Progress In Electromagnetics Research, Vol. 135, 297–316, 2013

IMPLEMENTATION OF FDTD-COMPATIBLE GREEN'S FUNCTION ON HETEROGENEOUS CPU-GPU PARALLEL PROCESSING SYSTEM

Tomasz P. Stefanski*

Department of Microwave and Antenna Engineering, Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Gdansk 80-233, Poland

Abstract—This paper presents an implementation of the FDTD-compatible Green's function on a heterogeneous parallel processing system. The developed implementation simultaneously utilizes computational power of the central processing unit (CPU) and the graphics processing unit (GPU) to the computational tasks best suited for each architecture. Recently, closed-form expression for this discrete Green's function (DGF) was derived, which facilitates its applications in the FDTD simulations of radiation and scattering problems. Unfortunately, implementation of the new DGF formula in software requires a multiple precision arithmetic and may cause long runtimes. Therefore, an acceleration of the DGF computations on a CPU-GPU heterogeneous parallel processing system was developed using the multiple precision arithmetic and the OpenMP and CUDA parallel programming interfaces. The method avoids drawbacks of the CPU- and GPU-only accelerated implementations of the DGF, i.e., long runtime on the CPU and significant overhead of the GPU initialization respectively for long and short length of the DGF waveform. As a result, the sevenfold speedup was obtained relative to the reference DGF implementation on a multicore CPU thus applicability of the DGF in FDTD simulations was significantly improved.

1. INTRODUCTION

The hybridization between the finite-difference time-domain (FDTD) method and integral equation based numerical methods requires consistency with the discrete electromagnetism theory [1–5] and the

Received 17 November 2012, Accepted 12 December 2012, Scheduled 24 December 2012
* Corresponding author: Tomasz P. Stefanski (tomasz.stefanski@pg.gda.pl).

discrete Green's function (DGF) that is compatible with the Yee's grid [6–12]. The DGF is directly derived from the FDTD update equations thus the FDTD method and its integral discrete formulation based on the DGF can be perfectly coupled [11]. Applications include electromagnetic simulations on disjoint domains [13] and construction of the boundary conditions, i.e., absorbing (ABC) [7, 8, 12] as well as truncating the grid in the presence of reflecting external media [14]. Computational efficiency of the FDTD method can therefore be improved due to elimination of free-space cells between objects and perfectly matched layers at the grid boundaries. The DGF can also be applied to the total-field/scattered-field formulation of the FDTD method to reduce erroneous field leakage across the boundary [15].

Due to the above-mentioned advantages of the DGF application, this function has been found to be an efficient tool facilitating the design process of antennas [10–12]. The FDTD efficiency in such simulations is mostly affected by the number of cells occupied by the vacuum in the region between an antenna and an observation point. In [10, 11], the DGF antenna simulations were demonstrated based on wire-antenna examples showing the savings in runtime and memory usage. In [12], the DGF approach to FDTD simulations was demonstrated in the design process of the ultra-wideband cylindrical antenna. This technique employs the DGF as the ABC placed close to the antenna surface and as the radiated field propagator to observation points at each FDTD iteration step. Antenna shape was optimized in order to achieve the desired radiated field characteristics, i.e., the required temporal pulse parameters at certain observation points. The DGF was computed at the preprocessing stage and it was combined with many different FDTD simulations in the optimization process of the antenna.

Recently, Jeng [9] derived the new analytic closed-form expression for the 3-D dyadic DGF in infinite free space based on the ordinary z-transform along with the partial difference operators. Although the new DGF expression has a very compact form, it involves binomial coefficients that may cause long runtimes and accuracy problems for large values of upper indices. In spite of that, the accelerated implementations of the DGF were developed on a multicore central processing unit (CPU) [16] and a graphics processing unit (GPU) [17] in a multiple precision arithmetic. However, it was found that the performance of the GPU-only implementation is deteriorated by overhead of the GPU initialization for short DGF waveforms. On the other hand, the CPU-only implementation requires longer computing runtimes than the GPU-only implementation for long DGF waveforms. It suggests that optimal approach to the DGF parallelization should

simultaneously employ the CPU and the GPU to the lengths of the DGF waveform best suited for each architecture.

Therefore, the objective of this paper is to demonstrate for the first time a new technique for acceleration of the DGF computations on the CPU-GPU heterogeneous parallel processing system. The acceleration of simulations on modern computing architectures represents an emerging trend in computational electromagnetics. In particular, the GPU acceleration has recently become a hot topic of investigations due to its low cost in comparison to the high-performance computing on clusters. Several numerical methods have already been implemented on GPUs, i.e., FDTD method [18–23], finite-difference frequency-domain method [24, 25], finite element method [26–29] and, method of moments [30–32]. Moreover, efforts have been made towards the acceleration of computations on heterogeneous parallel processing systems utilizing computational power of the CPU and the GPU [27, 33]. However, up to now, the acceleration of scientific computations requiring the multiple precision arithmetic has not yet been presented on the CPU-GPU heterogeneous parallel processing system to the best of the author's knowledge.

This paper is organized as follows. The idea behind the DGF formulation of the FDTD method is introduced in Section 2. In Section 3, the DGF implementations are presented for: (i) the multicore CPU, (ii) the GPU, and (iii) the hybrid CPU-GPU parallel processing system. In Section 4, results of numerical benchmarks are presented. Finally, the conclusions are given in Section 5.

2. THE 3-D DISCRETE GREEN'S FUNCTION FORMULATION OF THE FDTD METHOD

Linear and invariant systems such as FDTD equations can be represented by means of the convolution of the input sequence and the response of the system to the Kronecker delta-pulse source [6]:

$$\begin{bmatrix} \mathbf{E}\,|_{ijk}^{n} \\ \eta\mathbf{H}\,|_{ijk}^{n} \end{bmatrix} = \sum_{n'i'j'k'} \begin{bmatrix} \mathbf{G_{ee}}\,|_{i-i'j-j'k-k'}^{n-n'} & \mathbf{G_{eh}}\,|_{i-i'j-j'k-k'}^{n-n'} \\ \mathbf{G_{he}}\,|_{i-i'j-j'k-k'}^{n-n'} & \mathbf{G_{hh}}\,|_{i-i'j-j'k-k'}^{n-n'} \end{bmatrix} \begin{bmatrix} c\Delta t\eta\mathbf{J}\,|_{i'j'k'}^{n'} \\ c\Delta t\mathbf{M}\,|_{i'j'k'}^{n'} \end{bmatrix}, \quad (1)$$

where c denotes the speed of light, η the intrinsic impedance of free space, n the time index, and Δt the time-step size. Equation (1) is referred to as the convolution formulation of the FDTD method [11]. If the length of the DGF waveforms is equal to the number of time steps in the FDTD simulation, this formulation returns the same results as the FDTD method (assuming infinite numerical precision of computations).

