Progress In Electromagnetics Research B, Vol. 67, 137–151, 2016

GPU Accelerated Discontinuous Galerkin Time Domain Algorithm for Electromagnetic Problems of Electrically Large Objects

Lei Zhao[1, 2, *], Geng Chen[1], and Wenhua Yu[1]

Abstract—In this paper, an efficient time domain simulation algorithm is proposed to analyze the electromagnetic scattering and radiation problems. The algorithm is based on discontinuous Galerkin time domain (DGTD) method and parallelization acceleration technique using the graphics processing units (GPU), which offers the capability for accelerating the computational electromagnetics analyses. The bottlenecks using the GPU DGTD acceleration for electromagnetic analyses are investigated, and potential strategies to alleviate the bottlenecks are proposed. We first discuss the efficient parallelization strategies handling the local-element differentiation, surface integrals, RK time-integration assembly on the GPU platforms, and then, we explore how to implement the DGTD method on the Compute Unified Device Architecture (CUDA). The accuracy and performance of the DGTD method are analyzed through illustrated benchmarks. We demonstrate that the DGTD method is better suitable for GPUs to achieve significant speedup improvement over modern multi-core CPUs.

1. INTRODUCTION

During the last decades, finite element method (FEM), finite difference time domain (FDTD) and method of moments (MoM) have been dominant in computational electromagnetic methods [1–6]. These methods have their strengths and weaknesses when applied to solve electromagnetic problems. Recently, discontinuous Galerkin time domain (DGTD) is possible to construct a novel time domain numerical method [7–12], which has most of the FDTD advantages: spatially explicit algorithm, simplicity, easy parallelization, easy porting to GPUs, memory and computational cost only growing linearly with the number of elements, while it retains the most benefits from the FEM method such as adaptability of the unstructured meshes and spatial high-order convergence, which enable us to deal with the problems requiring precision variation over the entire domain or the solution lack of smoothness. The DGTD method has met an increased interest in the purpose of simulating complex practical problems. Indeed, the DGTD method can be seen as a crossover approach between the finite element time domain (FETD) method [13] whose accuracy depends on the order of basis function and the finite volume time domain (FVTD) method [14] whose neighboring cells are connected by numerical fluxes.

The parallelization technique of the DGTD method employs a widely used single program multiple data (SPMD) strategy that combines a partitioning of the underlying mesh and a message passing programming model [15]. Concerning the mesh partition, the compact nature of the DGTD method naturally lends itself to an element-wise decomposition, and thus yields a minimal overlap interface between neighboring subdomains. Such a kind of partition introduces a set of artificial interfaces that simply consists in triangulated surfaces. For each pair of neighboring elements, such an interface is duplicated in the topological definition of the concerned elements and the computation of numerical fluxes for these artificial boundary faces is performed twice on the both sides.

Received 18 February 2016, Accepted 18 April 2016, Scheduled 10 May 2016

* Corresponding author: Lei Zhao (lzhaomax@163.com).

[1] Center for Computational Science and Engineering, Jiangsu Normal University, China. [2] State key Laboratory of Millimeter Waves, Southeast University, China.

For more than three decades, the computer industry has witnessed an ever increasing drive of performance improvement. Designing more capable microprocessors have evolved towards the development of multi-core CPUs or multi-CPU workstations that effectively multiply the performance with multiple cores on the same chip or multiple CPUs on the same motherboard if they are working on one problem simultaneously. One other particularly noteworthy approach is the use of many-core devices such as GPUs [16–20] or Intel PHI coprocessors (aka, many integrated core (MIC)) [5], which accelerate computations orchestrated by a conventional CPU program. The PHI coprocessor is for the first time introduced to the EM simulation in 2013 by Yu and his colleagues [21]. A recent paper [17] discussed the adaptation of a multirate time stepping based DGTD method for solving the time-domain Maxwell's equations on a multiple GPU system.

In this paper, the GPU acceleration technique for the DGTD methods is developed based on the CUDA platform [22]. The computational cost including memory usage, spatial discretization, simulation time and accuracy is investigated, and various comparisons with the existing analytical and numerical methods have been made for typical EM problems. The implementation of the DGTD method mainly includes element local differentiation, flux extraction and flux lifting three parts, and we measure the time consumption of local differentiation, flux extraction and flux lifting with the basis function order increasing. It is evident from the numerical experiments that the GPU-based DGTD method is an accurate, reliable and fast algorithm for most practical EM problems.

2. THEORY AND METHOD FOR DGTD

Maxwell's equations are the general partial equations to describe the wave propagation and intersection between wave and objects. Its differential form can be expressed as follows:

$$\partial_t \mathbf{D} = \nabla \times \mathbf{H} + \mathbf{J} \tag{1}$$

$$\partial_t \mathbf{B} = -\nabla \times \mathbf{E} \tag{2}$$

$$\nabla \cdot \mathbf{D} = \rho \tag{3}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{4}$$

where, $\mathbf{E}$ is the electric field, $\mathbf{D}$ the electric flux density, $\mathbf{H}$ the magnetic field, and $\mathbf{B}$ the magnetic flux density. For the isotropic materials, they are related through the formula below:

$$\mathbf{D} = \varepsilon_0 \varepsilon_r \mathbf{E} \text{ and } \mathbf{B} = \mu_r \mu_0 \mathbf{H} \tag{5}$$

where ε_0 and μ_0 are the permittivity and permeability in free space, respectively, and ε_r and μ_r are the relative permittivity and permeability, respectively. The current $\mathbf{J}$ is typically related to the electric field $\mathbf{E}$, through Ohms law, $\mathbf{J} = \sigma \mathbf{E}$, where σ is the finite conductivity. In the source free region, Eqs. (1) and (2) can be expressed as the conservation formulation.

$$Q\partial_t \mathbf{q} + \nabla \cdot \mathbf{F} = 0 \tag{6}$$

where $Q = \begin{bmatrix} \varepsilon_r & 0 \\ 0 & \mu_r \end{bmatrix}$, $\mathbf{q} = \begin{bmatrix} \mathbf{E} \\ \mathbf{H} \end{bmatrix}$, $\mathbf{F} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix}^T$, $F_i = \begin{bmatrix} -e_i \times \mathbf{H} \\ e_i \times \mathbf{E} \end{bmatrix}$, and e_i is the unitary vector along the axis in the Cartesian coordinate system. Assuming that $\bar{\mathbf{q}}(\mathbf{r}, t)$ is an approximation solution of Eq. (6), since $\mathbf{q}(\mathbf{r}, t)$ and $\bar{\mathbf{q}}(\mathbf{r}, t)$ are not identical, the left hand side of Eq. (6) is not equal to zero after $\bar{\mathbf{q}}(\mathbf{r}, t)$ is used in Eq. (6), namely,

$$Q\partial_t \bar{\mathbf{q}}(\mathbf{r}, t) + \nabla \cdot \mathbf{F}(\bar{\mathbf{q}}(\mathbf{r}, t)) = res(\mathbf{r}, t) \tag{7}$$

