High-Order Error-Optimized FDTD Algorithm With GPU Implementation

Theodoros T. Zygiridis

Department of Informatics and Telecommunications Engineering, University of Western Macedonia, Kozani 50100, Greece

This paper presents the development of a two-dimensional (2-D) finite-difference time-domain (FDTD) solver that features reliable calculations and reduced simulation times. The accuracy of computations is guaranteed by specially-designed spatial operators with extended stencils, which are assisted by an optimized version of a high-order leapfrog integrator. Both discretization schemes rely on error-minimization concepts, and a proper least-squares treatment facilitates further control in a wideband sense. Given the parallelization capabilities of explicit FDTD algorithms, considerable speedup compared to serialized CPU calculations is accomplished by implementing the proposed algorithm on a modern graphics processing unit (GPU). As our study shows, the GPU version of our technique reduces computing times by several times, thus confirming its designation as a highly-efficient algorithm.

***Index Terms*—Finite-difference time-domain methods, graphics processing unit (GPU) computing, high-order algorithms, numerical dispersion.**

I. Introduction

FOR RELIABLE and rapid large-scale electromagnetic simulations at reasonable computational cost, one is expected to exploit advanced numerical methods, as well as the computing capacity of contemporary platforms. In the context of finite-difference time-domain (FDTD) algorithms, high-order approximations [1] are known to be more efficient than the second-order scheme [2], albeit requiring more operations per time step. If error-minimization concepts are applied, further improvement is feasible, leading to optimized models [3]–[5].

Another aspect to be considered is the implementation of computational methods on graphics processing units (GPUs), which is currently explored by many researchers. Featuring a large number of processing cores, modern GPUs offer augmented—yet low-cost—computing power for parallel applications, facilitated by easy-to-use programming languages [6]. The merits for FDTD algorithms are demonstrated in cases including examples of biomedical applications [7], dispersive media [8], and plasmonics [9]. The potential of GPU computing for electromagnetic simulations has been also recognized by developers of various commercial software packages (e.g., CST, XFDTD, SEMCAD, and others), whose current capabilities include the option for parallel program execution on GPUs. It is noted that many-core architectures are now being integrated to designs other than GPU-oriented ones (e.g., Xeon Phi), aiming at high performance via massive parallelization.

Based on the aforementioned principles, we present a two-dimensional (2-D) high-order FDTD method in optimized formulation. Aiming at the control of numerical flaws, we separately introduce error-reducing procedures for space and time discretizations. Spatial derivatives are approximated by modified finite-difference expressions with extended stencils, while the fourth-order leapfrog integrator [10] is re-formulated accordingly. By adding an extra step to the process, performance can be tuned over desired frequency bands by the least-squares technique. We are also interested in minimizing the computing times, especially in large problems. Hence, we exploit the scheme's parallelization potential and implement it on a GPU, achieving significant acceleration compared to serialized executions. Special features, such as data re-use due to the stencil size, and other practical issues are addressed. The results presented herein verify the valid design of the algorithm, and demonstrate the efficiency of the GPU implementation.

Manuscript received November 09, 2012; revised January 01, 2013; accepted January 11, 2013. Date of current version May 07, 2013. Corresponding author: T. T. Zygiridis (e-mail: tzygiridis@uowm.gr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMAG.2013.2241410

II. Methodology

Generalizing the single-frequency approach of [3], spatial parametric operators spanning $2N$ points along, e.g., x-axis

$$\mathrm{D}_x u|_{i,j} = \frac{1}{\Delta x}\sum_{m=1}^{N} C_m^{(N)}\left(u|_{i+\frac{m}{2},j} - u|_{i-\frac{m}{2},j}\right) \tag{1}$$

can be optimized by minimizing the function

$$E_X = \int_0^{2\pi}\left[2\sum_{m=1}^{N} C_m^{(N)}\sin\left(\frac{(2m-1)\pi}{N_x}\cos\phi\right) - \frac{2\pi}{N_x}\cos\phi\right]^2 d\phi \tag{2}$$

where $N_x = \lambda/\Delta x$ is the grid density along x-axis, and ϕ is the polar angle denoting the direction of wave propagation. The definition in (2) is derived by considering the test function