The analytic closed-form expression for the DGF in infinite free space for the (i, j, k) Yee's cell can be derived using the method of Jeng [9]. Only the electric field $\mathbf{G_{ee}}$ DGF component is presented here for the sake of brevity:

$$G_{ee,xz} \mid_{ijk}^{n} = \sum_{m=\alpha_x+\beta_x+\gamma_x}^{n-2} \binom{n+m}{2m+2} g_{xz} \mid_{ijk}^{m}, \tag{2}$$

$$G_{ee,yz} \mid_{ijk}^{n} = \sum_{m=\alpha_y+\beta_y+\gamma_y}^{n-2} \binom{n+m}{2m+2} g_{yz} \mid_{ijk}^{m}, \tag{3}$$

$$\begin{aligned} G_{ee,zz} \mid_{ijk}^{n} &= -s_x s_y s_z U \mid^{n-1} \delta \mid_{ijk} \\ &+ \sum_{m=\max(\alpha_{f,y}+\beta_{f,y}+\gamma_{f,y}-1,0)}^{n-2} \binom{n+m}{2m+2} f_{zz} \mid_{ijk}^{m+1} \\ &+ \sum_{m=\alpha_{h,y}+\beta_{h,y}+\gamma_{h,y}}^{n-2} \binom{n+m}{2m+2} h_{zz} \mid_{ijk}^{m}, \end{aligned} \tag{4}$$

where:

$$\begin{aligned} g_{xz} \mid_{ijk}^{m} &= -(-1)^{m+i+j+k} \sum_{\substack{\alpha+\beta+\gamma=m \\ \alpha\geqslant\alpha_x,\beta\geqslant\beta_x,\gamma\geqslant\gamma_x}} \binom{m}{\alpha\beta\gamma} \\ &\times \binom{2\alpha+1}{\alpha+i+1} \binom{2\beta}{\beta+j} \binom{2\gamma+1}{\gamma+k} s_x^{2\alpha+2} s_y^{2\beta+1} s_z^{2\gamma+2}, \end{aligned} \tag{5}$$

$$\begin{aligned} g_{yz} \mid_{ijk}^{m} &= -(-1)^{m+i+j+k} \sum_{\substack{\alpha+\beta+\gamma=m \\ \alpha\geqslant\alpha_y,\beta\geqslant\beta_y,\gamma\geqslant\gamma_y}} \binom{m}{\alpha\beta\gamma} \\ &\times \binom{2\alpha}{\alpha+i} \binom{2\beta+1}{\beta+j+1} \binom{2\gamma+1}{\gamma+k} s_x^{2\alpha+1} s_y^{2\beta+2} s_z^{2\gamma+2}, \end{aligned} \tag{6}$$

$$\begin{aligned} f_{zz} \mid_{ijk}^{m} &= -(-1)^{m+i+j+k} \sum_{\substack{\alpha+\beta+\gamma=m \\ \alpha\geqslant\alpha_{f,z},\beta\geqslant\beta_{f,z},\gamma\geqslant\gamma_{f,z}}} \binom{m}{\alpha\beta\gamma} \\ &\times \binom{2\alpha}{\alpha+i} \binom{2\beta}{\beta+j} \binom{2\gamma}{\gamma+k} s_x^{2\alpha+1} s_y^{2\beta+1} s_z^{2\gamma+1}, \end{aligned} \tag{7}$$

$$\begin{aligned} h_{zz} \mid_{ijk}^{m} &= -(-1)^{m+i+j+k} \sum_{\substack{\alpha+\beta+\gamma=m \\ \alpha\geqslant\alpha_{h,z},\beta\geqslant\beta_{h,z},\gamma\geqslant\gamma_{h,z}}} \binom{m}{\alpha\beta\gamma} \\ &\times \binom{2\alpha}{\alpha+i} \binom{2\beta}{\beta+j} \binom{2\gamma+2}{\gamma+k+1} s_x^{2\alpha+1} s_y^{2\beta+1} s_z^{2\gamma+3}. \end{aligned} \tag{8}$$

Other terms denote: $\alpha_x = \max(-i-1, i)$, $\beta_x = |j|$, $\gamma_x = \max(-k, k-1)$, $\alpha_y = |i|$, $\beta_y = \max(j, -j-1)$, $\gamma_y = \max(-k, k-1)$, $\alpha_{f,z} = \alpha_{h,z} = |i|$, $\beta_{f,z} = \beta_{h,z} = |j|$, $\gamma_{f,z} = |k|$, $\gamma_{h,z} = \max(|k|-1, 0)$. $U\,|^n$ and $\delta\,|_{ijk}$ respectively denote step and Kronecker delta functions. Δp is the discretization-step size along the p-direction ($p = x, y, z$) and $s_p = c\Delta t/\Delta p$ the Courant number. Expressions for other components of the $\mathbf{G_{ee}}$ DGF can be obtained rotating x, y, z subscripts and corresponding summation indices.

Equations (2)–(8) involve binomial coefficients, whose values are large integer numbers for high upper indices, whereas powers of Courant numbers may be very small real numbers. Numerical problems occur if these DGF equations are implemented in a common programming language with fixed-size precision arithmetic [9]. The DGF expressions have to be therefore implemented using the multiple precision arithmetic whose digits of precision are only limited by the available memory in a computing system. Final results of the DGF computations obtained using the multiple precision arithmetic are cast to fixed-size floating-point precision which is typical for standard computing architectures (32/64 bits).