The 3-D domain Ω is discretized into a set of tetrahedrons $\{\Omega_i | i = 1, 2, \ldots, N\}$. Assume that the approximation solution $\bar{\mathbf{q}}(\mathbf{r}, t) = \begin{bmatrix} \mathbf{E}_N \\ \mathbf{H}_N \end{bmatrix}$ can be expressed as a linear function $L_k(x)$ inside the local region k, namely:

$$\overline{\mathbf{q}}_N^k(\mathbf{r}, t) = \sum_{i=1}^{N_p} \mathbf{q}(\mathbf{r}_i,\ t)\, L_i(\mathbf{r}) \tag{8}$$

where N_p is the order of basis function and $L_k(x)$ the basis function for the local region k. 3-D Lagrange basis function is defined as follows:

$$\mathrm{L}_i(\mathbf{x}) = \sum_{k,l,m=0}^{k+l+m\leq p} a_{k,l,m}^i x^k y^l z^m \tag{9}$$

And then the total number of points can be expressed

$$N_p = \frac{(n+1)(n+2)(n+3)}{6} \tag{10}$$

where, n is the number of Lagrange order. The Lagrange polynomial is selected to be basis function because the field is equal to its coefficients in its Lagrange expansion. The grid point in the Lagrange polynomial is selected to be on the Lagrange-Gauss-Labatto (LGL) points [23], as shown in Fig. 1.

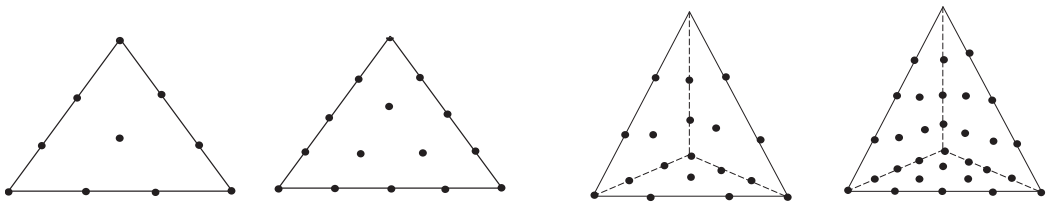


Figure 1. Relationship between the number of Lagrange function order and points in each element. (a) 3-order polynomial and 10 points (2-D); (b) 4-order polynomial and 15 points (2-D); (c) 3-order polynomial and 20 points (3-D); and (d) 4-order polynomial and 35 points (3-D).

We now use Galerkin's method to calculate the weighted residual [7]. Assuming that the test function is the same as the basis function $L_k(\mathbf{r})$, projecting the residual on the right side of Eq. (7) onto the test function $L_k(\mathbf{r})$, and letting the projection to be zero, we have:

$$\int_{\Omega_k} res(\mathbf{r},t)\, L_k(\mathbf{x})\, d\mathbf{r} = 0 \tag{11}$$

Inserting the expression of residual $res(\mathbf{r}, t)$ into Eq. (11), we have:

$$\int_{\Omega_k} \left[Q\partial_t \bar{\mathbf{q}}(\mathbf{r},t) + \nabla\cdot\mathbf{F}(\bar{\mathbf{q}}(\mathbf{r},t))\right] L_k(\mathbf{r})\, d\mathbf{r} = 0 \tag{12}$$

Performing the spatial integration by parts, we have the weak coupling formulation:

$$\int_{\Omega_k} \left(Q\partial_t\bar{\mathbf{q}}\mathrm{L}_i(\mathbf{r}) - \mathbf{F}(\bar{\mathbf{q}})\cdot\nabla\mathrm{L}_i(\mathbf{r})\right) d\mathrm{v} = \int_{\partial\Omega} (\hat{\mathbf{n}}\cdot\mathbf{F}(\bar{\mathbf{q}}))\, L_i(\mathbf{r}) ds \tag{13}$$

where, $\hat{n}$ is the outward unitary vector of the DGTD element. Performing the integration by part for Eq. (13), we have the strength coupling formulation as follows:

$$\int_{\Omega_k} \left(Q\partial_t\bar{\mathbf{q}}\mathrm{L}_i(\mathbf{r}) - \nabla\cdot\mathbf{F}(\bar{\mathbf{q}})\,\mathrm{L}_i(\mathbf{r})\right) d\mathbf{r} = \int_{\partial\Omega_k} \hat{\mathbf{n}}\cdot(\mathbf{F}(\bar{\mathbf{q}}) - \mathbf{F}^*(\bar{\mathbf{q}}))\mathrm{L}_i(\mathbf{r}) ds \tag{14}$$

In this paper, we only consider the strength coupling case in the DGTD implementation. Since there are two points at the same position on the two sides of interface, we can use the first-order upwind flux to approximate the right side of Eq. (14), and then we have:

$$\hat{\mathbf{n}}\cdot(\mathbf{F}(\bar{\mathbf{q}}) - \mathbf{F}^*(\bar{\mathbf{q}})) = \frac{1}{2}\begin{bmatrix} \dfrac{1}{\mathrm{Y}^+ + \mathrm{Y}^-}\hat{\mathbf{n}}\times(Z^+(\mathbf{H}_N^+ - \mathbf{H}_N^-) - \hat{\mathbf{n}}\times(\mathbf{E}_N^+ - \mathbf{E}_N^-)) \\ \dfrac{1}{\mathrm{Z}^+ + \mathrm{Z}^-}\hat{\mathbf{n}}\times(-Y^+(\mathbf{E}_N^+ - \mathbf{E}_N^-) - \hat{\mathbf{n}}\times(\mathbf{H}_N^+ - \mathbf{H}_N^-)) \end{bmatrix} \tag{15}$$

where $(\mathbf{E}_N^-, \mathbf{H}_N^-)$ refers to the local solution on the trace of element Ω_k, $(\mathbf{E}_N^+, \mathbf{H}_N^+)$ refers to the trace of $(\mathbf{E}_N, \mathbf{H}_N)$ along the shared face in the neighboring element. $\mathrm{Z}^{\pm} = \sqrt{\mu_r^{\pm}/\varepsilon_r^{\pm}}$ is the local dielectric impedance and $\mathrm{Y}^{\pm} = \sqrt{\varepsilon_r^{\pm}/\mu_r^{\pm}}$ is the local dielectric admittance. The upwind scheme is conditionally stable and has a better accuracy. Then, when the tetrahedral meshes are used we can obtain the semi-discrete system of Maxwell's equations, and the semi-discrete system can be expressed as:

$$\partial_t \mathbf{E}_N = \frac{1}{\varepsilon_r}(M)^{-1}\left(\mathbf{S}\times\mathbf{H}_N + \sum_{n=1}^{4}\left(G_{s_n}\frac{\left(\mathbf{E}_N^+ - \mathbf{E}_N^-\right) - \hat{\mathbf{n}}\left(\hat{\mathbf{n}}\cdot\left(\mathbf{E}_N^+ - \mathbf{E}_N^-\right)\right) + Z^+\hat{n}\times\left(\mathbf{H}_N^+ - \mathbf{H}_N^-\right)}{Z^+ + Z^-}\right)\right) \tag{16}$$

$$\partial_t \mathbf{H}_N = \frac{1}{\mu_r}(M)^{-1}\left(\mathbf{S}\times\mathbf{E}_N + \sum_{n=1}^{4}\left(G_{s_n}\frac{\left(\mathbf{H}_N^+ - \mathbf{H}_N^-\right) - \hat{\mathbf{n}}\left(\hat{\mathbf{n}}\cdot\left(\mathbf{H}_N^+ - \mathbf{H}_N^-\right)\right) + Y^+\hat{n}\times\left(\mathbf{E}_N^+ - \mathrm{E}_N^-\right)}{Y^+ + Y^-}\right)\right) \tag{17}$$

In which, the mass matrix is defined as:

$$M_{i,j} = \int_{\Omega_k} \mathrm{L}_i\left(\mathbf{r}\right)\mathrm{L}_j\left(\mathbf{r}\right)d\mathrm{v} \tag{18}$$

$\mathbf{S} = (S_x, S_y, S_z)^T$ is a stiffness matrix vector. $S_\zeta,\ \zeta = x, y, z$ are the stiffness matrices in the physical space Ω_k and are expressed as:

$$\left(S_\zeta\right)_{i,j} = \int_{\Omega_k} \mathrm{L}_i\left(\mathbf{r}\right)\partial_\zeta \mathrm{L}_j\left(\mathbf{r}\right)d\mathrm{v}, \quad \zeta = x, y, z \tag{19}$$

G_{s_n} is a face mass matrix for the surface of tetrahedron, and can be expressed as:

$$\left(G_{s_n}\right)_{i,j} = \int_{s_n} \mathrm{L}_i\left(\mathbf{r}\right)L_j\left(\mathbf{r}\right)ds, \quad \mathbf{r}\in s_n \tag{20}$$

Taking the source term $s(t)$ into account, we write Eqs. (16) and (17) in the simplified format:

$$\partial_t q = Qq + s(t) \tag{21}$$

where, Q is the operator in the semi-discrete system. Its more common expression can be in the following form:

$$\partial_t q\left(\mathbf{r}\right) = f\left(q\left(\mathbf{r}\right), t\right) \tag{22}$$

There are many ways to solve the equation above. The accurate solution to DGTD update equations requires using high-order polynomial. To reduce the memory usage in the DGTD method, we use the five-stage and fourth-order accurate and Runge-Kutta method to solve the DGTD update equations [24].

$$q_0 = q(t_n) \tag{23a}$$

$$R_i = a_i R_{i-1} + \Delta t f(q_{i-1}, t_n + c_i\Delta t) \tag{23b}$$

$$q_i = q + b_i R_i \tag{23c}$$

$$q(t_{n+1}) = q_4 \tag{23d}$$

where, the subscript i is taken to be 1, 2, 3, 4, and 5, which means that the number of stages is 5. The coefficients a_i, b_i, and c_i define the properties (order of accuracy, stability contour) of the low storage Runge-Kutta (LSRK) scheme. The time evolution of the electromagnetic fields is most conveniently accomplished using low-storage Runge-Kutta method for a number of reasons. First, it is desirable to accompany the higher-order accurate DG discretization in space with a higher-order accurate time stepping scheme. Suitable LSRK schemes are available for up to fourth order. Secondly, given a total number of N expansion coefficients, LSRK methods require a total of $2N$ values stored in two registers q and R.

As an explicit method, the LSRK scheme is subject to the Courant-Friedrichs-Lewy condition (CFL condition). As soon as the time step Δt exceeds a critical time step, the numerical solution is subject to unphysical exponential growth. The critical time step, for which the numerical solution just does not

grow exponentially, depends both on the spatial disretization, on the time stepping scheme and on the physical model. Similar to the popular FDTD algorithm, the maximum time step for LSRK scheme is governed by

$$\Delta t_{\max} \leq \frac{\text{mesh width}}{\text{v}} \tag{24}$$

where, v is the wave speed in dielectric medium. The smallest distance between nodes within an element influences the DGTD time step. The higher the order and the more deformed an element is, the smaller the maximum time step will be.

3. GPU ACCELERATION FOR DGTD METHOD

GPU has been popular with fast development of high performance computing techniques for a dozen of years. Here we investigate the GPU acceleration technique for the DGTD method based the CUDA platform [25]. GPU has evolved into a high performance computing due to its tremendous computational power and very high memory bandwidth. The main concerns about GPU in the past years are portable; however, CUDA as a general computation platform completely changes this situation. As a C-like programming language with its own compiler and libraries, CUDA significantly makes the GPU codes portable [26, 27]. A GPU consists of several texture processing clusters. Each cluster consists of a large block of texture fetch units and several streaming multiprocessors, which comprises eight computing units and two super functional units. Each multiprocessor has certain resources, for example, special shared memory. It is not cache but programmers may use it. This shared memory allows to exchange data between threads of a single block. All threads of a single block are always executed by the same multiprocessor. Threads from different blocks cannot exchange data between each other. Shared memory is often useful, except those cases when several threads access the same memory bank. Multiprocessors can access video memory as well, but it involves high latencies. To accelerate memory access and reduce the frequency of video memory calls, each multiprocessor has a small amount of cache for constants and texture data.

In order to use GPU effectively, it is also important to use buffering to store data (registers, shared memory, etc.) and minimize data exchange between CPU and GPU. When optimizing GPU applications, it is critical to achieve an optimal balance between the size and number of blocks. More threads in a block will reduce the effect of memory latencies but will also reduce the number of available registers. NVIDIA recommends using blocks of 128 or 256 threads as a compromise value to reach optimal latencies and number of registers. Two key optimization methods for the DGTD GPU acceleration include: (1) using shared memory as much as possible, because it is much faster than global video memory; (2) reading and writing from global memory must be coalesced, if possible. For this purpose the special data types to read and write 32/64/128 bits in a single operation is important. If it is difficult to coalesce reading operations, we may try to use texture lookups.

During the preprocessing, the local computation is converted to the matrix-vector-products. The DGTD implementation on the GPU platforms comprises the kernels, differentiation, surface integrals, and the Runge-Kutta integration. Some elementary matrices such as the differentiation matrix on the reference element are stored in the texture memory because there are constants in memory and texture memory has texture cache. Using texture memory presents some benefits compared to read from global or constant memory. The basic procedure of the DGTD GPU acceleration is summarized in the steps below:

(1) Load constant data including material information, mesh information, excitation, and output parameters from the host to device memory. Bundle the constant data to the texture memory to utilize the cache memory.