$$u = \exp\{\mathrm{j}[\omega t - k(x\cos\phi + y\sin\phi)]\} \tag{3}$$

and integrating the square of the pertinent error over all angles (j is the imaginary unit, $k=\omega/c_0$ is the wavenumber). An expression E_Y similar to (2) is introduced for the operator D_y. To ensure error minimization and find the optimum coefficients, the critical points of E_X are detected from $\partial E_X/\partial C_m^{(N)} = 0$. N equations are thus deduced, expressed in matrix form as $[\mathbf{A}(N_x)][\mathbf{c}] = [\mathbf{b}(N_x)]$, where $[\mathbf{c}]$ is the coefficient vector. For example, in the case $N=3$, one obtains $[\mathbf{A}(N_x)] = [a_{ij}]_{3\times 3}$, $[\mathbf{c}] = [C_1^{(3)}\, C_2^{(3)}\, C_3^{(3)}]^{\mathrm{T}}$, and $[\mathbf{b}(N_x)] = [b_i]_{3\times 1}$, where

$$a_{ii} = 1 - J_0\left((2i-1)\frac{2\pi}{N_x}\right) \tag{4}$$

$$a_{ij} = J_0\left(|i-j|\frac{2\pi}{N_x}\right) - J_0\left((i+j-1)\frac{2\pi}{N_x}\right), \quad i\neq j \tag{5}$$

$$b_i = \frac{2\pi}{N_x}J_1\left((2i-1)\frac{\pi}{N_x}\right) \tag{6}$$

$i,j=1,2,3$, and $J_{0,1}$ are Bessel functions of the first kind. The first terms of the corresponding Maclaurin series verify that the new coefficients are "corrections" of the values used for sixth-order formal accuracy, as we find

$$C_1^{(3)} \simeq \frac{75}{64}+\frac{25}{512}\left(\frac{\pi}{N_x}\right)^2 \tag{7}$$

$$C_2^{(3)} \simeq -\frac{25}{384}-\frac{25}{1024}\left(\frac{\pi}{N_x}\right)^2 \tag{8}$$

$$C_3^{(3)} \simeq \frac{3}{640}+\frac{5}{1024}\left(\frac{\pi}{N_x}\right)^2. \tag{9}$$

The procedure presented thus far requires the selection of an optimization frequency (equivalently, a unique N_x). If necessary (e.g., in more broadband simulations), one may calculate the unknown coefficients by treating the following system:

$$\begin{bmatrix}[\mathbf{A}(N_1)]\\ \vdots \\ [\mathbf{A}(N_\kappa)]\end{bmatrix}[\mathbf{c}]=\begin{bmatrix}[\mathbf{b}(N_1)]\\ \vdots \\ [\mathbf{b}(N_\kappa)]\end{bmatrix} \tag{10}$$

in a least-squares sense. This additional step enables error control over wider frequency bands, provided that the mesh densities $N_i, i=1,\ldots,\kappa$ span the range of preferred wavelengths.

Similar concepts can be applied to high-order integrators. Recall the origin of the fourth-order leapfrog formula

$$\left.\frac{\partial u}{\partial t}\right|^n=\frac{u|^{n+\frac{1}{2}}-u|^{n-\frac{1}{2}}}{\Delta t}-\frac{\Delta t^2}{24}\left.\frac{\partial^3 u}{\partial t^3}\right|^n+O(\Delta t^4) \tag{11}$$

where the derivative $\partial^3 u/\partial t^3$ is approximated by spatial ones. For instance, the proposed temporal operator for E_x introduces two degrees of freedom K_1, K_2, and is formulated as follows:

$$\begin{aligned}\mathrm{D}_t E_x|^{n+\frac{1}{2}}=&\frac{E_x|^{n+1}-E_x|^n}{\Delta t}\\&-\frac{(c_0\Delta t)^2}{24\epsilon}(K_1\mathrm{D}_{yyy}+K_2\mathrm{D}_{yxx})H_z|^{n+\frac{1}{2}}\end{aligned} \tag{12}$$

where D_{yyy} and D_{yxx} are standard second-order mixed-derivative expressions shown in the equation at the bottom of the page. Setting $K_1=K_2=1$ recovers fourth-order accuracy. Considering the test function (3), as well as the fact that the plane-wave assumption dictates $E_x=-\eta H_z\sin\phi$ (η is the medium's intrinsic impedance), the temporal error estimator can be defined as

$$\begin{aligned}E_T=&\left[\frac{1}{Q}\sin\left(\frac{\pi Q}{N_x}\right)-\frac{\pi}{N_x}\right]R\sin\phi\\&-\frac{1}{24}\left(\frac{Q}{R}\right)^2\left\{K_1\left[\sin\left(\frac{3\pi R}{N_x}\sin\phi\right)\right.\right.\\&\left.-3\sin\left(\frac{\pi R}{N_x}\sin\phi\right)\right]\\&\left.+2K_2R^2\left[\cos\left(\frac{2\pi}{N_x}\cos\phi\right)-1\right]\sin\left(\frac{\pi R}{N_x}\sin\phi\right)\right\}\end{aligned} \tag{13}$$