3. PARALLEL IMPLEMENTATIONS OF THE DISCRETE GREEN'S FUNCTION COMPUTATIONS

An application of the DGF requires generation of its time-domain waveforms for a number of Yee's cells. The developed CPU-GPU implementation takes advantage of the CPU- and GPU-only codes but avoids their above-mentioned drawbacks. The implementations on CPU and GPU use respectively MPIR [34] and CUMP [35] multiple precision arithmetic libraries. The OpenMP [36] and CUDA [37] programming interfaces were employed for parallelization of computations on respectively CPU and GPU architecture.

The MPIR is an open-source library forked from the GNU multiple precision arithmetic library (GMP) [34]. This library was designed to provide the fastest possible arithmetic on CPUs for all applications that need higher precision than 64 bits. The speed of MPIR results from using fullwords as a basic arithmetic type, sophisticated algorithms, and optimized assembly code for many different CPUs. The advantage for the MPIR increases with the operand sizes since this library uses many asymptotically faster algorithms.

The CUMP is an open-source library optimized for execution of the multiple precision computations on GeForce 400/500 GPU architectures from Nvidia. The speed of CUMP results from using fullwords as a basic arithmetic type and an application of the register

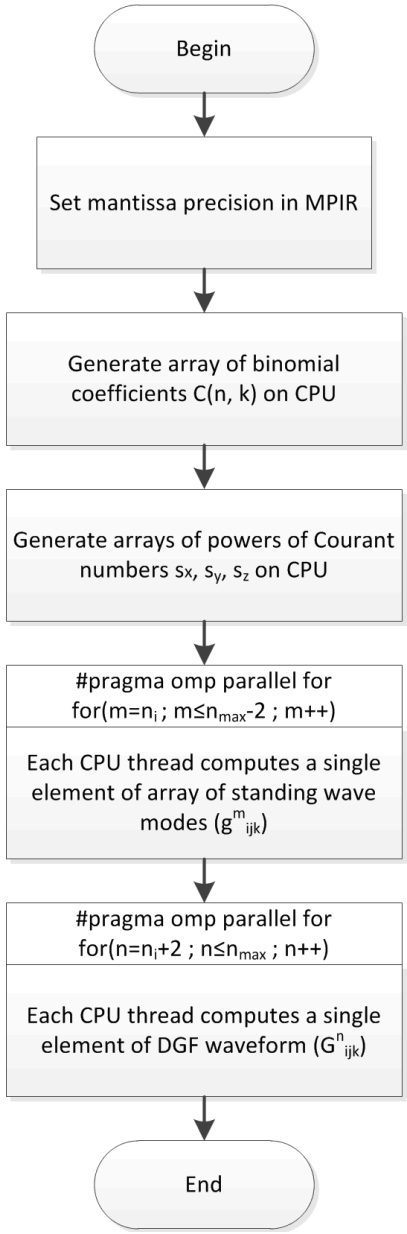


Figure 1. Flowchart of the DGF implementation on CPU.

blocking technique [35]. Unfortunately, the CUMP library employs basic multiplication algorithm which requires $O(n^2)$ operations. In spite of that, the GPU-only implementation of the DGF has already provided the sixfold speedup over the CPU-only implementation [17].

All presented DGF implementations start execution from setting the mantissa size of floating-point variables in the multiple precision arithmetic libraries. The mantissa size has to be large enough to guarantee convergence of the DGF computations. Afterwards, the lookup tables of binomial coefficients and powers of Courant numbers are generated. Sizes of these arrays depend on the upper limit of the time index ($n_{\max}$) of the computed DGF waveforms.

The CPU-only, GPU-only, and CPU-GPU implementations will be described below only for the electric field $G_{ee,xz}$ DGF component due to the sake of brevity.

3.1. Implementation on Multicore Central Processing Unit

Flowchart of the DGF implementation on a multicore CPU [16] is presented in Fig. 1. The $G_{ee,xz}$ component of the DGF is a summation of the g_{xz} standing wave modes multiplied by temporal coefficients [9]. It is efficient to compute values of the standing wave modes once and store the result in the lookup table since these values are employed many times in the $G_{ee,xz}$ waveform computations. Array of the standing wave modes is generated for indices from the range of the corresponding summation limits in (2). This step can be efficiently parallelized since the computation of every single standing wave mode represents an independent computational task. Afterwards, summation in (2) can also be executed in parallel.

The summation in expression for the standing wave mode (5) is executed in a loop, with the temporary result being a floating-point variable with user-selectable mantissa precision. In every loop step, integer result of multiplications of the binomial coefficients (from the lookup table) is calculated and cast to the floating-point variable. Afterwards, this variable is multiplied by powers of the Courant numbers (from the lookup table), and the result is added to the aforementioned temporary summation result in every loop step.

3.2. Implementation on Graphics Processing Unit

Flowchart of the DGF implementation on a GPU [17] is presented in Fig. 2. Analogously to the CPU implementation, values of the g_{xz} standing wave modes are computed once and the result is stored as the lookup table in a global memory on the GPU device. Although overhead of data transfer between the global memory and the GPU

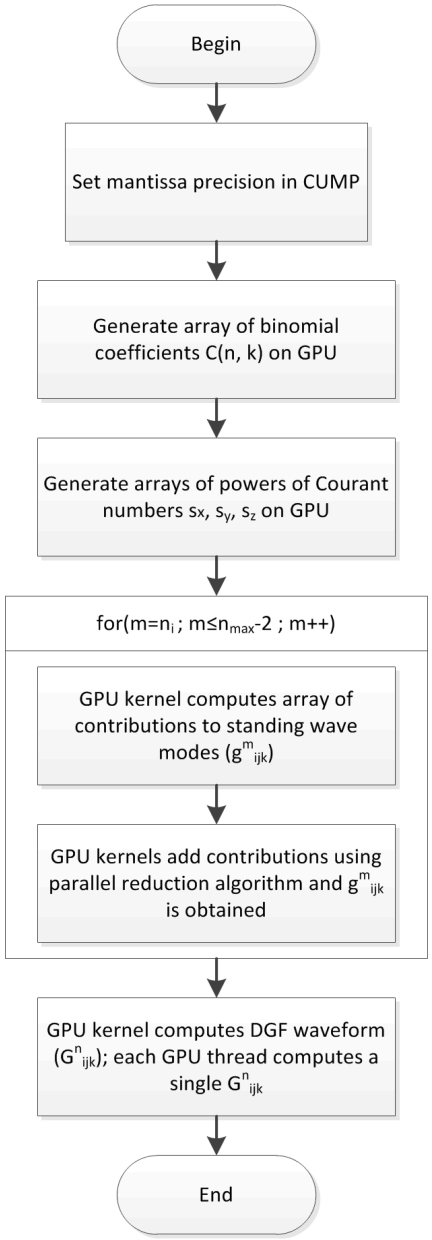


Figure 2. Flowchart of the DGF implementation on GPU.

is very high, it is still advantageous to employ this strategy rather than execute redundant computations requiring the multiple precision arithmetic. Access time to the global memory was identified as a rate-limiting step in the multiple precision computations on GPU [35]. Therefore, an interleaved memory layout was employed in the CUMP library to store arrays of multiple precision numbers. As a result, the library can take advantage of the coalesced memory access on GPU.