(2) Local curl operator:
   - Load fields into shared memory.
   - Synchronize with all the other threads of the block so that each thread can safely read shared memory locations that were populated by different threads.
   - Compute volume integral.

(3) Numerical flux and surface integrals:

- Load fields into shared memory.
- Synchronize with all the other threads of the block so that each thread can safely read shared memory locations that were populated by different threads.
- Calculate numerical flux and surface integrals.

(4) Field update of Runge-Kuttascheme:
- Update fields bytheLow storage Runge-Kutta scheme.
- Write the results back tothe device memory.

(5) Repeat (2) to (4).

3.1. Layout of Degree of Freedom

For the sake of convenience, we introduce the parameters used in the DGTD method. K is the number of tetrahedra in the domain. N_p is the number of degrees of freedoms (DOFs) in one element for each field component for a given order of Lagrange polynomial. M_k is the number of tetrahedra in one block, which depends on the interpolation order. $M_k \times N_p$ is taken to be a multiple of 32. N_{fp} is the number of DOFs on the face F_{ik} for each field component for a given order of Lagrange polynomial.

The memory layout is extremely important, which must match the kernel load patterns in order to achieve an efficient memory access. As show in Fig. 2, the pad memory (add some zeros to match the memory layout pattern) is used to keep the size of memory to be the multiple of 128. There are three strategies about the GPU performance optimization. The first one is maximal parallel execution to achieve maximal utilization; the second one is optimizing memory usage to achieve maximal memory throughout; and the third one is optimizing instructions usage to achieve maximal instruction throughout. The first and second strategies affect the DGTD performance. The number of threads per block should be a multiple of wrap size to avoid wasting computing resources.

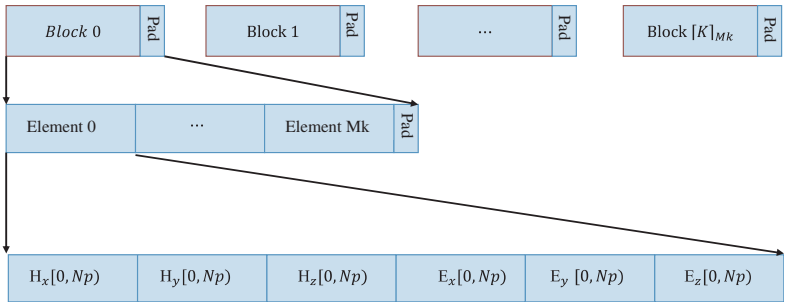


Figure 2. Global memory layout in the DGTD method.

3.2. Local-Element Curl Kernel

The differentiation kernel evaluates the curl operations, which is the product of three separate matrices whose results are combined to find the x-, y-, and z-derivatives on the reference elements. Each differentiation matrix has $N_p \times N_p$ entries and multiplies a vector of N_p entries of one field component. Toward this goal, the derivatives will still have to apply the reference derivatives to global differentiation coefficients for each element, there are total 3×3 coefficients to be loaded. One thread is normally used to handle one point in an element. For example, if we set the Lagrange polynomial order to be 2, there will be 10 DoFs of each field component. If using one element one block principle, there will be 22 threads wasted in a wrap. Because all the threads in one block are grouped to several wraps of 32 threads. So we should set the number of threads multiple of wrap size. Even the maximum threads in a block have 1,024 threads, it is not a good idea to use all 1,024 threads in one block. From authors' experience the threads per block can choose a number between 64 and 128.

The next strategy to improve performance is to use the share memory. Share memory is faster than global memory, but each block has only 64 k. In the differentiation operator, the three differentiation matrices need to be loaded, since the matrices are constants they can be stored in the share memory.

And each field component is used N_p times in the matrix-vector product. If we load them to the share memory by coalescing access, we also store them in the share memory. The thread block is given by $N_p \times M_K$. The thread index in the first dimension t_x reflects the number of elements in the block, and the second is the number of DOFs per elements. For the maximum usage, the index t_y equals to number of the rows in D_x, D_y, and D_z. As a result, we launch a grid of $[K]_{MK}$ thread blocks, which are indexed by b_x. During initialization, M_k is determined by evaluating the minimum execution time in the range M_k belong to the integer from 1 to 10. As K is a real number not multiple of M_k, we should append some elements.

3.3. Surfaces Kernel

If one thread works on all the points on one tetrahedral surface and we do not fuse the flux in the blocks, we have to compute numerical flux and surface integral using two kernels. This method results in efficient usage of threads but there will be at least $6K \times N_f \times N_{fp}$ (6 indicates 6 field components at each point) extra global memory to store the flux with extra memory latency. If fusing the flux in one block, since there are $N_f \times N_{fp} = N_p$ DOFs on one tetrahedral surface, it will cause thread waste. If we fuse the flux temporarily in the share memory, this will significantly reduce memory usage. If $N_f \times N_{fp}$ threads are used for one element, the total threads in one block is $M_k \times N_f \times N_{fp}$. The dominant simulation time of this kernel is spent on the computation of the numerical flux and surface integral.

3.4. Update Kernel in the Runge-Kutta Method

Compared to the local curl operator and flux calculation, the update kernel is relatively simple. It requires only local information and 2 Runge-Kutta floating coefficients located in register. Therefore, it is not necessary to use shared memory because the fields are used only once. The pseudo main routine of the DGTD code on GPU platform is described in listing 1.

