Accelerated Implementation of the S-MRTD Technique Using Graphics Processor Units

G.Baron, E.Fiume, and C.D.Sarris
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, M5S 3G4, Canada
E-mail: gerard.baron@utoronto.ca, elf@dgp.toronto.edu, cds@waves.utoronto.ca

***Abstract*— A Time-Domain electromagnetic modeling technique, namely a high-order Scaling Function based MRTD (S-MRTD), can be dramatically accelerated, through its implementation in commodity graphics hardware. This implementation is achieved by mapping the numerical operations of S-MRTD to graphics operations, optimally executed by a graphics card, along the lines of previous work in the area of general purpose computing in computer graphics. The sustained speed-ups achieved, for two-dimensional problems, reach a factor of 30, significantly higher than any speed-ups reported for FDTD so far.**

I. Introduction

Almost ten years ago, the Multi-resolution Time-Domain (MRTD) technique was introduced in [1], [2], as a means of alleviating the limitations imposed by numerical dispersion on the choice of the cell size of the Finite-Difference Time-Domain (FDTD) method. Through homogeneous and inhomogeneous cavity numerical experiments and a Fourier dispersion analysis, [1], [2] showed the potential of this technique to achieve relatively small dispersion errors, at coarse discretization rates approaching the Nyquist limit of $\lambda/2$ (λ being the smallest simulated wavelength). However, this feature was not associated with the ability of the technique to support a multi-resolution grid, but with the choice of cubic spline (Battle-Lemarie) functions as a field expansion basis. In turn, the use of the latter resulted in update equations of added complexity and computational cost, compared to FDTD. Therefore, although the use of MRTD allowed for the reduction in the number of cells within a given domain, it increased the operations per cell compared to FDTD, partially counterbalancing the advantages it offered in the first place, in terms of simulation time. Still, a widely accepted advantage of the technique is its ability to yield accurate results even at coarse meshes, potentially reducing the number of required cells by almost two orders of magnitude in three-dimensional problems.

More recently, within the computer graphics community, the area of general-purpose computing has emerged [3], [4]. Although originally designed for the specialized task of accelerating video games, modern Graphics Processing Units (GPUs) can also be employed for the execution of numerical operations, as long as the latter can be disguised as image processing transformations, such as shading or texturing. The idea of utilizing GPUs for scientific computing has gained significant momentum, attracting wide research interest across multiple disciplines. In computational electromagnetics, this concept was introduced in [5], [6], which implemented a two-dimensional FDTD scheme equipped with periodic and Mur's absorbing boundary conditions, achieving acceleration rates of a factor close to 10. Larger acceleration rates were demonstrated when a Uniaxial Perfectly Matched Layer (UPML) set of equations was considered in [7], accompanied by an error analysis indicating that the errors produced by the GPU were within acceptable limits. Such a conclusion is reassuring for scientific computing applications, bearing in mind that graphics cards are intended to support visually convincing object rendering, rather than numerical accuracy.

The hardware acceleration of FDTD has been successfully pursued in the past, using custom made hardware such as FPGAs [8]. What makes the case of GPU-based scientific computing particularly attractive is that graphics hardware is very fast, very cheap, and continues to be enriched at rates that outstrip those of general purpose CPUs and FPGAs. However, it is currently limited by the fixed amount of memory available in a GPU.

This paper investigates the implementation of the MRTD technique on a GPU. This direction of research addresses the issue of the GPU memory limitation, by replacing the FDTD scheme with a much more memory efficient one. Furthermore, the numerical results of this paper suggest that the GPU performance actually improves with the arithmetic complexity of the programming involved and therefore, MRTD techniques are shown to be ideally suited to GPU acceleration, precisely because they employ more arithmetic operations per cell. Thus, what has long been considered as a drawback of this technique becomes an advantage as far as GPU acceleration is concerned. It is finally noted that the basis used for the MRTD field expansions is the Deslauriers-Dubuc bi-orthogonal interpolating basis [9]. As noted in [9], the use of this basis facilitates the application of localized boundary conditions (such as perfect electric conductors) and inhomogeneous media (such as a UPML), effectively addressing one more shortcoming of MRTD. Since no wavelets are involved, this scheme belongs to the S-MRTD class, under the terminology of [1].