In the developed implementation, each CUDA thread within a kernel computes a single contribution to the summation in (5). A limited set of (α, β, γ) indices corresponds to the mth mode of the standing wave (5). Let us assume that (α_x, β_x, γ_x) are equal to zero for the sake of brevity. Then, α index takes values from the range $(0, 1, 2, \ldots, m)$. Corresponding values of β index are as follows:

$$\begin{aligned}
&\alpha = 0 \Longrightarrow \beta = 0, 1, \ldots, m \\
&\alpha = 1 \Longrightarrow \beta = 0, 1, \ldots, m-1 \\
&\ldots \\
&\alpha = m \Longrightarrow \beta = 0.
\end{aligned}$$

Therefore, the total number of threads required to represent the set of (α, β, γ) indices is equal to:

$$1 + 2 + 3 + \ldots + (m+1) = \frac{(m+1)(m+2)}{2} = \binom{m+2}{2}. \quad (9)$$

Such a number of threads concurrently runs and computes the contributions to the sum (5). The indices (α, β, γ) are computed for each thread based on its global index. If $(\alpha_x, \beta_x, \gamma_x)$ are non-zero numbers, then m is reduced and the described above procedure is still applicable. Then, final values of the (α, β, γ) indices have to be increased respectively by $(\alpha_x, \beta_x, \gamma_x)$.

Computational kernels employ device functions from the CUMP library to execute multiple precision computations. The binomial coefficients and the powers of the Courant numbers are multiplied and a result is stored in an array of the contributions to the g_{xz}. In the next step, summation in (5) is executed using the parallel reduction algorithm [38]. Afterwards, an output DGF waveform is computed using the formula (2) and g_{xz} values from the lookup table. In this case, each thread concurrently computes final result for the corresponding time index (n).

3.3. Implementation on Heterogeneous Parallel Processing System

Flowchart of the CPU-GPU heterogeneous implementation is presented in Fig. 3. After generation of the lookup tables, the

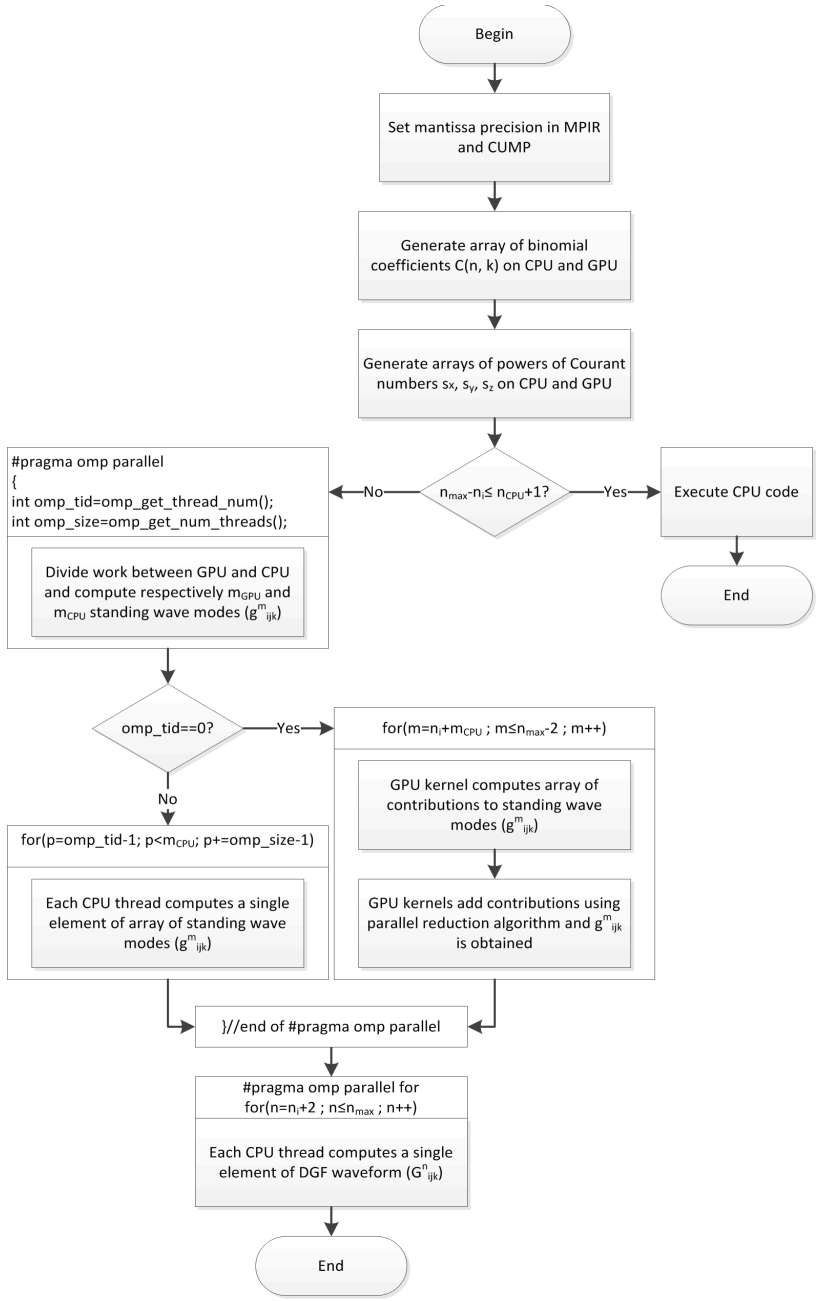


Figure 3. Flowchart of the DGF implementation on heterogeneous CPU-GPU parallel processing system.

algorithm decides if it is advantageous to execute computations only on the CPU. Such situation occurs for short enough DGF lengths whose runtime on the CPU is lower than time of the GPU initialization. The following condition is evaluated:

$$n_{\max} - n_i \leqslant n_{\mathrm{CPU}} + 1, \tag{10}$$

where $n_i = \alpha_x + \beta_x + \gamma_x$ and n_{CPU} denotes the number of standing wave modes solved only on CPU (value of this parameter depends on CPU and GPU throughputs).