Listing 1 Implementation of the DGTD method on the GPU platform.

```
void MaxwellsKernel3d_Grid2d(Mesh *mesh, float frka, float frkb, float fdt, int steps, int increment) {
    //Grab data from device and initiate sends
    MaxwellsMPISend3d(mesh);
    BlocksPerGrid = mesh->K;
    ThreadsPerBlock = p_Np;
    //Evaluate volume derivatives
    MaxwellsGPU_VOL_Kernel3D_MK_Grid_2D<<<Blocks,
            threads>>>(c_rhsQ, c_Q, c_vgeo, mesh->K);
    //Finalize sends and recvs, and transfer to device
    //for GPU parallel processing.
    MaxwellsMPIRecv3d(mesh, c_partQ);
    //Add plane excitation
    Blocks = dim3(GridDim_x, GridDim_y);
    threads = dim3(MKF, p_Nfp * p_Nfaces);
    MaxwellsGPU_SURF_Kernel3D_PlaneWave_MK_Grid_2d<<<Blocks,
        threads>>>(mesh->pulse_width, mesh->pulse_td,
        mesh->frequency, c_Q, c_rhsQ, c_idM_idP, mesh->time,
        mesh->dt, c_huygens, c_PEC, mesh->K);
    //Update Runge-Kutta Step
    ThreadsPerBlock = 1024;
    BlocksPerGrid = (Ntotal + ThreadsPerBlock −1) / ThreadsPerBlock;
    Blocks = dim3(GridDim_x, GridDim_y);
    MaxwellsGPU_RK_Kernel3D_grid_2d_res<<<Blocks,
        ThreadsPerBlock>>>(Ntotal, c_resQ, c_rhsQ, c_Q, frka, frkb, fdt);
    if (mesh->use_pml) {   //for PML boundary
        Blocks = dim3(GridDim_x, GridDim_y);
```

```
        threads = dim3(MK, p_Np);
        MaxwellsGPU_RK_Kernel3D_grid_2d_PML_res<<<Blocks,
            threads>>>(mesh->pml_el_id.size(), c_resQ, c_rhsQ, c_Q,
            c_pml_Q, c_pml_resQ, frka, frkb, fdt);
        //Update intermediate variablesQ and P in the PML region
        Blocks = dim3(GridDim_x, GridDim_y);
        threads = dim3(MK, p_Np);
        MaxwellsGPU_RK_Kernel3D_grid_2d_PML_Q<<<Blocks,
            threads>>>(mesh->pml_el_id.size(), c_resQ, c_rhsQ, c_Q,
            c_pml_Q, c_pml_resQ, frka, frkb, fdt);
    }
    Ntotal = mesh->K * BSIZE * p_Nfields;
    ThreadsPerBlock = 1024;
    BlocksPerGrid = (Ntotal + ThreadsPerBlock -1) / ThreadsPerBlock;
    Blocks = dim3(GridDim_x, GridDim_y);
    MaxwellsGPU_RK_Kernel3D_grid_2d_Q<<<Blocks,
        ThreadsPerBlock>>>(Ntotal, c_resQ, c_rhsQ,
        c_Q, frka, frkb, fdt);
}
```

4. NUMERICAL RESULTS

For the PEC objects, we can apply the plane wave excitation on the PEC surface directly and use the electric and magnetic surface currents to calculate the output parameters related to far zone fields. However, for the dielectric objects, we use the total/scattering field method that introduces a Huygens' surface (TF/SF surface) inside the computational domain in order to simply the plane wave excitation and far field calculation. To avoid the field interpolation on the Huygens' surface, we mesh the Huygens' surface when we mesh the objects and the computational domain. It is worth to mention that the Huygens' surface can be an arbitrary shape and at least two cells from the objects. For example, we can generate a conformal Huygens' surface for a dielectric coating structure based on the outside shape to simplify the computation. Generally speaking, there are several cells inside the white space, however, we need to keep a safe distance for the complex structures to ensure the PML convergent.

4.1. Benchmark Tests

The first benchmark is the RCS (radar cross section) prediction of a dielectric sphere, whose radius is 300 mm, and the relative permittivity is 4.0. A 4-layer UPML (uniaxial perfectly matched layer) is used to truncate the DGTD mesh. The guideline of the meshing technique is same as that in the frequency method, such as MoM and FEM. The minimum and maximum cell sizes are 24 mm and 100 mm, respectively, and the maximum cell size is inside the PML region. Total number of cells is 208,264 (not including the unknowns inside the PML region) and the total memory usage is 350 MB. The time step size is 0.0045 ns; the total number of time steps is 29,783; and the simulation time is 57 minutes on a NVIDIA GTX780M GPU on the host DELL Allienware laptop (Intel Xeon i-7 4990MQ, 2.8 GHz and 16 GB DDR3 memory). The plane wave excitation is added on the TF/SF surface, whose side length is 700 mm. The bistatic RCS of the dielectric sphere at 900 MHz is plotted in Fig. 3. For the sake of comparison, we use different methods such as finite integral technique (FIT) [28], MoM [29] and FEM [30] for the same problem, and the performance of each method is summarized in Table 1. The degree of freedoms (DoF) in the Table is multiply of number of cells and number of points in one element. The simulation time in the Table is on single core. In addition, we also include the simulation using GPU for the DGTD method in Table 1.

Next, we use the DGTD method to simulate a PEC almond, as shown in Fig. 4(a), which is

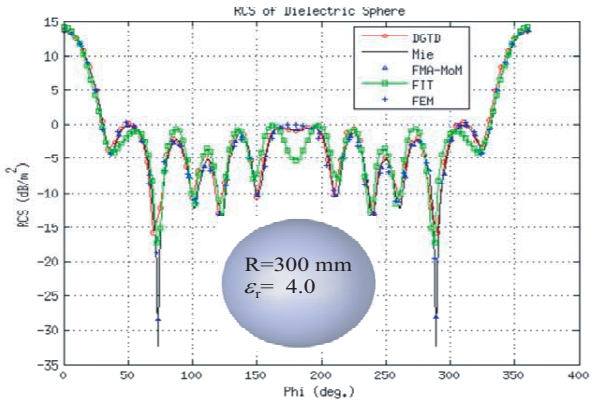


Figure 3. RCS pattern of the dielectric sphere with a radius of 300 mm.

Table 1. Simulation time and memory usage for dielectric sphere using different methods.

Option	FIT [28]	MoM [29]	FEM [30]	DGTD [This work]
Min cell size (mm)	4	47	12	50
Max cell size (mm)	8	207	700	250
Number of elements ($\times 10^3$)	4,574 (hexahedron)	16 (triangle) 151 (tetrahedron)	57 (tetrahedron)	111 (tetrahedron)
DoF ($\times 10^3$)	4,574	16	364	1,115
Δt (ns)	0.0077	N/A	N/A	0.0057
Total time steps	13,128	N/A	N/A	17,647
Memory (GB)	0.6	6.84	3.27	0.24
CPU Time (sec.)	2,026 (CPU)	691 (CPU)	916 (CPU)	782 (GPU) 15,163 (CPU)

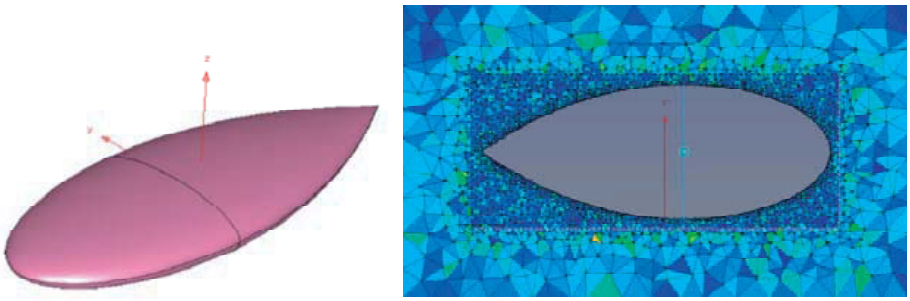
combined from one half ellipsoid and one half elliptic. The expression of the one half ellipsoid is:

$$\begin{aligned} x &= dt \\ y &= 0.193333f\left(\sqrt{1-\left({}^{t}\!/_{0.416667}\right)^2}\right)\cos(\varphi) \\ y &= 0.06444d\left(\sqrt{1-\left({}^{t}\!/_{0.416667}\right)^2}\right)\sin(\varphi) \end{aligned}$$