where $Q=c_0\Delta t/\Delta x$ and $R=\Delta y/\Delta x$. Naturally, time integration is now influenced not only by dispersion, but also by anisotropy errors, due to the appearance of ϕ-dependent terms. For performance optimization, K_1 and K_2 are chosen to cancel dominant terms in the error's trigonometric series representation. This yields a system $[d_{ij}]_{2\times 2}[\mathbf{k}]=[e_i]_{2\times 1}$, where

$$\begin{aligned}d_{i,1}=&\frac{1}{12}\left(\frac{Q}{R}\right)^2\left[J_{2i-1}\left(\frac{3\pi R}{N_x}\right)\right.\\&\left.-3J_{2i-1}\left(\frac{\pi R}{N_x}\right)\right]\\d_{i,2}=&\frac{Q^2}{6}\left[J_{2i-1}\left(\sqrt{R^2+4}\frac{\pi}{N_x}\right)\right.\\&\left.\times\cos[(2i-1)\phi_0]-J_{2i-1}\left(\frac{\pi R}{N_x}\right)\right]\\e_1=&\frac{R}{Q}\sin\left(\frac{\pi Q}{N_x}\right)-\frac{\pi R}{N_x}\\e_2=&0.\end{aligned}$$

$[\mathbf{k}]=[K_1K_2]^T$, $\phi_0=\arctan(2/R)$ and $i=1,2$. As previously, least squares can be beneficial for the integrator's attributes.

Fig. 1 is indicative of the scheme's efficiency and performance (case $N=2$ is shown), where the phase-velocity error

$$e_c=\frac{1}{2\pi}\int_0^{2\pi}\left|1-\frac{\tilde{c}}{c_0}\right|d\phi \tag{14}$$

versus mesh density is illustrated for various optimization conditions ($\tilde{c}$ is the numerical phase speed). For comparison, the curves for Yee's method and the standard fourth-order approach are also given. Unlike commonly-adopted schemes, the proposed one remedies errors at frequencies around the optimization point, while remaining sufficiently wideband. The least-squares treatment slightly alters the respective curves, and results in more uniform error distributions. Note that the stability limit of the algorithm with $N=2$ is almost the same as that of the standard fourth-order approach.

The presented concepts can be applied to the design of optimized algorithms for three-dimensional (3-D) problems. The spatial operators should minimize an error formula similar to (2), now based on double integration (over θ and ϕ), after the test function (3) is replaced by its 3-D counterpart. The derivation of the time integrator relies on the optimization of mixed

$$\mathrm{D}_{yyy}H_z|^{n+\frac{1}{2}}_{i+\frac{1}{2},j}=\frac{H_z|^{n+\frac{1}{2}}_{i+\frac{1}{2},j+\frac{3}{2}}-H_z|^{n+\frac{1}{2}}_{i+\frac{1}{2},j-\frac{3}{2}}-3\left(H_z|^{n+\frac{1}{2}}_{i+\frac{1}{2},j+\frac{1}{2}}-H_z|^{n+\frac{1}{2}}_{i+\frac{1}{2},j-\frac{1}{2}}\right)}{\Delta y^3}$$

$$\begin{aligned}D_{yxx}H_z|^{n+\frac{1}{2}}_{i+\frac{1}{2},j}=&\frac{H_z|^{n+\frac{1}{2}}_{i+\frac{3}{2},j+\frac{1}{2}}-H_z|^{n+\frac{1}{2}}_{i+\frac{3}{2},j-\frac{1}{2}}+H_z|^{n+\frac{1}{2}}_{i-\frac{1}{2},j+\frac{1}{2}}-H_z|^{n+\frac{1}{2}}_{i-\frac{1}{2},j-\frac{1}{2}}}{\Delta x^2\Delta y}\\&-\frac{2}{\Delta x^2\Delta y}\left(H_z|^{n+\frac{1}{2}}_{i+\frac{1}{2},j+\frac{1}{2}}-H_z|^{n+\frac{1}{2}}_{i+\frac{1}{2},j-\frac{1}{2}}\right)\end{aligned}$$