II. S-MRTD as Image Processing and GPU Implementation

The GPU implementation of S-MRTD is achieved following the same concepts that guided the previous implementations of FDTD [5]-[7]. The S-MRTD time marching procedure can be converted to a state-space form, whereby a present state is determined from its immediately preceding values. Time-marching subsequently relates to the resources of stream computing by first, organizing present electric and magnetic field state and update-equations, into separate uniform variables and kernels respectively; and second, defining an input stream whose elements correspond uniquely to cells of a simulated mesh. Advancing a simulation a full time-step, by the so-called leap-frogging, involves two passes, where each sub-pass determines either a new electric or magnetic field state by applying the corresponding kernel. Upon conclusion of each sub-pass, the resulting output-stream contains either the present electric or magnetic field state and is used in subsequent updates. As for the efficacy of update-equations as stream-computing kernels, that is realized by recognizing update-equations as discrete spatial convolutions. For example, consider a 2-D S-MRTD electric field update-equation for simple media:

$$
\begin{aligned}
{}_{n+1}E^{y,\phi}_{i,j+\frac{1}{2},k} = {}_{n}E^{y,\phi}_{i,j+\frac{1}{2},k} &+ \frac{\Delta t}{\epsilon \Delta z} \sum_{p=-p_0}^{p_0-1} \alpha(p)\, {}_{n+\frac{1}{2}}H^{x,\phi}_{i,j+\frac{1}{2},k+\frac{1}{2}+p} \\
&- \frac{\Delta t}{\epsilon \Delta x} \sum_{p=-p_0}^{p_0-1} \alpha(p)\, {}_{n+\frac{1}{2}}H^{z,\phi}_{i+\frac{1}{2}+p,j+\frac{1}{2},k},
\end{aligned}
$$

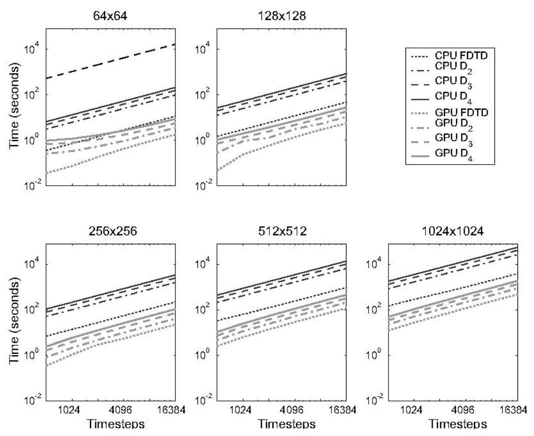
where $\alpha(p)$ are the so-called connection (or stencil) coefficients over a stencil p_0. Evidently, these coefficients provide the weights for the discrete spatial convolutions, that the second and third terms on the right hand-side can be interpreted as. In general, even the first term can be interpreted as a discrete spatial convolution with an impulse, so that all update-equations amount to sums of these convolutions. Hence, an analogy with image-processing emerges for the update-equations, by which the latter amount to image-processing filters. Then, the implementation of S-MRTD on a GPU boils down to mapping the spatial dimensions of the simulation region to the dimensions of the image files processed by the graphics card, packing field values and material properties into separate texture objects and their color channels and finally, designing Pixel-shaders that procedurally update electric and magnetic field components.

III. GPU-Accelerated S-MRTD: Performance Results

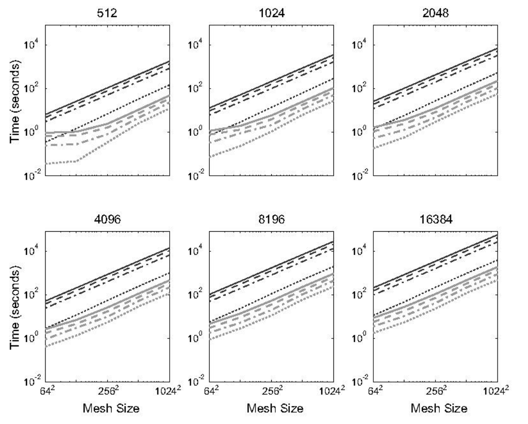
In this section, the relative performance of our GPU-accelerated and CPU-reference simulations is estimated through numerical experiments. Under consideration were square resonant free-space cavities with perfect electric conducting boundaries excited by a 1.0 GHz Gaussian pulse at their center. To evaluate the burden of mesh size on performance, four cavities of progressively larger dimensions of $\{64 \times 64\}$, $\{128 \times 128\}$, $\{256 \times 256\}$, $\{512 \times 512\}$, and $\{1024 \times 1024\}$ cells of dimensions $\{0.0025 \times 0.0025\}$ m^2 progressively represented fourfold increases in memory overhead. To evaluate the relative performance of different order S-MRTD schemes, these meshes were simulated with FDTD and S-MRTD based on the Deslauriers-Dubuc functions. It is noted that the latter are produced via an autocorrelation of the Daubechies functions and their connection coefficients are exactly those of the Daubechies S-MRTD [10]. In this paper's notation a D_k S-MRTD is produced by the coefficients of a Daubechies scaling function ϕ_k, with k vanishing moments and corresponds to a $2k-1$ order Deslauriers-Dubuc scheme. For all schemes, a time step equal of 0.5 of the FDTD stability limit was used. Finally, to evaluate the effect of simulation length, execution times were measured from start to 512, 1024, 2048, 4096, 8192, and 16384 time steps.