And the expression of the one half elliptic is:

$$\begin{aligned} x &= dt \\ y &= 4.83345d\left(\sqrt{1-\left({}^{t}\!/_{2.08335}\right)^2}\right)\cos(\varphi) \\ y &= 0.06444d\left(\sqrt{1-\left({}^{t}\!/_{2.08335}\right)^2}\right)\sin(\varphi) \end{aligned}$$

The almond dimensions are $2,499$ (length) $\times$ 966 (width) $\times$ 330 (height) mm. The dimension of computational domain is $3,600 \times 2,100 \times 1,430$, which is truncated by using one 4-layer UPML. The total number of elements is 235,305, and the minimum and maximum cell sizes are 30 mm and 230 mm, respectively. The mesh distribution on one cross section is shown in Fig. 4(b), and the TF/SF surface that is used to calculate the RCS patterns is the border of fine and coarse meshes in Fig. 4(b). The



(a) (b)

Figure 4. (a) Almond configuration. (b) Mesh distribution of the almond on one cross section. The fine and coast meshes are located on the two sides of the interface of TF/SF.

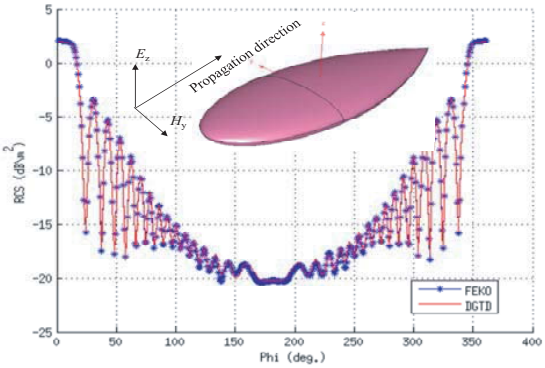


Figure 5. RCS of the almond at 1 GHz obtained by using the DGTD and MoM methods.

number of simulation time steps is 5,000, and the size of time step is 0.0053 ns on the NVIDIA GTX780M GPU. The RCS patterns of almond are plotted in Fig. 5. From these two benchmark tests, we obviously notice that the DGTD algorithm can give accurate and reliable results.

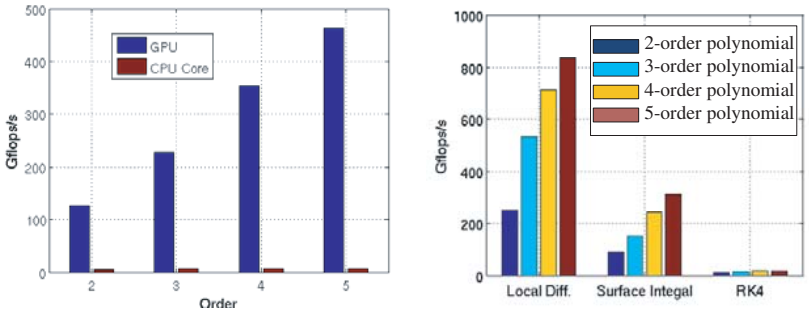
4.2. Performance Investigation

In this part, we investigate the performance of the DGTD method on the GPU platform. Here, the CPU code written in C language is compiled with GCC 4.47 and optimization option "−O3." To get an idea about the GPU performance, we use one CPU core as the reference when we show the GPU performance of the DGDT code. The test GPU platform includes a NVIDIA GeForce GTX780M mounted on the host with an Intel Xeon Core i7 CPU 49002.7 GHz. The test example is an empty cubic cavity with a side length of 2,000 mm.

For the sake of test, the polynomial order in the numerical simulations is taken to be 2, 3, 4, and 5, respectively. The polynomial order is the same for all the elements in the domain. The comparison of GPU and one CPU core for the DGTD simulation is shown in Fig. 6, which shows that the main computational kernel in the DGTD method is the volume and surface integrals. When the polynomial order is 5, the maximum computational performance is about 820 Gflops/s, which is almost the peak performance of the matrix-matrix calculation of CUBLAS on this GPU which is about 858.45 Gflops/s. The speedup of GPU is plotted in Fig. 7, which show that 24–75 times speed up can be achieved as the polynomial order increases.

4.3. Engineering Applications

The first application is an EM scattering problem from the PEC JAS-39 aircraft, as shown in Fig. 8(a). The dimensions of aircraft are $13,100$ (length) $\times\, 7,050$ (width) $\times\, 3,200$ (height) mm. The computational



(a) (b)

Figure 6. (a) Performance comparison between GPU and CPU. (b) GPU performance of different kernels in the DGTD method.

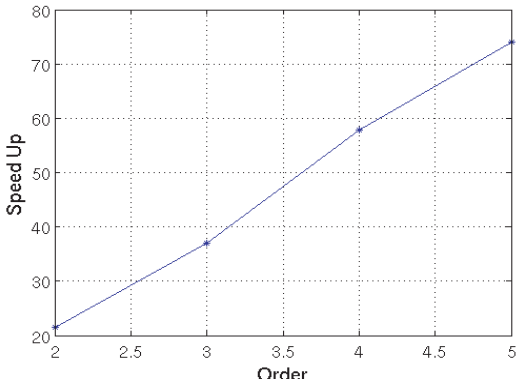
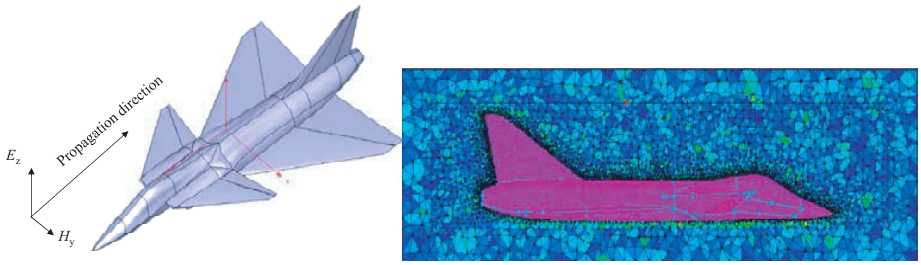


Figure 7. Speedup of GPU compared to one CPU core for the DGTD method.