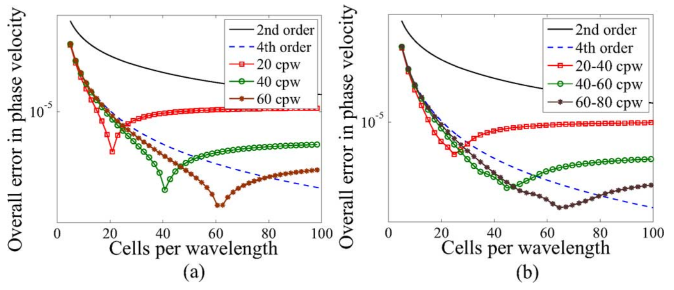


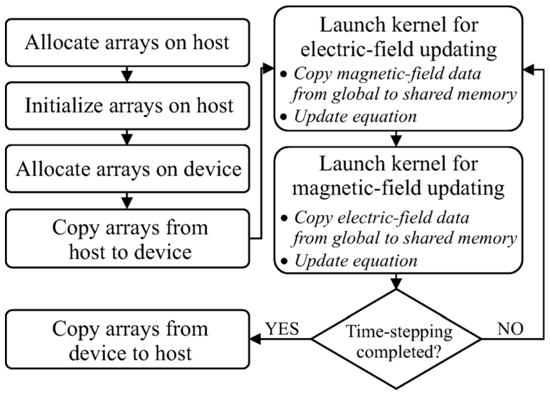
Fig. 1. Overall error in numerical phase velocity versus discretization density (in cells per wavelength—cpw) for different optimizations: (a) single-frequency design, (b) wideband optimization with least-squares treatment.

derivatives, based on proper expansions of error expressions in terms of spherical harmonic functions (e.g., see [11]).

III. GPU Implementation

Modern GPUs are nowadays programmed easily, due to the development of Compute Unified Device Architecture (CUDA). Enriched with the necessary extensions, CUDA is a programming model based on standard languages that facilitates efficient and inexpensive acceleration of scientific applications. Explicit FDTD methods are naturally suited for parallel implementation, because the update of any electric-field component requires only already available magnetic-field values, and vice versa. Pursuing the minimization of computing times, a parallel code has been developed, and executed on an NVIDIA Tesla C1060 card. The latter features 4 GB of global memory and 240 streaming processor cores. As we consider the 2-D TE case, updating is based on two kernels, one for E_x, E_y and one for H_z. It has been found that a grouping with a 16×8 threads-per-block pattern ensures good performance. Practically, each launched thread is assigned to perform the update of a single field variable. A typical flow chart describing the GPU code implementation is given in Fig. 2. As shown, data pertinent to initial values and frequently-used constants are copied from the host (CPU) to the device (GPU) memory, before the time stepping has begun. Then, we execute a loop incorporating the aforementioned kernels. When time stepping is finished, the required data are copied back to the host memory. Note that this flow chart does not entail data movement between host and device during the iterative process.

A known issue is that GPU performance may be hindered by repeated accesses to the card's global memory, due to high latency. To alleviate this problem, calculations within each block exploit the—small, but fast—on-chip shared memory (16 kB for each block of threads), thus reducing the number of required reads from the—larger, yet slower—global memory. In fact, shared memory can be considered a type of explicitly-managed cache. Loading variables directly from global memory ignores the data sharing among neighboring nodes, which is intense due to the extended stencils. The internal region of each block is easily loaded, as there is a one-to-one correspondence between threads and field values. In addition, we must include extra



Allocate arrays on host
Initialize arrays on host
Allocate arrays on device
Copy arrays from host to device
Launch kernel for electric-field updating
• Copy magnetic-field data from global to shared memory
• Update equation
Launch kernel for magnetic-field updating
• Copy electric-field data from global to shared memory
• Update equation
Time-stepping completed?
YES
NO
Copy arrays from device to host

Fig. 2. Flow-chart describing the implementation of the algorithm on a GPU.

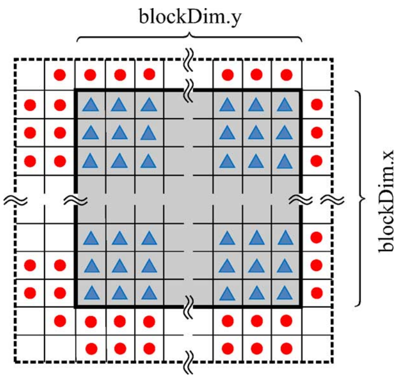


Fig. 3. Pattern indicating the H_z values loaded into shared memory by the threads of a single block. The triangles correspond to internal-region values, each one loaded by the corresponding thread. The circles indicate additional values that need to be loaded, to account for the extended stencil.

nodes around that internal region, due to the extended operators. Hence, some threads are responsible for loading more than one component to shared memory. For example, Fig. 3 depicts the H_z components that should reside in the shared memory, in order to update the electric field within the block with dimensions blockDim.x × blockDim.y. Clearly, a larger matrix compared to the block size is required (in this case, its dimensions must be (blockDim.x+3) × (blockDim.y+3)). It should be noted that special attention is also given to blocks attached to the boundaries of the computational domain, as they do not need all the additional nodes.

