Progress In Electromagnetics Research M, Vol. 16, 185–196, 2011

ANALYSIS OF 3-DIMENSIONAL ELECTROMAGNETIC FIELDS IN DISPERSIVE MEDIA USING CUDA

M. R. Zunoubi

Department of Electrical and Computer Engineering
State University of New York, New Paltz, NY, USA

J. Payne

US Air Force Research Laboratory
Human Effectiveness Directorate (AFRL/HE)
Brooks City-Base, TX, USA

Abstract—This research presents the implementation of the Finite-Difference Time-Domain (FDTD) method for the solution of 3-dimensional electromagnetic problems in dispersive media using Graphics Processor Units (GPUs). By using the newly introduced CUDA technology, we illustrate the efficacy of GPUs in accelerating the FDTD computations by achieving appreciable speedup factors with great ease and at no extra hardware/software cost. We validate our approach by comparing the results with their corresponding simulated results obtained from Remcom's XFDTD software.

1. INTRODUCTION

Due to the growing interest in the fast and efficient analysis of large-scale complex electromagnetic radiation and scattering problems, much research has been devoted to enhancing numerical solutions of such problems in terms of their speed and memory requirements. The Finite Difference Time Domain (FDTD) technique is commonly used for these types of problems thanks to its ease of implementation and a broad range of capabilities. However, for many problems of interest, the computational runtime can be prohibitive. Traditionally, a computer cluster has been used to circumvent this problem by devising

Received 25 November 2010, Accepted 6 January 2011, Scheduled 20 January 2011
Corresponding author: Mohammad R. Zunoubi (zunoubm@engr.newpaltz.edu).

a parallel-FDTD methodology [1], which itself needs a relatively large space, is expensive, requires occasional maintenance, and is shared among various research teams. Alternatively, researchers have recently focused on implementing the FDTD technique on Graphic Processing Units (GPUs), which possess inherent attributes for threaded computing and can be easily integrated into a standalone desktop with minimal extra cost and space.

CUDA (Compute Unified Device Architecture) was recently introduced by NVIDIA to leverage the parallel compute engine in NVIDIA GPUs [2]. As commercial sectors like Remcom, Computer Simulation Technology (CST), Acceleware, SPEAG, and Agilent have integrated the GPU-accelerated FDTD technique into their software packages for the analysis of electromagnetic fields in normal and complex media, researchers in academia have also focused on taking advantage of inherent attributes of the GPUs for parallel computing. Such implementations that use the CUDA technology include the works presented in [3] and [4] originally, and the research presented by Sypek et al. for a TM^z solution of electromagnetic fields [5], the double precision implementation of the FDTD on the Tesla GPU [6], and the work reported by Okoniewski's group [7], more recently.

In this article, we present the implementation of the full 3-dimensional FDTD method in dispersive media on GPUs using CUDA technology. The Convolutional Perfectly Matched Layer (CPML) is used to truncate the solution domain [8], and a plane wave incident field is utilized to simulate realistic exposure scenarios. The software validity is established by making comparisons with simulated results obtained by Remcom's XFDTD software. Speedup factors are reported which show promise for the rapid solution of large-scale electromagnetic radiation and scattering problems.

2. FORMULATION

Here, we present the 3-dimensional FDTD formulation for the plane wave penetration through a general dispersive media. Accordingly, we follow the procedure outlined in [9] for the Piecewise-Linear Recursive-Convolution (PLRC) method for the single-pole Debye representation of the dispersive media along with the CPML 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:

$$E_x\big|_{i+1/2,j,k}^{n+1} = c_a\big|_{i+1/2,j,k}\, E_x\big|_{i+1/2,j,k}^{n} + c_b\big|_{i+1/2,j,k}$$
$$\cdot\left(\frac{H_z\big|_{i+1/2,j+1/2,k}^{n} - H_z\big|_{i+1/2,j-1/2,k}^{n}}{\kappa_{yj}} - \frac{H_y\big|_{i+1/2,j,k+1/2}^{n} - H_z\big|_{i+1/2,j,k-1/2}^{n}}{\kappa_{zk}}\right.$$
$$\left. + \psi_{E_{x,y}}\big|_{i+1/2,j,k}^{n} + \psi_{E_{x,z}}\big|_{i+1/2,j,k}^{n}\right) + c_c\big|_{i+1/2,j,k}\, \psi_{E_x}\big|_{i+1/2,j,k}^{n} \qquad (1)$$

where c_a, c_b, c_c are the coefficients associated with space-time discretization and material properties, κ_{yj} and κ_{zk} are the scaled tensor parameters, $\psi_{E_{x,y}}$ and $\psi_{E_{x,z}}$ are the auxiliary arrays for the CPML, and ψ_{E_x} is the recursive accumulator variable given in [9] and defined in terms of ε_s, ε_∞, and τ which are the static relative permittivity, the relative permittivity at infinite frequency, and relaxation time, respectively, for the Debye material.