If the condition is fulfilled, the CPU-only code is further executed. Otherwise, the parallel OpenMP section is opened with a single thread (`omp_tid=0`) controlling execution of the code on GPU and the rest of threads executing the DGF computations on CPU. Similar parallelization strategy has already been reported in the literature for the CPU-GPU heterogeneous processing system [39]. The load is distributed between both architectures to provide the highest acceleration of the DGF computations. The number of the g_{xz} modes required for computation of the $G_{ee,xz}$ component is equal to $n_{\max} - n_i - 1$. Lower m_{CPU} modes are computed on CPU whereas the rest of modes (m_{GPU}) is computed on GPU. The load distribution ratio between CPU and GPU is defined as:

$$\mathrm{LDR} = \frac{m_{\mathrm{CPU}}}{m_{\mathrm{CPU}} + m_{\mathrm{GPU}}}. \tag{11}$$

Value of this parameter is fixed for the DGF computations that are distributed between CPU and GPU. If the LDR is equal to 0 then the code is executed only on the GPU. On the other hand, if the LDR is equal to 1 then the code is only executed on the CPU.

In the CPU-GPU heterogeneous DGF implementation, values of the g_{xz} standing wave modes are computed once and the result is stored in the CPU memory. The described-above algorithms of the CPU- and GPU-only acceleration are employed in computations of the standing wave modes. Then, results computed on GPU in the multiple precision are sent to the host memory. Finally, summation in (2) is executed in parallel on CPU in order to obtain $G_{ee,xz}$ DGF.

4. NUMERICAL RESULTS

The described above DGF implementations were integrated with the FDTD solver. Numerical tests were executed in double floating-point precision on the workstation equipped with Intel i7 CPU (4 cores) and Nvidia GTX 680 GPU. The Courant numbers were taken as $s_x = s_y = s_z = 0.99/\sqrt{3}$ for the results presented here. For tests evaluating runtimes, the field waveforms were computed in position

$(i, j, k) = (10, 20, 30)$ from the source. It allows to easily compare speed of the developed method with other results already published in literature [9, 16, 17].

Figure 4 shows runtimes for the DGF generation using the CPU- and GPU-only implementations. Additional results for GTX 470 (belonging to the generation of GPUs that is older than GTX 680) are shown for the sake of demonstrating how characteristics depend on the computational power of the GPU. The electric field $G_{ee,xz}$ DGF waveform was computed from $n = 0$ to the upper limit $n_{\max}$ shown on the horizontal axis. As seen, GPU and CPU characteristics intersect for the $n_{\max}$ being equal to 140 and 170 respectively for GTX 680 and GTX 470. Below the intersection point, the reference CPU code outperforms developed GPU implementation. However, if the $n_{\max}$ value is above the intersection point, the GPU implementation outperforms reference CPU implementation. It demonstrates that the optimal approach to the DGF parallelization should indeed employ the CPU and GPU for the values of $n_{\max}$ best suited for each architecture.

Figure 5 shows the speedup of the CPU-GPU heterogeneous implementation relative to the GPU implementation vs. the LDR parameter value. As seen, optimal load distribution in the point $(i, j, k) = (10, 20, 30)$ is obtained for LDR ≈ 0.6. This LDR value is higher than 0.5 because computational overhead grows when the

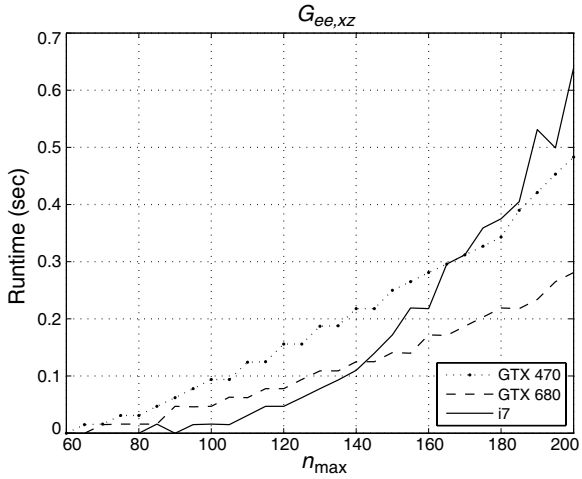


Figure 4. Runtimes for generating the electric field $G_{ee,xz}$ DGF for CPU- (i7) and GPU-only (GTX 470 and GTX 680) implementations vs. $n_{\max}$ value. Mantissa size of floating point variables was set to 2048 bits.