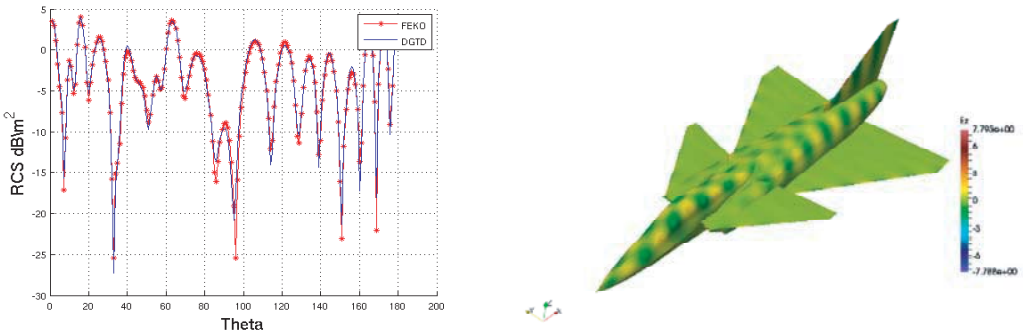


(a) (b)

Figure 8. (a) JAS-39 Aircraft model and the incident plane wave in the $-x$-direction with the E_z polarization. (b) Mesh distribution of the aircraft JAS-39 on one cross section. The fine and coast meshes are located on the two sides of the interface of TF/SF.

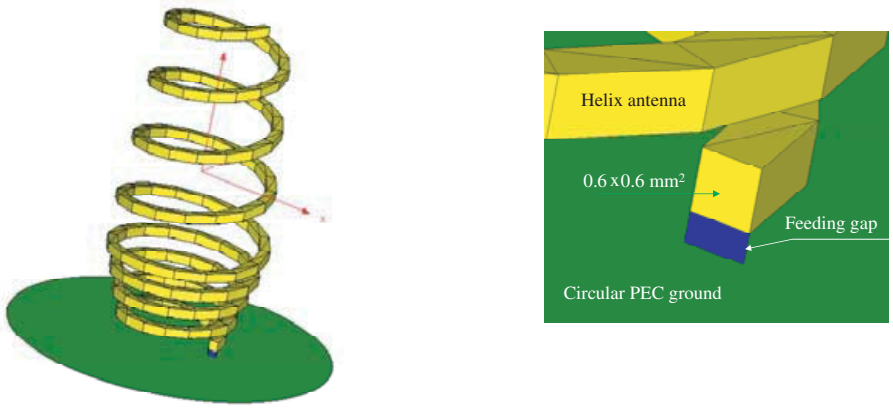
domain is $17,500 \times 10,800 \times 5,750$ mm, which is truncated by UPML that has a thickness of 1,000 mm. The total number of elements is 1,085 k, and the minimum and maximum cell sizes are 50 mm and 500 mm inside the PML region, respectively. The mesh distribution in one cross section is plotted in Fig. 8(b). The plane wave is incident along the x-direction with the E_z polarization. The time step size is 0.012 ns, and total simulation time is 57 minutes on the NVIDIA GTX780M GPU. The variation of RCS with angle θ at 300 MHz obtained by using the DGTD and MoM methods is plotted in Fig. 9(a), which shows good agreements in all directions. The surface distribution at 300 MHz on the aircraft is plotted in Fig. 9(b).

Next application case is a helix antenna mounted on an infinitely thin circular PEC plate with a radius of 10 mm. The cross section of PEC helix is $3 \times 3\,\text{mm}^2$, and the excitation is a voltage source



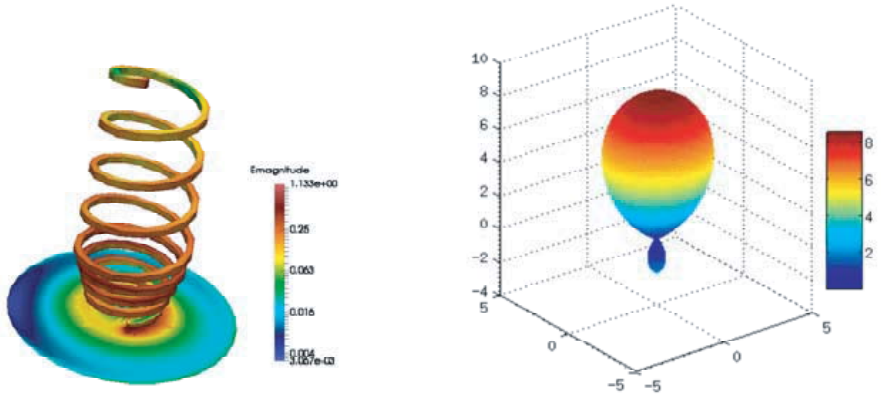
(a) (b)

Figure 9. (a) 2-D RCS pattern of the JAS-39 aircraft at 300 MHz using the DGTD and MoM methods. (b) Surface current distribution at 300 MHz on the JAS-39 aircraft.



(a) (b)

Figure 10. (a) Configuration of the helix antenna with a circular PEC ground (the radius is 10 mm). (b) The structure around the feeding area.



(a) (b)

Figure 11. (a) Surface current distribution at 15 GHz. (b) 3-D directivity pattern at 9 GHz.

located the gap between the PEC ground and helix, as shown in Fig. 10. The DGTD method is employed to simulate the helix antenna, the total number of mesh elements is 115 k and the minimum and maximum cell sizes are 1 mm on the helix and 7.8 mm near the first-order ABC. The TF/SF surface is used to calculate the directivity pattern. We can use the surface currents to calculate the far field parameters, however, the spherical TF/SF surface is used for the far field calculation. It is worthwhile

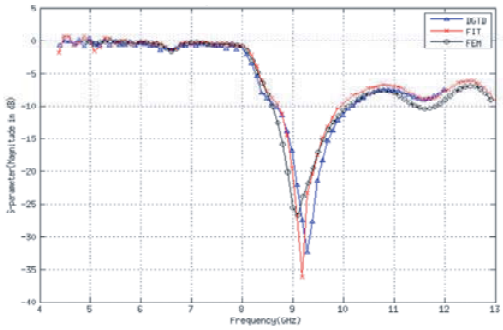
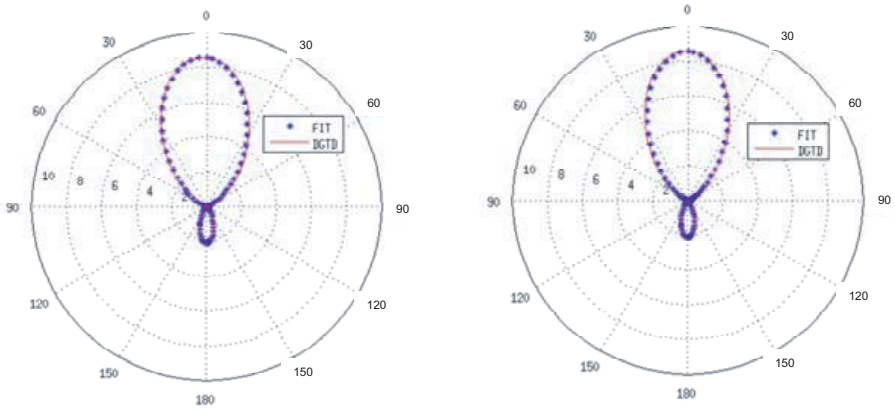


Figure 12. S-parameter of the helix antenna obtained by using different methods.