To model the tissue material, we generate three *id* files that are used to assign the constitutive parameters to the corresponding field components in the x, y, and z directions, respectively. For *normal* materials, ε_r, μ_r, and σ are assigned and used to calculate the regular FDTD updating coefficients. For *dispersive* materials, the Debye parameters ε_s, ε_∞, and τ are assigned and used to calculate the dispersive FDTD updating coefficients for the electric fields and for the recursive accumulator variable ψ_{Ex}. Note that these accumulators are formed only for the dispersive media by checking the tissue *id* arrays and are used in Equation (1) in which case coefficient c_c is nonzero. Otherwise, accumulator variables are not formed and coefficient c_c is set to zero as well.

3. CUDA IMPLEMENTATION

3.1. Memory Management

The implementation of the 3-dimensional FDTD formulation involves allocating and initializing on the *Device's* (GPU) *global* memory six one-dimensional arrays for the E_x, E_y, E_z and H_x, H_y, H_z field components, three arrays for recursive accumulator variables ψ_{Ex}, ψ_{Ey}, ψ_{Ez}, twenty four auxiliary $\psi_{Ei,j}$ and $\psi_{Hi,j}$ arrays where i,j are interchangeable combinations of x, y, z such as $\psi_{Ex,y}$ and $\psi_{Hx,y}$ for the CPML, and two arrays for the incident electric and magnetic fields. Since the above quantities need to be updated at every time step, they have to be stored in such a way that they can be both read from and be written to while on the *device*. However, once on the *device*, it is advantageous to use the fast but limited *shared* memory as described in the next Section to handle updating equations. Additionally, we need to allocate and assign three integer arrays *tissueIDx*, *tissueIDy*, and *tissueIDz* that contain the indices assigned to the different tissue material under exposure, and 13 arrays for coefficients c_a, c_b, and c_c, scaled tensor parameters κ's, and the constants used in calculating auxiliary ψ-arrays. However, since no updating of these arrays is needed during the time marching process, we use the *texture* memory of the device for storage which allows for fast read-access when updating the field quantities.

3.2. Device Kernels

In order to implement our FDTD approach efficiently on the GPU, the *device* implementation is divided into two *kernels*. One *kernel* is used to update the magnetic field components and the other is used to update the electric fields. We take full advantage of the Single Instruction Multiple Data (SIMD) architecture of the GPU and overcome the memory latency by grouping the threads into $i \times j \times k$ columns of the grid. These are loaded into the *shared* memory of the *device* with one thread assigned to an output element. Since the FDTD updating equations require interaction with the neighboring grids in the form of $i\pm1$ and $j\pm1$, we allow four halo regions to include the neighboring elements in the xy-plane. Due to the limited amount of the *shared* memory, we load only 3 k-planes of data at a time, with an outer loop covering the total number of planes in the z-direction. Within this outer loop, we read in plane $k+1$, calculate, and store new values for plane k [10]. The main advantage here is that the threads within each small block access their own *shared* memory, the latency of which is two orders of magnitude lower than that of the *global* memory. This enables extremely fast FDTD updating on the *device*. After completing all the computations inside the k outer loop, the resulting updated electric or magnetic fields are stored in the *global* memory to be read by the *host*. It should be noted that the thread dimensions i and j of each block loaded into the *shared* memory must be optimized in order to take full advantage of the GPU computational capabilities. This will be addressed in the Results Section.