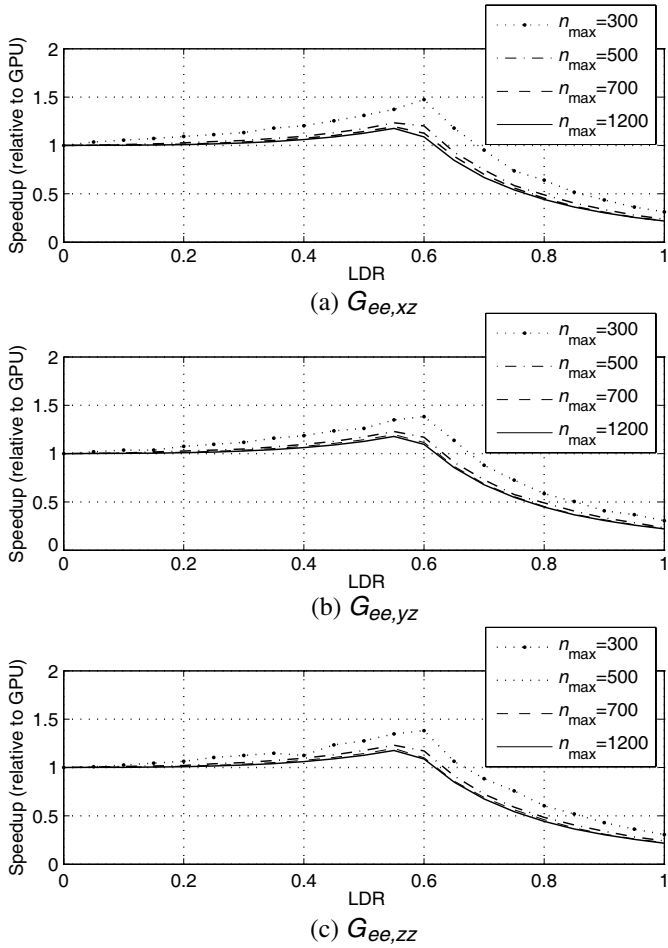


Figure 5. Speedup of the CPU-GPU heterogeneous implementation over the GPU-only implementation vs. load distribution ratio between CPU and GPU (LDR). The $G_{ee,xz}$ DGF waveforms were computed from $n = 0$ to the upper limit $n_{\max}$ equal to 300, 500, 700, and 1200 samples. Mantissa size of floating point variables was set to 2048 bits (300, 500, 700 samples) and 3072 bits (1200 samples). The n_{CPU} parameter was set to 80 samples.

m index of the standing wave mode is increased. In the developed method, the GPU computes standing wave modes associated with higher computational overhead. For short DGF waveforms, time of the GPU initialization is comparable to the total time of the DGF computations. The overhead of the GPU initialization is not hidden

by computations in this case. Therefore, the best speedup of the CPU-GPU heterogeneous implementation is observed for $n_{\max} = 300$ due to the worse performance of the reference GPU-only implementation.

Figure 6 shows runtimes for the DGF generation on the CPU-GPU heterogeneous parallel processing system. The electric field $\mathbf{G_{ee}}$ DGF waveforms were computed from $n = 0$ to the upper limit $n_{\max}$ shown on the horizontal axis. Mantissa size of floating point

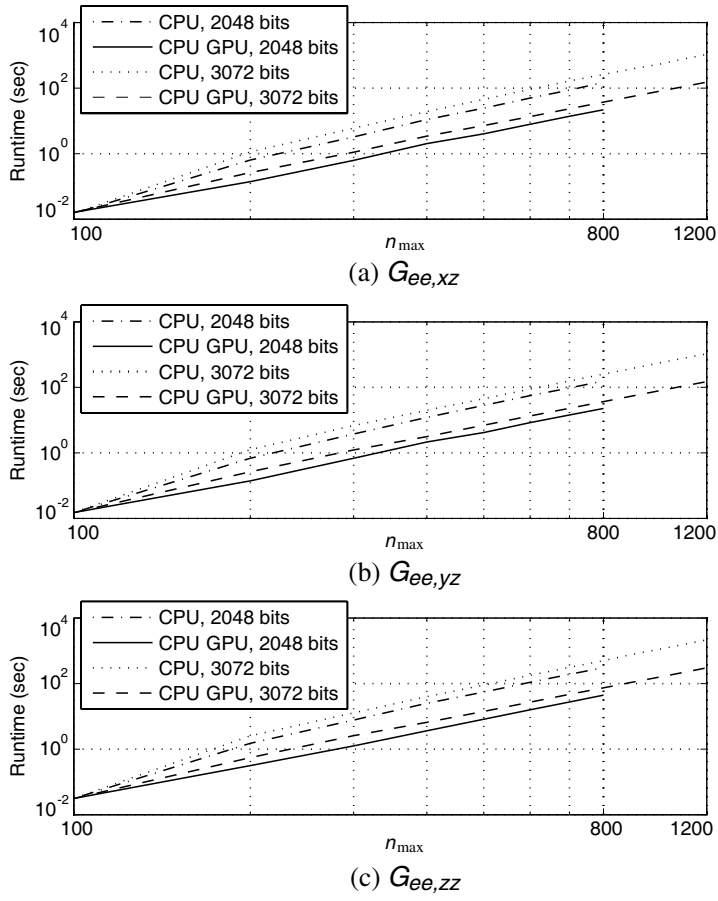
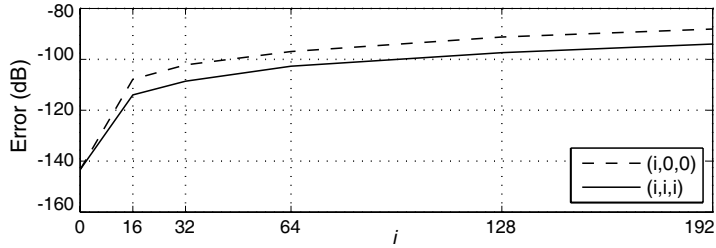
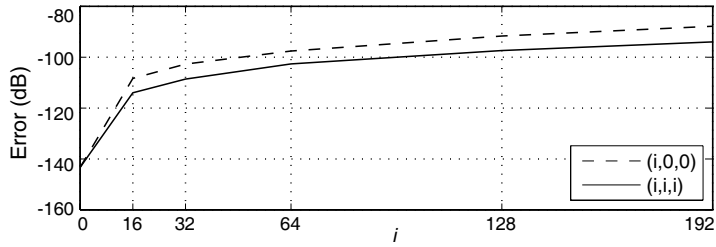


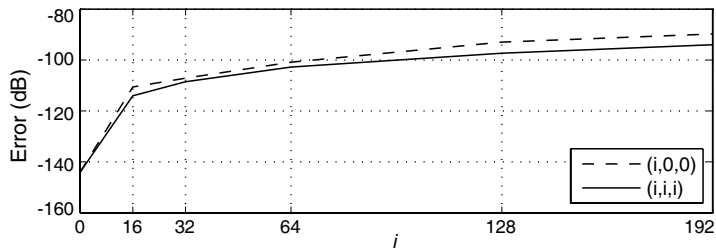
Figure 6. Runtimes for generating the electric field $\mathbf{G_{ee}}$ DGF from $n = 0$ to the upper limit $n_{\max}$. Results for the CPU-GPU heterogeneous implementation are compared with the CPU-only implementation. Mantissa size of floating point variables was set to 2048 and 3072 bits. Parameters of the CPU-GPU heterogeneous implementation are as follows: LBR = 0.6 and $n_{\mathrm{CPU}} = 80$.