Figs. 1(a), 1(b) are logarithmic plots of measured GPU and CPU execution times for FDTD, D_2, D_3, and D_4 schemes in red and blue respectively, as a function of simulation length and mesh size. Recall that in log-log plots slope is indicative of an exponential tendency, whereas the relative separation of similar trends is indicative of a multiplicative factor between trends. All plots illustrate overwhelmingly linear relationships; a dashed black line of slope 1.0 (i.e. exponent zero) is included for reference in the upper left corner plot. Using the execution times of the CPU as baseline, Fig. 2 is derived, depicting the relative speed-up of GPU. The latter shows how FDTD and S-MRTD schemes asymptotically approach 10 and 30$\times$ speed-up per time-step per cell, respectively. This trend, and the relatively high-performance of S-MRTD schemes are explained by the fact that operations involved with the spatial convolutions that the update equations are mapped to, are pipelined in parallel in a GPU. As a result, scheme complexity, in terms of stencil size costs considerably less on the GPU. Furthermore, the GPU exhibits a comparative latency before beginning to process data. Since longer running simulations amortize this latency, it is most evident as a function of mesh size for our shortest length simulation (512 time steps). Less apparent for large meshes, the effect is on the order of the tens-to-hundreds of milliseconds and appears constant with respect to simulation scheme. With respect to Fig. 2, the latency dominates the performance of smaller meshes and accounts for their lower initial speed-ups.

For a fixed mesh size, FDTD is consistently faster than



(a) Execution time vs. number of time steps for different mesh sizes.



(b) Execution time vs. mesh size for different number of time steps.

Fig. 1. Dependence of execution time on mesh size and time steps.

S-MRTD. Note though that due to its superior dispersion performance, S-MRTD can achieve similar accuracy with FDTD, using a smaller mesh than FDTD. A factor of two reduction in the discretization rate of FDTD has been indicated as typically possible by previous S-MRTD dispersion analyses [1], [2], [10]. A comparison of the FDTD and D_3 S-MRTD execution time in Fig. 1(a), in $\{512 \times 512\}$ and $\{256 \times 256\}$ meshes respectively shows that the latter is faster than the former by a factor of 20. An indication of the relative performance improvement of both methods upon their GPU implementation is also provided by inspection of the separation of the FDTD and S-MRTD execution time curves (for the same mesh) for the CPU and GPU case; the large separations observed for the CPU implementations are greatly decreased for the GPU ones, paving the way for the gradual domination of S-MRTD over FDTD in terms of overall performance.

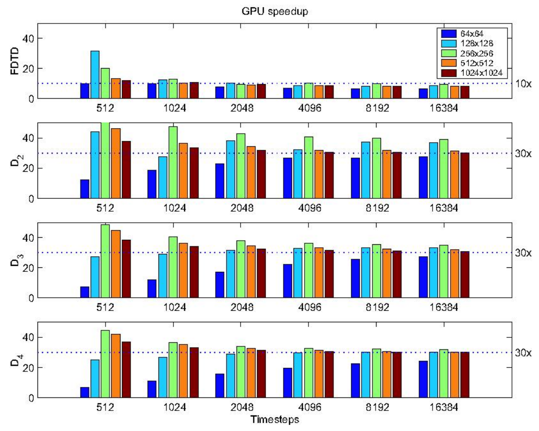
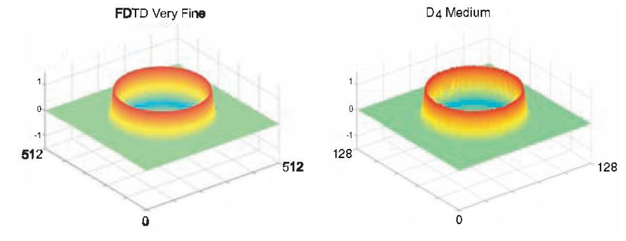


Fig. 2. GPU speed-up for for Deslauriers-Dubuc-based S-MRTD schemes; D_k implies a scheme of order $2k-1$.