Following, we present first a pseudo-code for the *host* (CPU) side that manages the *device* memory allocations, copying data from the *host* to *device*, calling GPU kernels, and copying the data after FDTD updating from the *device* to the *host*:

- cutilSafeCall (cudaMalloc((void**)&_d_E_x, grid_size);
- Repeat for other filed components, incident fields, *tissueID*s, and CPML arrays;

```
- cutilsafeCall (cudaMemcpy(d_$E_x$, $E_x$, grid_size, cudaMemcpy-
  HostToDevice);
- Repeat for all the above arrays;
- Bind textures for read-only data as discussed in Section 3.1:
  cudaBindTexture (0, *ttissueIDx*, d_*tissueIDx*);
- Repeat for all other constant arrays;
- For all time steps do
  {
  Update $H$ <Grid of thread blocks, Size of thread block>;
  Syncthread;
  Update $E$ < Grid of thread blocks, Size of thread block>;
  Syncthread;
  }
- cutilsafeCall (cudaMemcpy($E_x$, d_$E_x$, grid_size, cudaMemcpyDe-
  viceToHost);
- Repeat for $Ey$, $E_z$, $H_x$, $H_y$, and $H_z$.
```

We next present a pseudo-code for the *device* side that uses the *texture*, *shared*, and *global* memory hierarchies to perform the FDTD updating equations efficiently:

```
- Load the read-only data in the texture memory:
  texture<int, 1, cudaReadModeElementType>*ttissueIDx*;
- Repeat for all other constant arrays;
- Update $H$ <Grid of thread blocks, Size of thread block>
  {
  Set up indices for halos;
  Set up indices for main block;
  Read initial plane of $E_x$, $E_y$, and $E_z$ arrays into the *shared*
  memory;
  Loop over $k$-planes;
  Move two planes down and read in new plane $k$+1 into the *shared*
  memory;
  Syncthreads;
  Perform FDTD updating equations for $H$;
  Correct for plane wave interface;
  Update $\psi_{Hi,j}$ ($i$,$j$are interchangeable combinations of $x$, $y$, $z$ such
  as $\psi_{Hx,y}$) terms and $H$ fields inside the CPML regions;
  Syncthreads;
  Store $H$ fields into the *global* memory;
```

```
    }
- Update $E$ <Grid of thread blocks, Size of thread block>
    {
    Set up indices for halos;
    Set up indices for main block;
    Read initial plane of $H_x$, $H_y$, and $H_z$ arrays into the *shared* memory;
    Loop over $k$-planes;
    Move two planes down and read in new plane $k+1$ into the *shared* memory;
    Syncthreads;
    Perform regular FDTD updating equations for $E$ in main computational volume;
    Force fields to zero at boundaries;
    Perform PLRC updates for dispersive media by means of checking the $tissueIDx$, $tissueIDy$, and $tissueIDz$ arrays and updating $\psi_{Ex}$, $\psi_{Ey}$, and $\psi_{Ez}$ terms according to Equation (9.17) in [9];
    Correct for plane wave interface;
    Update $\psi_{Ei,j}$ ($i$,$j$ are interchangeable combinations of $x$, $y$, $z$ such as $\psi_{Ex,y}$) terms and $E$ fields inside the CPML regions;
    Syncthreads;
    Store $H$ fields into the *global* memory;
    }
```

The flowchart of the CUDA accelerated FDTD scheme is also seen in Figure 1 which shows the general steps taken in implementing our tool on both the CPU and GPU sides.

3.3. Arithmetic Instructions

The computational efficiency of our kernels is maximized by facilitating the __mul24 and __fdividef functions for integer multiplications and floating-point divisions, respectively. __mul24 performs 24-bit integer multiplication in 4 clock cycles compared to the 16 clock cycles for the conventional 32-bit multiplication and __fdividef provides for single-precision floating-point division in 20 clock cycles which is superior to the 36 clock cycles typically needed for dividing floating point values [2].

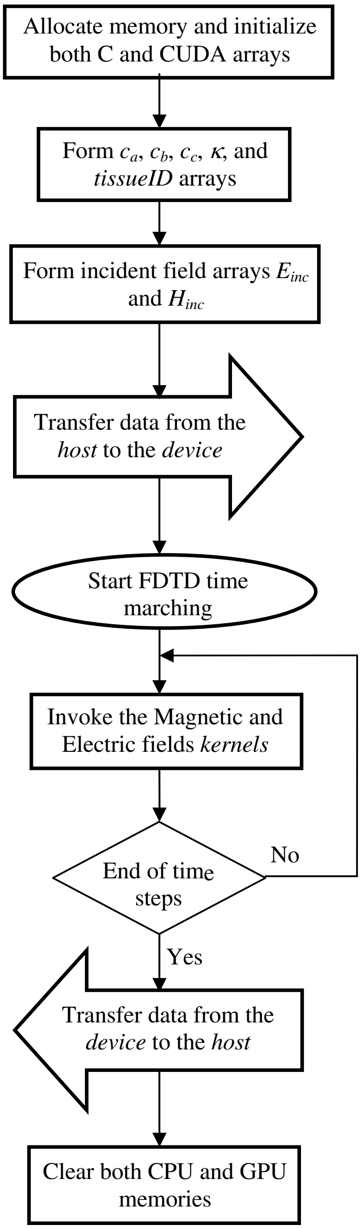
Allocate memory and initialize both C and CUDA arrays
Form c_a, c_b, c_c, κ, and *tissueID* arrays
Form incident field arrays E_{inc} and H_{inc}
Transfer data from the *host* to the *device*
Start FDTD time marching
Invoke the Magnetic and Electric fields *kernels*
End of time steps
No
Yes
Transfer data from the *device* to the *host*
Clear both CPU and GPU memories