(a) $G_{ee,xz}$



(b) $G_{ee,yz}$



(c) $G_{ee,zz}$

Figure 7. Maximum relative error in the electric field $\mathbf{G_{ee}}$ DGF for the first 708 time-steps as a function of the cell position in the axial $(i, 0, 0)$ and diagonal (i, i, i) directions from the source. Mantissa size of floating-point variables was set to 2048 bits.

variables was set to 2048 and 3072 bits. There is an upper limit of the n time index, for which DGF computations are still convergent for user-selectable mantissa precision. Therefore, characteristics terminate at the maximum value of the n index assuring the convergence. Increase of the runtime is observed if the precision of the floating-point computations is increased. The results presented for the CPU-GPU heterogeneous implementation include overheads of the GPU initialization and data transfer between host and the GPU device. In spite of that, the characteristics do not intersect as it was for the GPU-only implementation [17]. Computation of 1200

time samples of the $G_{ee,xz}$ DGF waveform lasts 151 sec (2048 bits of mantissa size) whereas it requires more than 1062 sec for the CPU-only implementation. It results in the sevenfold speedup over the CPU-only implementation whereas the sixfold speedup was obtained for the GPU-only implementation [17]. Computation of 600 time samples of the $G_{ee,xz}$ DGF waveform lasts 7.8 sec (2048 bits of mantissa size) whereas it required more than 10^5 sec for the seminal implementation of the closed-form DGF expression [9]. It can be concluded that the speedup factor of four orders of magnitude compared to the seminal implementation was obtained. Of course, validity of such a comparison is limited since both implementations were developed using different software tools and were executed on different machines.

Figure 7 shows accuracy of the $\mathbf{G_{ee}}$ DGF implementation on the CPU-GPU heterogeneous parallel processing system. Evaluation of the accuracy, in comparison to the standard FDTD method, required execution of the reference simulations for sufficiently large domains (grid size: 920^3) terminated by ABCs. The relative error between DGF waveforms generated using the developed technique ($E\,|_{ijk}^{n}$) and the reference FDTD simulation ($E_{ref}\,|_{ijk}^{n}$) was defined as:

$$\text{Error} = 20\log_{10}\left[\frac{\max\left|E\,|_{ijk}^{n} - E_{ref}\,|_{ijk}^{n}\right|}{\max\left|E_{ref}\,|_{ijk}^{n}\right|}\right]\text{(dB)}. \tag{12}$$

As seen, the error increases if distance to the source is increased. Error is slightly larger for axial direction than for diagonal direction in the Yee's grid. The error is below -90 dB for the first 708 time-steps of the FDTD update procedure.

5. CONCLUSIONS

Implementation of the new analytic closed-form expression for the 3-D dyadic discrete Green's function on the heterogeneous parallel processing system is presented in this paper for the first time. Fast and accurate computations of the discrete Green's function require the hardware acceleration and the multiple precision arithmetic. The developed implementation simultaneously utilizes computational power of the central processing unit and the graphics processing unit to the computational tasks best suited for each architecture.

The sevenfold speedup was obtained relative to the reference code executed on a multicore central processing unit. The developed implementation will facilitate further applications of the discrete Green's function in the FDTD simulations of radiation and scattering problems.

ACKNOWLEDGMENT

This work was realized within the HOMING PLUS Program of the Foundation for Polish Science, co-financed from the European Union Regional Development Fund. The author is grateful to T. Nakayama for assistance with, and access to, the CUMP library.

REFERENCES

1. Chew, W. C., "Electromagnetic theory on a lattice," *Journal of Applied Physics*, Vol. 75, No. 10, 4843–4850, 1994.
2. Clemens, M. and T. Weiland, "Discrete electromagnetism with the finite integration technique," *Progress In Electromagnetics Research*, Vol. 32, 65–87, 2001.
3. Schuhmann, R. and T. Weiland, "Conservation of discrete energy and related laws in the finite integration technique," *Progress In Electromagnetics Research*, Vol. 32, 301–316, 2001.
4. Bossavit, A., "'Generalized finite differences' in computational electromagnetics," *Progress In Electromagnetics Research*, Vol. 32, 45–64, 2001.
5. Teixeira, F. L., "Geometric aspects of the simplicial discretization of Maxwell's equations," *Progress In Electromagnetics Research*, Vol. 32, 171–188, 2001.
6. Vazquez, J. and C. G. Parini, "Discrete Green's function formulation of FDTD method for electromagnetic modelling," *Electron. Lett.*, Vol. 35, No. 7, 554–555, 1999.
7. Holtzman, R. and R. Kastner, "The time-domain discrete Green's function method (GFM) characterizing the FDTD grid boundary," *IEEE Trans. Antennas Propag.*, Vol. 49, No. 7, 1079–1093, 2001.
8. Holtzman, R, R. Kastner, E. Heyman, and R. W. Ziolkowski, "Stability analysis of the Green's function method (GFM) used as an ABC for arbitrarily shaped boundaries," *IEEE Trans. Antennas Propag.*, Vol. 50, No. 7, 1017–1029, 2002.
9. Jeng, S.-K., "An analytical expression for 3-D dyadic FDTD-compatible Green's function in infinite free space via z-transform and partial difference operators," *IEEE Trans. Antennas Propag.*, Vol. 59, No. 4, 1347–1355, 2011.
10. Vazquez, J. and C. G. Parini, "Antenna modelling using discrete Green's function formulation of FDTD method," *Electron. Lett.*, Vol. 35, No. 13, 1033–1034, 1999.
11. Ma, W., M. R. Rayner, and C. G. Parini, "Discrete Green's

function formulation of the FDTD method and its application in antenna modeling," *IEEE Trans. Antennas Propag.*, Vol. 53, No. 1, 339–346, 2005.