(a) FDTD; $\{512 \times 512\}$ mesh.

(b) D_4 S-MRTD; $\{128 \times 128\}$ mesh.

Fig. 3. Gaussian pulse propagation in a rectangular domain, modeled by FDTD and S-MRTD.

Fig. 3 shows a relevant numerical experiment that effectively reinforces this point. A $\{6.4 \times 6.4\}$ m^2 domain, excited by a transverse electric Gaussian pulse of $f_{max} =$ 1 GHz, is discretized in $\{512 \times 512\}$ Yee's cells ($\lambda/24$). A radially propagating circular wavefront can be observed in Fig. 3(a), stemming from a GPU-FDTD simulation. Fig. 3(b) shows the same waveform at the same time, resolved by D_4 S-MRTD in a $\{128 \times 128\}$ mesh. Despite the reduction in the discretization rate from $\lambda/24$ to $\lambda/6$, the wavefront still largely preserves its shape. Relative errors (in the Eucledian norm) compared to a reference FDTD simulation conducted in a $\{1024 \times 1024\}$ mesh and execution times of S-MRTD and FDTD are also reported in Fig. 4, for a total simulation time window of 7.55 ns. These results confirm that the predictions of the S-MRTD dispersion analysis are equally valid for GPU-based versions of the technique.

IV. GPU-Accelerated S-MRTD: Accuracy Results

In this section, the net effect of inordinate floating precision on the overall accuracy of a GPU-accelerated time-domain simulation is considered. A $\{512 \times 512\}$ cell rectangular mesh, discretizing an air-filled cavity, excited

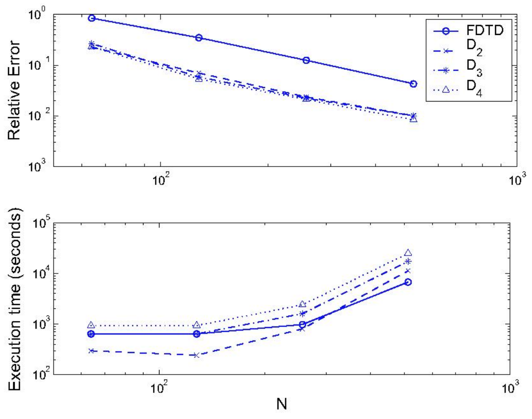


Fig. 4. Relative Errors compared to a reference FDTD simulation ($\{1024 \times 1024\}$ mesh) and execution times of $\{N \times N\}$ mesh FDTD and S-MRTD simulations in a GPU.

by a transverse electric field, with the temporal profile of a 1.0 GHz Gaussian pulse, is simulated with FDTD, D_2, D_3 and D_4, for 16384 time steps, at a time step corresponding to 0.5 of the FDTD stability limit. Fig. 5 includes plots of inaccuracy for FDTD, D_2, D_3, and D_4 schemes as a function of time step. Errors were calculated with respect to a double-precision round-nearest simulation (in the Eucledian norm), every 16 time steps for a total of 1024 data points. In each plot, numerical results for GPU, single-precision truncate-chop, and single-precision, CPU-equivalent, round-nearest simulations are depicted in blue, green, and red respectively. Our results show the GPU exhibiting a high-degree of correlation with, and consistently accruing error at a greater rate than, an equivalent single-precision CPU simulation. However, that error is shown to be very small, largely independent of scheme, and per-time step on the order of an inordinate floating-precision round mode. In particular, that error is closely approximated by the truncate-chop round-mode. This behavior supports the supposition that the GPU inaccuracy is rooted in an inordinate hardware implementation of floating point arithmetic, and will improve with future hardware generations.