Figure 1. Flowchart of the CUDA accelerated FDTD scheme.

4. RESULTS

To demonstrate the accuracy of our developed tool, we simulate a $1 \times 2.5 \times 1$ cm cuvette with a 1 mm thick wall made from Polystyrene ($\varepsilon_r = 2.0$) and filled with water as shown in Figure 2. To model the frequency dependence of water, the Debye parameters of $\varepsilon_S = 81$, $\varepsilon_\infty = 1.8$, $\sigma_S = 0$, and $\tau = 9.4e{-}12$ are used. We choose the cell size of 0.25 mm in all three dimensions resulting in a $\Delta t = 4.81 \times 10^{-13}$ s and we truncate the solution domain by an 8-cell CPML boundary. The cuvette is exposed to a y-polarized Gaussian plane wave propagating in the x direction. The simulation is performed for 4100 time steps and the results of the y-component of the electric field are recorded on the vacuum-Polystyrene boundary and plotted in Figure 3. Examining the results of this figure indicates that the steady-state solution has been reached which is a requirement for performing the fast Fourier transforms on the time domain data. The incident field due to the Gaussian pulse $Ae^{-\alpha(t-\beta\Delta t)^2}$, where $\beta = 32$, $\propto = (\frac{4}{\beta\Delta t})^2$, $A =$ 1000 V/m, is subtracted from the total field E_y to obtain the reflected field due to the material discontinuity. The reflection coefficient versus frequency is then calculated by normalizing the Fourier transform of the reflected field by the Fourier transform of the incident pulse. Results can be seen in Figure 4, where a comparison is made with the results obtained from the Remcom's XFDTD software illustrating the adequacy and accuracy of our FDTD-CUDA implementation.

We then demonstrate the applicability of our tool to calculating the absorbed dose of dispersive media exposed to non-ionizing ultra-

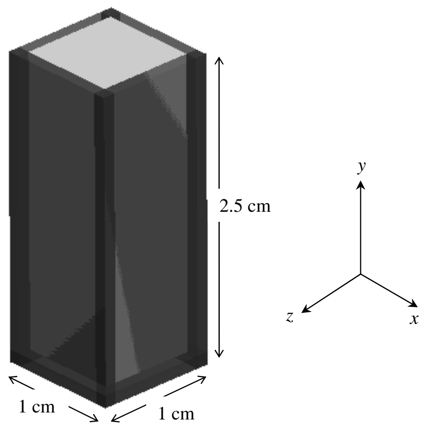

Figure 2. Geometry of exposed polystyrene cuvette filled with water and exposed to a y-polarized electric field traveling in the x-direction.

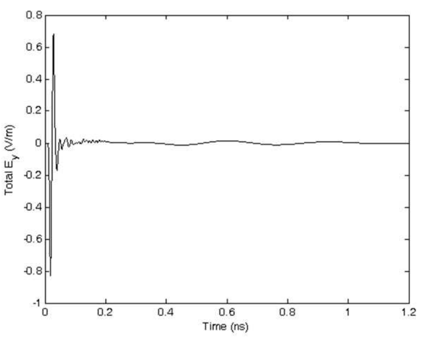

Figure 3. Amplitude of the y-component of the electric field computed in the middle of the cuvette.