(a) (b)

Figure 13. 3-D directivity patterns of the helix antenna at $f = 9\,\text{GHz}$ in the $\phi = 0°$ and 90°. (a) plane and (b) 90° plane.

mentioning that the TF/SF surface can be an arbitrary shape. The time step size is 1.43×10^{-4} ns and the total number of time steps is 37,000, which took 26 minutes on the NVIDIA GTX780M GPU. The surface current at 15 GHz and the 3-D directivity pattern at 9 GHz are plotted in Fig. 11. The variation of S_{11} parameter with frequency is plotted in Fig. 12, and the results obtained by using FIT and FEM methods are also plotted in the same figure for comparison. The directivity patterns in the two major planes are plotted in Fig. 13 and the results obtained by using the FIT method is also plotted in the figure for the comparison.

5. CONCLUSIONS

The DGTD method has been successfully used to solve Maxwell's equations for a variety of EM problems with its advantages of explicit upgrade scheme, high parallel efficiency, wideband characteristic, and non-conformal mesh. Due to the unstructured tetrahedral meshes, the DGTD method has the advantages of both the FDTD and FEM methods. Since the DGTD method is parallel in nature, the good performance on the GPU platform has been demonstrated from the numerical results.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61372057, in part by the Open Project of State Key Laboratory of Millimeter Waves under Grant No. K201613.

REFERENCES

1. Jin, J., *The Finite Element Method in Electromagnetics*, 2nd edition, John Wiley & Sons, New York, NY, 2002.
2. Taflove, A. and S. Hagness, *Computational Electromagnetics: The Finite-Difference Time-Domain Method*, 3rd edition, Artech House, Norwwod, MA, 2005.
3. Peterson, A. and R. Mittra, *Computational Methods for Electromagnetics*, Wiley-IEEE Press, New Jersey, NJ, 1997.
4. Yu, W., R. Mittra, X. Yang, et al., *Parallel Finite Difference Time Domain Method*, Artech House, Norwood, MA, 2006.
5. Yu, W., Y. Rahmat-Samii, and A. Elsherbeni, *Advanced Computational Electromagnetic Methods and Applications*, Artech House, Norwood, MA, 2015.
6. Yee, K., "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Trans. Antennas Propag*, Vol. 14, 302–307, 1966.
7. Hesthaven, J. and T. Warburton, *Nodal Discontinuous Galerkin Methods, Algorithms, Analysis, and Applications*, Springer, New York, NY, 2008.
8. Chen, J. and Q. Liu, "Discontinuous Galerkin time-domain methods for multiscale electromagnetic simulations: A review," *Proceeding of The IEEE*, Vol. 101, No. 2, 242–254, 2013.
9. Jin, J., *From FETD to DGTD for Computational Electromagnetics*, ACES 2015 Tutorial, Williamsburg, VA, March 22–26, 2015.
10. Niegemann, J., *Introduction to Computational Electromagnetics The Discontinuous Galerkin Time-Domain (DGTD) Method*, Technical Report, Lab for Electromagnetic Fields and Microwave Electronics (IFH), ETH Zürich, 2012.
11. Busch, K., M. Konig, and J. Niegemann, "Discontinuous Galerkin methods in nanophotonics," *Laser Photonics Rev.*, Vol. 5, No. 6, 773–809, 2011.
12. Tobon, L. E., Q. Ren, Q. Sun, J. Chen, and Q. H. Liu, "New efficient implicit time integration method for DGTD applied to sequential multidomain and multiscale problems," *Progress In Electromagnetics Research*, Vol. 151, 1–8, 2015.
13. Yan, S. and J.-M. Jin, "Theoretical formulation of a time-domain finite element method for nonlinear magnetic problems in three dimensions (invited paper)," *Progress In Electromagnetics Research,* Vol. 153, 33–55, 2015.
14. Shankara, V., A. Mohammadiana, and W. Halla, "A time-domain, finite-volume treatment for the Maxwell equations," *Electromagnetics*, Vol. 10, No. 1, 127–145, 1990.
15. Karypis, G. and V. Kumar, "Parallel multilevel k-way partition scheme for irregular graphs," *SIAM Rev.*, Vol. 41, No. 2, 278–300, 1999.
16. Gödel, N., N. Nunn, T. Warburton, and M. Clemens, "Scalability of higher-order discontinuous Galerkin FEM computations for solving electromagnetic wave propagation problems on GPU clusters," *IEEE. Trans. Magn.*, Vol. 46, No. 8, 3469–3472, 2010.
17. Komatitsch, D., G. Erlebacher, D. Göddeke, and D. Michéa, "High-order finite-element seismic wave propagation modeling with MPI large GPU cluster," *J. Comput. Phys.*, Vol. 229, No. 20, 7692–7714, 2010.
18. D. Komatitsch, D. Göddeke, G. Erlebacher, and D. Michéa, "Modeling the propagation of elastic waves using spectral elements on a cluster of 192 GPUs," *Comput. Sci. Res. Dev.*, Vol. 25, 75–82, 2010.
19. Komatitsch, D., D. Michéa, and G. Erlebacher, "Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA," *J. Parallel Distrib. Comput.*, Vol. 69, 451–460, 2000.
20. Williamson, J., "Low-storage Runge-Kutta schemes," *Journal of Computational Physics*, Vol. 35, No. 1, 48–56, 1980.
21. Yang, X. and W. Yu, "PHI coprocessor acceleration techniques for computational electromagnetics methods," *ACES Journal*, Vol. 29, No. 12, 1013–1016, 2014.

22. Yu, W., X. Yang, and W. Li, *VALU, AVX and GPU Acceleration Techniques for Parallel FDTD Methods*, SciTech Publishing (An Imprint of the IET), Edison, NJ, 2014.
23. Shen, J., T. Tang, and L.-L. Wang, *Spectral Methods: Algorithms, Analysis and Applications*, Springer, 2011.
24. Van Der Vegt, J. and H. van der Ven, "Space-time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows: I. General formulation," *Journal of Computational Physics*, Vol. 182, No. 2, 546–585, 2002.
25. NVIDIA CUDA Parallel Programming and Computing Platform, http://www.nvidia.com/object/cuda_home_new.html.
26. NVIDIA, *CUDA C Programming Guide*, http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz3Yh95qkZB.
27. POINTWISE, http://www.pointwise.com/apps/.
28. https://www.cst.com/.
29. https://www.feko.info/.
30. http://www.ansys.com/Products/Electronics/ANSYS-HFSS.