12. Holtzman, R, R. Kastner, E. Heyman, and R. W. Ziolkowski, "Ultra-wideband cylindrical antenna design using the Green's function method (GFM) as an absorbing boundary condition (ABC) and the radiated field propagator in a genetic optimization," *Microw. Opt. Tech. Lett.*, Vol. 48, No. 2, 348–354, 2006.
13. De Hon, B. P. and J. M. Arnold, "Stable FDTD on disjoint domains — A discrete Green's function diakoptics approach," *Proc. The 2nd European Conf. on Antennas and Propag. (EuCAP)*, 1–6, 2007.
14. Malevsky, S., E. Heyman, and R. Kastner, "Source decomposition as a diakoptic boundary condition in FDTD with reflecting external regions," *IEEE Trans. Antennas Propag.*, Vol. 58, No. 11, 3602–3609, 2010.
15. Schneider, J. B. and K. Abdijalilov, "Analytic field propagation TFSF boundary for FDTD problems involving planar interfaces: PECs, TE, and TM," *IEEE Trans. Antennas Propag.*, Vol. 54, No. 9, 2531–2542, 2006.
16. Stefanski, T. P., "Fast implementation of FDTD-compatible Green's function on multicore processor," *IEEE Antennas Wireless Propag. Lett.*, Vol. 11, 81–84, 2012.
17. Stefanski, T. P. and K. Krzyzanowska, "Implementation of FDTD-compatible Green's function on graphics processing unit," *IEEE Antennas Wireless Propag. Lett.*, Vol. 11, 1422-1425, 2012.
18. Sypek, P., A. Dziekonski, and M. Mrozowski, "How to render FDTD computations more effective using a graphics accelerator," *IEEE Trans. Magn.*, Vol. 45, No. 3, 1324–1327, 2009.
19. Toivanen, J. I., T. P. Stefanski, N. Kuster, and N. Chavannes, "Comparison of CPML implementations for the GPU-accelerated FDTD solver," *Progress In Electromagnetics Research M*, Vol. 19, 61–75, 2011.
20. Tay, W. C., D. Y. Heh, and E. L. Tan, "GPU-accelerated fundamental ADI-FDTD with complex frequency shifted convolutional perfectly matched layer," *Progress In Electromagnetics Research M*, Vol. 14, 177–192, 2010.
21. Stefanski, T. P. and T. D. Drysdale, "Acceleration of the 3D ADI-FDTD method using graphics processor units," *IEEE MTT-S International Microwave Symposium Digest*, 241–244, 2009.
22. Xu, K., Z. Fan, D.-Z. Ding, and R.-S. Chen, "GPU accelerated

unconditionally stable Crank-Nicolson FDTD method for the analysis of three-dimensional microwave circuits," *Progress In Electromagnetics Research*, Vol. 102, 381–395, 2010.

23. Shahmansouri, A. and B. Rashidian, "GPU implementation of split-field finite-difference time-domain method for Drude-Lorentz dispersive media," *Progress In Electromagnetics Research*, Vol. 125, 55–77, 2012.
24. Zainud-Deen, S. H. and E. El-Deen, "Electromagnetic scattering using GPU-based finite difference frequency domain method," *Progress In Electromagnetics Research B*, Vol. 16, 351–369, 2009.
25. Demir, V., "Graphics processor unit (GPU) acceleration of finite-difference frequency-domain (FDFD) method," *Progress In Electromagnetics Research M*, Vol. 23, 29–51, 2012.
26. Dziekonski, A., A. Lamecki, and M. Mrozowski, "GPU acceleration of multilevel solvers for analysis of microwave components with finite element method," *IEEE Microw. Wireless Comp. Lett.*, Vol. 21, No. 1, 1–3, 2011.
27. Dziekonski, A., A. Lamecki, and M. Mrozowski, "Tuning a hybrid GPU-CPU V-cycle multilevel preconditioner for solving large real and complex systems of FEM equations," *IEEE Antennas Wireless Propag. Lett.*, Vol. 10, 619–622, 2011.
28. Dziekonski, A., P. Sypek, A. Lamecki, and M. Mrozowski, "Finite element matrix generation on a GPU," *Progress In Electromagnetics Research*, Vol. 128, 249–265, 2012.
29. Dziekonski, A., A. Lamecki, and M. Mrozowski, "A memory efficient and fast sparse matrix vector product on a GPU," *Progress In Electromagnetics Research*, Vol. 116, 49–63, 2011.
30. Peng, S. and Z. Nie, "Acceleration of the method of moments calculations by using graphics processing units," *IEEE Trans. Antennas Propag.*, Vol. 56, No. 7, 2130–2133, 2008.
31. Xu, K., D. Z. Ding, Z. H. Fan, and R. S. Chen, "Multilevel fast multipole algorithm enhanced by GPU parallel technique for electromagnetic scattering problems," *Microw. Opt. Technol. Lett.*, Vol. 52, No. 3, 502–507, 2010.
32. Lopez-Fernandez, J. A., M. Lopez-Portugues, Y. Alvarez-Lopez, C. Garcia-Gonzalez, D. Martinez, and F. Las-Heras, "Fast antenna characterization using the sources reconstruction method on graphics processors," *Progress In Electromagnetics Research*, Vol. 126, 185–201, 2012.
33. Gao, P. C., Y. B. Tao, Z. H. Bai, and H. Lin, "Mapping the SBR and TW-ILDCs to heterogeneous CPU-GPU architecture

for fast computation of electromagnetic scattering," *Progress In Electromagnetics Research*, Vol. 122, 137–154, 2012.

34. Granlund, T., "The multiple precision integers and rationals library," Edition 2.2.1, GMP Development Team, 2010, http://www.mpir.org.
35. Nakayama, T. and D. Takahashi, "Implementation of multiple-precision floating-point arithmetic library for GPU computing," *Proc. 23rd IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, 343–349, 2011.
36. OpenMP Architecture Review Board, "OpenMP application program interface," Version 3.1, July 2011, www.openmp.org.
37. Nvidia, "CUDA C programming guide," Version 4.2, http://developer.nvidia.com/cuda/nvidia-gpu-computing-documentation.
38. Harris, M., "Optimizing parallel reduction in CUDA," NVIDIA Developer Technology, http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf.
39. Shen, W., D. Wei, W. Xu, X. Zhu, and S. Yuan, "Parallelized computation for computer simulation of electrocardiograms using personal computers with multi-core CPU and general-purpose GPU," *Computer Methods and Programs in Biomedicine*, Vol. 100, No. 1, 87–96, 2010.