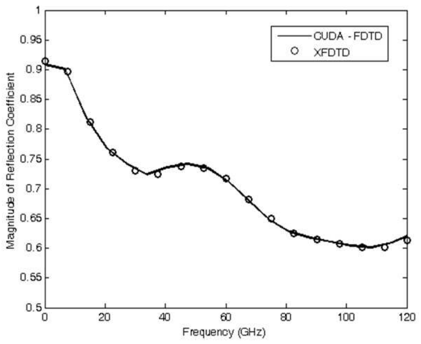

Figure 4. Magnitude of the reflection coefficient computed on the vacuum-polystyrene interface.

wideband (UWB) electromagnetic pulses of nanosecond duration. This dosimetry is critical for establishing dose-response curves for nanosecond electromagnetic pulses. Here, we consider the problem of exposing water inside the cuvette described above to a short pulse described as [11]:

$$\begin{gathered} E_y = E_0(e^{-\alpha t} - e^{-\beta t}), \\ \text{with } E_0 = 18.5\,\text{KV m}^{-1},\ \alpha = 10^8\,\text{s}^{-1},\ \beta = 2.0\times10^{10}\,\text{s}^{-1} \end{gathered} \tag{2}$$

which has a rise time of 150 ps and a width of 10 ns. Since the penetration of the electric field component in the direction of polarization is defined by the rise time and pulse width, we plot both the incident field and the total field inside water in the middle of the cuvette in Figure 5. This plot indicates that the pulse inside water is a superposition of a short pulse, induced during a fast rise time, and recursively longer pulses induced by the slow variation in the incident pulse.

After illustrating the accuracy and adequacy of our CUDA-accelerated FDTD tool, we perform a study of the speedup factors achieved when using GPUs. As discussed in Section 3.2, the dimension of the 2D blocks $i \times j$ plays a critical role in the performance of the graphic processors. Note that dimension k is fixed at three as explained earlier due to the limitation imposed by the amount of the *shared* memory of the *device*. Therefore, a study was performed by varying the dimensions $i$and j following the format of $(i \times j) = (2 \times j)$, $(4{\times}j)$, $(8{\times}j)$, $(16{\times}j)$, $(32{\times}j)$, $(64{\times}j)$, $(128{\times}j)$, and $(256{\times}j)$ while setting $j = 2$, 4, 8, 16, 32, 64, 128, and 256 for an FDTD problem space of

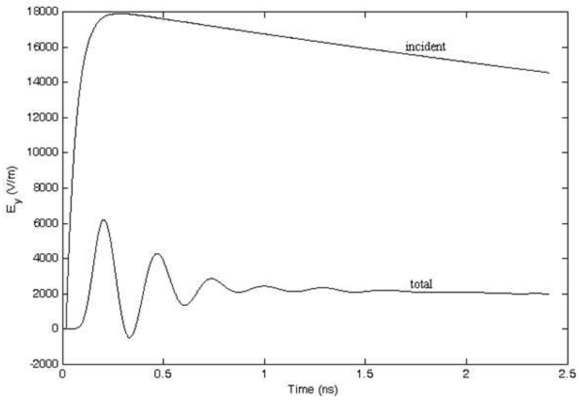

Figure 5. Amplitude of the incident and total electric field computed in the middle of the cuvette.

$128 \times 128 \times 128$ (equivalent to 2,097,152 grids) and timing simulations for 2000 time steps. Our study showed that a block dimension of 32×4 would be optimal with a GPU computational time of 19.94 s, and was therefore used for all further calculations. Also note that a poor choice of the block dimensions could cause a severe performance degradation, as was the case with $i \times j = 8 \times 16$ with a GPU time of 72.73 s.

Once the optimum block size was determined, we simulated three-dimensional FDTD problems that ranged from $64 \times 64 \times 32$ (131,072 FDTD cells) to $256 \times 256 \times 100$ (6, 553, 600 FDTD cells) while keeping $\Delta t = 4.81 \times 10^{-13}$ s and compared the CPU time of the C implementation of our code with its corresponding GPU time using CUDA. Results are recorded in Table 1 for 2000 time steps. The CPU for these simulations was an Intel Core2 Quad Q6600, 2.40 GHz, and 3.25 GB of RAM and the GPU was an NVIDIA GeForce 9800 GT which has 112 Stream Processors, 1024 MB Standard Memory, a Memory Bandwidth of 57.6 GB/s, and a Shader Processing Rate of 504Gigaflops. Note that these timing are for the FDTD time marching process only and do not include the pre- and post-processing calculations and also since we are using one graphic processor, we use only one core out of the Quad processors available on the CPU.

