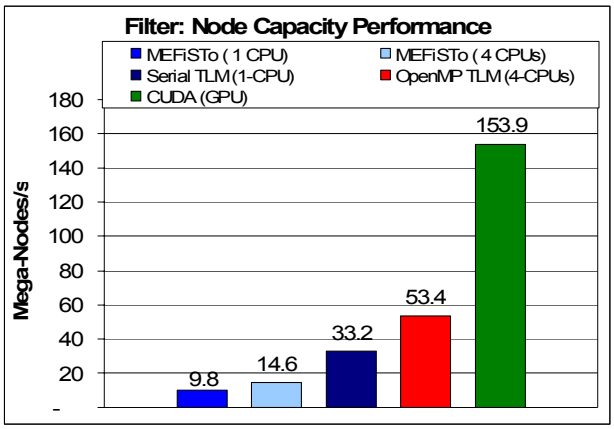


Fig. 7 Performance of various TLM implementations for a bandpass filter in a WR28 waveguide.

Structurally, the original TLM code is essentially a series of nested for-next loops. The complexity of adapting the original TLM program to the NVIDIA GPU architecture requires stream computing based parallel algorithms to be implemented. Novel strategies are necessary in cases that exhibit cross block synchronization dependencies, as seen in the example of the 'radiative node method'.

It is found that for two-dimensional shunt node TLM, our new GPU based algorithm can deliver up to 210 million nodes per second of simulation speed with just a single GPU. This is a significant increase in performance that warrants further research and development in computational electromagnetics procedures for this hardware platform. In particular, the innovative procedure described in this paper could be adopted in order to port three-dimensional FDTD and TLM algorithms to the NVIDIA GPU environment.

ACKNOWLEDGEMENT

The authors wish to acknowledge the financial support from the National Science and Engineering Research Council of Canada.

REFERENCES

[1] M. Macedonia, "The GPU Enters Computing's Mainstream", *IEEE Computer*, vol. 36, issue 10, pp.106 – 108, October 2003.

[2] G. Shen, G. P. Gao, S. Li, H. Y. Shum and Y. Q. Zhang, "Accelerating Video Decoding Using GPU", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, issue 5, pp. 685 – 693, May 2005.

[3] J.Y. Hong and M.D. Wang, "High speed processing of biomedical images using programmable GPU", *International Conference on Image Processing*, vol. 4, pp. 2455 – 2458, vol. 4. October 24 – 27, 2004.

[4] Y. Heng and L. Gu, "GPU-based Volume Rendering for Medical Image Visualization", *27th Annual International Conference on Engineering in Medicine and Biology*, pp. 5145 – 5148, 2005.

[5] O. Fialka and M. Cadik, "FFT and Convolution Performance in Image Filtering on GPU", *IEEE Proceedings of the Information Visualization*, pp. 609 – 614, July 05-07, 2006.

[6] J.S. Meredith, S.R. Alam and J.S. Vetter, "Analysis of a Computational Biology Simulation Technique on Emerging Processing Architectures", *IEEE International Symposium on Parallel and Distributed Processing*, pp. 1 – 8, March 26-30, 2007.

[7] S.E. Krakiwsky, L.E. Turner and M.M. Okoniewski, "Graphics Processor Unit Acceleration of Finite-Difference Time-Domain Algorithm", *Proceedings of IEEE International Symposium on Circuits and Systems*, vol.5, pp. V-265 – V268, May 23-26, 2004.

[8] H. Takizawa, N. Yamada, S. Sakai, and H. Kobayashi, "Radiative Heat Transfer Simulation Using Programmable Graphics Hardware", *5th IEEE/ACIS International Conference on Computer and Information Science*, pp. 29 – 37, July 10-12, 2006.

[9] Z. Luo; H. Liu; X. Wu, "Artificial Neural Network Computation on Graphic Process Unit", *Proceedings of IEEE International Joint Conference on Neural Networks*, vol. 1, pp. 622 – 626, Jul. 31 to Aug. 4, 2005.

[10] S. Harding, W. Banzhaf, "Fast Genetic Programming and Artificial Developmental Systems on GPUs", *21st International Symposium on High Performance Computing Systems and Applications*, pp. 2, May 2007.

[11] F. Zhe, Q. Feng, A. Kaufman and S. Yoakum-Stover, "GPU Cluster for High Performance Computing", *Proceedings of the ACM/IEEE Conference on Supercomputing*, pp. 47, 2004.

[12] folding.stanford.edu/FAQ-ATI.html

[13] Wolfgang J. R. Hoefer, "The Transmission-Line Matrix Method — Theory and Applications", *IEEE Transactions on Microwave Theory and Technique*, vol. MTT-33. No. 10, pp.882-893, October 1985.

[14] NVIDIA, developer.nvidia.com/object/cuda.html.

[15] Faustus Scientific Corporation, www.faustcorp.com.

[16] P. B. Johns, "A symmetrical condensed node for the TLM method," *IEEE Transactions on Microwave Theory and Technique*, vol-35, no. 4, pp. 370–377, April 1987.

[17] www.yatpac.org