IV. Numerical Results

We examine the electromagnetic field of a 5 cm × 5 cm cavity's mode at 45.76 GHz. Specifically, the $\mathrm{TE}_{13,8}$ mode is excited by setting the initial conditions according to the exact solution. Performance is evaluated via the L_2 error norm, shown in (15) at the bottom of the page, for various meshes (Nx, Ny are the number of cells along each direction). Spatial density ranges from 16.11 to 133.51 cells per wavelength. Furthermore, simulations with single- and double-precision calculations are performed. Our goal is twofold: to verify the

$$L_2|^{n+1/2} = \sqrt{\frac{\sum_{i=0}^{Nx-1}\sum_{j=0}^{Ny-1}\left(H_z|_{i+1/2,j+1/2}^{n+1/2} - H_{z,\mathrm{exact}}|_{i+1/2,j+1/2}^{n+1/2}\right)^2}{NxNy}} \tag{15}$$

TABLE I
PERFORMANCE COMPARISON FOR VARIOUS GRIDS (DATA SHOWN FOR CALCULATIONS WITH DOUBLE/SINGLE PRECISION)

Problem Size	**CPU time (s)**	**GPU time (s)**	**Speedup (×)**
128×128	2.58 / 2.57	0.14 / 0.09	18.4 / 28.6
256×256	21.32 / 21.43	0.86 / 0.45	24.5 / 47.6
512×512	176.3 / 175.35	5.73 / 2.6	30.8 / 67.4
1024×1024	1595.74 / 1464.68	43.14 / 19.03	37.0 / 77.0

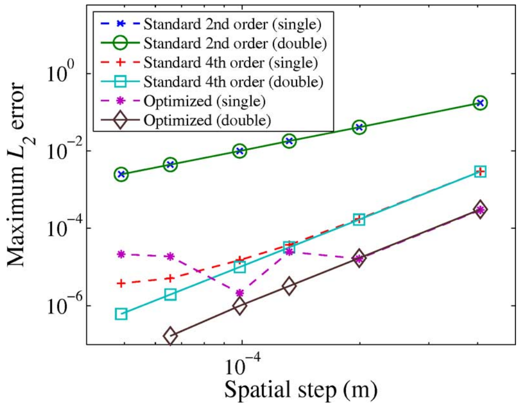


Fig. 4. Maximum L_2 errors versus spatial step for standard and optimized FDTD algorithms, executed on the GPU with single and standard precision.

accuracy of the new algorithm, and quantify the time suppression our GPU implementation succeeds.

To gain insight about the GPU implementation, Table I presents the execution times for single- and double-precision computations (2000 iterations are performed in the case of the coarsest mesh; denser ones are treated accordingly). The presented CPU data correspond to serialized, compiler-optimized code, executed on an i7-950@3.06 GHz processor. Evidently, there can be significant gain due to massive parallelization. For example, a 77-time speedup is accomplished in the case of the 1024×1024 grid for single-precision simulations. This acceleration is less pronounced if double precision is used (practically, a twofold deterioration is then observed). The reason is that the C1060 card features only one double-precision unit per multiprocessor, in contrast to eight single-precision units present (integrating more double-precision units in newer cards may rectify this problem). Nevertheless, non-trivial time reduction by up to 37 times is achieved with double precision.

Algorithmic reliability is evident in Fig. 4, where errors versus simulations' times are plotted. As seen, our method performs better than the standard fourth-order scheme, by reducing errors by 10 times. In addition, the new technique retains fourth-order convergence, even though the formal order of the operators has been sacrificed. These observations are based on calculations with double-precision variables. On the other hand, rounding errors due to the single-precision standard may limit the achievable accuracy level. Hence, to fully exploit optimized algorithms in dense meshes, one should pursue double-precision implementations. We also notice an irregular behavior of our technique in single-precision tests, which should be attributed to the dependence of the performance on the accurate representation of the operator coefficients.