The speedup factors achieved by performing the FDTD computations on the GPU are also given in Table 1. As it can be seen, a speedup factor of 128.5 was obtained for the largest model space possible (6, 553, 600 voxels) before exceeding the 1024 MB *global* memory limit of the 9800GT GPU card. It is expected, however, that with the newly manufactured NVIDIA Tesla C1060 GPUs, the above computations could be carried out for hundreds of millions of FDTD

Table 1. CPU and GPU FDTD computational times and the resulting speedup factors.

Grid points	CPU(s)	GPU(s)	Speedup
131,072	175.45	3.92	44.75
262, 144	402.11	7.27	55.31
819,200	1330.4	13.2	100.6
1,310,720	2135.5	20.9	102.2
2,097,152	3466.1	33.2	104.5
6,553,600	11221.7	87.4	128.5

grid points with even greater computational efficiency. We also need to note that in these simulations, minimum data transfer between the *device* and the *host* was required. However, the above speedup factors could be reduced drastically for applications involving large amount of data transfer as discussed in [12].

5. CONCLUSION

We presented a CUDA-based FDTD analysis of 3-dimensional Maxwell's equations in dispersive media on an NVIDIA 9800GT GPU card. The accuracy of our implementation was illustrated by calculating the reflection coefficient on a vacuum/dielectric interface for a cuvette filled with water and comparing the results with their corresponding results obtained by the Remcom's XFDTD software. Very good agreements were observed. We also presented a very practical problem of exposing dispersive media to the UWB electromagnetic pulses of nanosecond duration. We further investigated the computational advantage of performing the 3-dimensional FDTD computation on GPUs using CUDA by solving for the electromagnetic fields for various problem dimensions. It was shown that speedup factors of up to 128.5 can be achieved by facilitating the memory hierarchy of GPUs and using CUDA's arithmetic instructions. It was shown that *shared* memory usage minimizes the memory latency associated with graphic cards when optimized block dimension is used.

REFERENCES

1. Yu, W., R. Mittra, T. Su, Y. Liu, and X. Yang, *Parallel Finite-Difference Time-Domain Method*, Artech House, July 2006.

2. "NVIDIA CUDA compute unified device architecture programming guide," 3.2, NVIDIA Corporation, Nov. 2010.
3. Balevic, A., L. Rockstroh, A. Tausendfreund, S. Patzelt, G. Goch, and S. Simon, "Accelerating simulations of light scattering based on finite-difference time-domain method with general purpose GPUs," *Proc. IEEE CSE'08, 11th IEEE Int. Conference on Computational Science and Engr.*, 16–18, São Paulo, Brazil, July 2008.
4. Valcarce, A., G. De La Roche, A. Juttner, D. López-Pérez, and J. Zhang, "Applying FDTD to the coverage prediction of WiMAX femtocells," *Eurasip Journal of Wireless Communications and Networking. Special issue: Advances in Propagat. Modeling for Wireless Systems*, Feb. 2009.
5. Sypek, P., A. Dziekonski, and M. Mrozowski, "How to render FDTD computations more effective using a graphics accelerator," *IEEE Trans. Magnetics*, Vol. 45, No. 3, 1324–1327, Mar. 2009.
6. Demir, V., "Performance analysis of CUDA implementation of FDTD on Tesla GPU using double precision arithmetic," *2010 USNC-URSI National Radio Science Meeting*, Boulder, CO, Jan. 6–9, 2010.
7. Ong, C. Y., M. Weldon, S. Quiring, L. Maxwell, M. C. Hughes, C. Whelan, and M. Okoniewski, "Speed it up," *IEEE Microwave Magazine*, Vol. 11, No. 2, Apr. 2010.
8. Roden, A. and S. D. Gedney, "Convolutional PML (CPML): An efficient FDTD implementation of the CFS-PML for arbitrary media," *Microwave and Opt. Tech. Letters*, Vol. 27, 334–339, June 2000.
9. Taflove, A. and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 2nd Edition, Artech House, 2000.
10. Giles, M., "Jacobi iteration for a Laplace discretization on a 3d structured grid," Technical Report, 2008.
11. Simicevic, N., "Three-dimensional FDTD simulation of biomaterial exposure to electromagnetic nanopulses," *Phys. Med. Biol.*, Vol. 50, No. 21, 5041–5053, Nov. 2005.
12. Zunoubi, M. R., J. Payne, and W. P. Roach, "CUDA implementation of TE^z-FDTD solution of Maxwell's equations in dispersive media," *IEEE Antennas Wireless Propagat. Lett.*, Vol. 9, 756–759, Sept. 2010.