V. Conclusions

Unprecedented sustained graphics hardware acceleration rates (of a factor of 30) have been demonstrated for the S-MRTD technique. These rates surpass the achievable FDTD ones by a factor of three, indicating that high-order spatial finite difference methods are optimally suited to the rapidly developing area of general purpose computing on graphics cards. The advantage of S-MRTD over FDTD on GPUs is further enhanced by its superior dispersion behavior, that allows for the use of coarse grids and consequently for a better utilization of the memory available in a graphics card. In addition, the large difference in the achievable speed-ups of S-MRTD and FDTD implies that the drawback of increased operations per cell associated

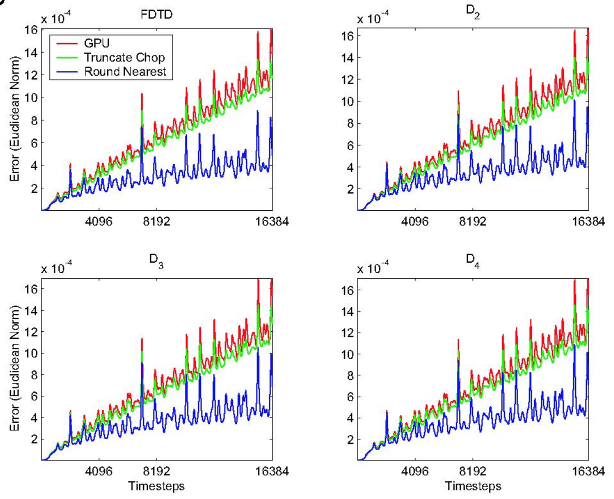


Fig. 5. Accuracy of GPU results with respect to a double precision CPU simulation, compared to truncate-chop and round-nearest CPU simulation results.

with CPU implementations of S-MRTD is effectively eliminated in their GPU counterparts.

VI. Acknowledgment

This work was supported by the Natural Sciences and Engineering Research Council of Canada, through a Discovery Grant.

References

[1] M. Krumpholz, L. P. B. Katehi, "New prospects for time domain analysis," *IEEE Microwave and Guided Wave Lett.*, vol.5, no.11, pp.382-384, Nov. 1995.

[2] M. Krumpholz, L. P. B. Katehi, "MRTD: New Time-Domain Schemes Based on Multiresolution Analysis," *IEEE Trans. Microwave Theory Tech.*, vol.44, no.4, pp.555-561, Apr. 1996.

[3] *General-Purpose Computing Using Graphics Hardware website*, [Online]. Available:www.gpgpu.org.

[4] M. Harris and D. Luebke, *GPGPU: General-Purpose Computing on Graphics Hardware*, SIGGRAPH 2004 Course Notes, Aug. 2004.

[5] S. E. Krakiwsky, L. E. Turner, and M. M. Okoniewski, "Graphics processor unit (GPU) acceleration of finite-difference time-domain (FDTD) algorithm," *Proc. 2004 International Symp. on Circuits and Systems*, vol. 5, pp. V-265 - V-268, May 2004.

[6] S. E. Krakiwsky, L. E. Turner, and M. M. Okoniewski, "Acceleration of Finite-Difference Time-Domain (FDTD) Using Graphics Processor Units (GPU)," *2004 IEEE MTT-S International Microwave Symposium Digest*, vol. 2, pp. 1033 - 1036, Jun. 2004.

[7] G. S. Baron, C. D. Sarris, E. L. Fiume, "Acceleration of Finite-Difference Time-Domain (FDTD) Using Graphics Processor Units (GPU)," *Proc. 2005 IEEE AP-S International Symposium on Antennas Prop.*, Jul. 2005.

[8] R. N. Schneider, M. M. Okoniewski, L. E. Turner, "Custom hardware implementation of the finite-difference time-domain (FDTD) method," *2002 IEEE MTT-S International Microwave Symposium Digest*, vol. 2, pp. 875 - 878, Jun. 2002.

[9] M. Fujii, W. J. R. Hoefer, "Application of biorthogonal interpolating wavelets to the Galerkin scheme of time dependent Maxwell's equations, " *IEEE Microwave Wireless Components Lett.*, vol. 11, no. 1, pp. 22-24, Jan. 2001.

[10] M. Fujii, W.J.R. Hoefer, "Dispersion of time-domain wavelet Galerkin method based on Daubechies compactly supported scaling functions with three and four vanishing moments", *IEEE Microwave Guided Wave Lett.*, vol. 10, no. 4, pp. 125-127, Apr. 2000.