To further quantify the efficiency of GPU implementations, maximum errors as a function of the computational times are given in Fig. 5. Again, we consider the performance for single- and double-precision calculations. One can conclude that single-precision simulations are more efficient than

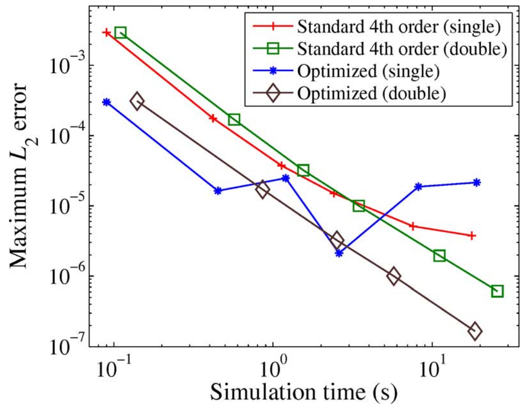


Fig. 5. Maximum L_2 errors versus simulation time for the standard fourth-order and optimized FDTD schemes. Simulations are executed on the GPU considering single- and double-precision accuracy.

double-precision ones, provided that single-precision errors are less significant than dispersion flaws (e.g., in coarser grids). Moreover, even when the proposed scheme exploits the double-precision standard, its GPU performance can be more efficient than that of the fourth-order algorithm with single precision.

V. CONCLUSION

An FDTD algorithm with dispersion-relation-preserving properties has been developed in this paper. We have shown that it attains low error levels, even when compared to existing high-order schemes with similar computational complexity. Furthermore, its GPU implementation enables reduction of computing times. Hence, it can be considered a reliable and efficient means of simulating wave-matter interactions.

REFERENCES

[1] E. Turkel, "High-order methods," in *Advances in Computational Electromagnetics: The Finite-Difference Time-Domain Method*, A. Taflove, Ed. Norwood, MA, USA: Artech House, 1998, ch. 2, pp. 63–110.

[2] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Trans. Antennas Propag.*, vol. AP-14, no. 3, pp. 302–307, May 1966.

[3] T. T. Zygiridis and T. D. Tsiboukis, "Low-dispersion algorithms based on the higher-order (2,4) FDTD method," *IEEE Trans. Microw. Theory Tech.*, vol. 52, no. 4, pp. 1321–1327, Apr. 2004.

[4] M. F. Hadi, "A finite volumes-based 3-D low dispersion FDTD algorithm," *IEEE Trans. Antennas Propag.*, vol. 55, no. 8, pp. 2287–2293, Aug. 2007.

[5] B. Finkelstein and R. Kastner, "The spectral order of accuracy: A new unified tool in the design methodology of excitation-adaptive wave equation FDTD schemes," *J. Comp. Phys.*, vol. 228, no. 24, pp. 8958–8984, 2009.

[6] D. D. Donno, A. Esposito, L. Tarricone, and L. Catarinucci, "Introduction to GPU computing and CUDA programming: A case study on FDTD," *IEEE Antennas Propag. Mag.*, vol. 52, no. 3, pp. 116–122, Jun. 2010.

[7] J. Chi, F. Liu, E. Weber, Y. Li, and S. Crozier, "GPU-accelerated FDTD modeling of radio-frequency field-tissue interactions in high-field MRI," *IEEE Trans. Biomed. Eng.*, vol. 58, no. 6, pp. 1789–1796, Jun. 2011.

[8] M. R. Zunoubi, J. Payne, and W. P. Roach, "CUDA implementation of TE^z-FDTD solution of Maxwell's equations in dispersive media," *IEEE Antennas Wireless Propag. Lett.*, vol. 9, pp. 756–759, 2010.

[9] K. H. Lee, I. Ahmed, R. S. M. Goh, E. H. Khoo, E. P. Li, and T. G. G. Hung, "Implementation of the FDTD method based on Lorentz-Drude dispersive model on GPU for plasmonics applications," *Progr. Electromagn. Res.*, vol. 116, pp. 441–456, 2011.

[10] H. Spachmann, R. Schuhmann, and T. Weiland, "Higher order time integration schemes for Maxwell's equations," *Int. J. Numer. Model.*, vol. 15, no. 5–6, pp. 419–437, 2002.

[11] T. T. Zygiridis and T. D. Tsiboukis, "Optimized three-dimensional FDTD discretizations of Maxwell's equations on Cartesian grids," *J. Comput. Phys.*, vol. 226, no. 2, pp. 2372–2388, Oct. 2007.