Performance Evaluation of a Multi-GPU Enabled Finite Element Method for Computational Electromagnetics

Tristan Cabel, Joseph Charles, and Stéphane Lanteri

INRIA Sophia Antipolis-Méditerranée Research Center, NACHOS project-team
06902 Sophia Antipolis Cedex, France
Stephane.Lanteri@inria.fr

Abstract. We study the performance of a multi-GPU enabled numerical methodology for the simulation of electromagnetic wave propagation in complex domains and heterogeneous media. For this purpose, the system of time-domain Maxwell equations is discretized by a discontinuous finite element method which is formulated on an unstructured tetrahedral mesh and which relies on a high order interpolation of the electromagnetic field components within a mesh element. The resulting numerical methodology is adapted to parallel computing on a cluster of GPU acceleration cards by adopting a hybrid strategy which combines a coarse grain SPMD programming model for inter-GPU parallelization and a fine grain SIMD programming model for intra-GPU parallelization. The performance improvement resulting from this multiple-GPU algorithmic adaptation is demonstrated through three-dimensional simulations of the propagation of an electromagnetic wave in the head of a mobile phone user.

1 Introduction

Efforts to exploit GPUs, for non-graphical applications have been underway since 2003 and has evolved into programmable and massively parallel computational units with very high memory bandwidth. From this time to the present day a review of research works aiming at harnessing GPUs for the acceleration of scientific computing applications would hardly fit into one page. In particular, the development of GPU enabled high order numerical methods for the solution of partial differential equations is a rapidly growing field. Focusing on contributions that are dealing with wave propagation problems, GPUs have been considered for the first time for computational electromagnetics and computational geoseismics applications respectively by Klöckner et al. [3] and by Komatitsch et al. [5]-[4]. The present work shares several concerns with [3] which describes the development of a GPU enabled discontinuous Galerkin (DG) method formulated on an unstructured tetrahedral mesh for the discretization of hyperbolic systems of conservation laws. As it is the case with the DG method considered in [3], the approximation of the unknown field in a tetrahedron relies on a high order nodal interpolation method which is a key feature in view of exploiting the processing capabilities of a GPU architecture. A recent evolution of the work described in [3] is presented in Gödel et al. [2] where the authors discuss the adaptation of a multirate time stepping based DG method for solving the time-domain Maxwell equations on a multiple GPU system. Here, we study the performance of a multi-GPU enabled numerical methodology for the simulation of electromagnetic wave propagation.

2 The Physical Problem and Its Numerical Treatment

We consider the Maxwell equations in three space dimensions for heterogeneous linear isotropic media. The electric field $\boldsymbol{E}(\boldsymbol{x},t) = {}^t(E_x, E_y, E_z)$ and the magnetic field $\boldsymbol{H}(\boldsymbol{x},t) = {}^t(H_x, H_y, H_z)$ verify:

$$\epsilon\partial_t\boldsymbol{E} - \mathrm{curl}\boldsymbol{H} = -\boldsymbol{J} \quad , \quad \mu\partial_t\boldsymbol{H} + \mathrm{curl}\boldsymbol{E} = 0, \tag{1}$$

where the symbol ∂_t denotes a time derivative and $\boldsymbol{J}(\boldsymbol{x},t)$ is a current source term. These equations are set on a bounded polyhedral domain Ω of $\mathbb{R}^3$. The electric permittivity $\epsilon(\boldsymbol{x})$ and the magnetic permeability coefficients $\mu(\boldsymbol{x})$ are varying in space, time-invariant and both positive functions. The current source term $\boldsymbol{J}$ is the sum of the conductive current $\boldsymbol{J}_\sigma = \sigma\boldsymbol{E}$ (where $\sigma(\boldsymbol{x})$ denotes the electric conductivity of the media) and of an applied current $\boldsymbol{J}_s$ associated to a localized source for the incident electromagnetic field. Our goal is to solve system (1) in a domain Ω with boundary $\partial\Omega = \Gamma_a \cup \Gamma_m$, where we impose the following boundary conditions: $\boldsymbol{n} \times \boldsymbol{E} = 0$ on Γ_m, and $\mathcal{L}(\boldsymbol{E},\boldsymbol{H}) = \mathcal{L}(\boldsymbol{E}_{\mathrm{inc}},\boldsymbol{H}_{\mathrm{inc}})$ on Γ_a where $\mathcal{L}(\boldsymbol{E},\boldsymbol{H}) = \boldsymbol{n} \times \boldsymbol{E} - \sqrt{\frac{\mu}{\varepsilon}}\boldsymbol{n} \times (\boldsymbol{H} \times \boldsymbol{n})$. Here $\boldsymbol{n}$ denotes the unit outward normal to $\partial\Omega$ and $(\boldsymbol{E}_{\mathrm{inc}},\boldsymbol{H}_{\mathrm{inc}})$ is a given incident field. The first boundary condition is called *metallic* (referring to a perfectly conducting surface) while the second condition is called *absorbing* and takes here the form of the Silver-Müller condition which is a first order approximation of the exact absorbing boundary condition. This absorbing condition is applied on Γ_a which represents an artificial truncation of the computational domain.

For the numerical treatment of system (1), the domain Ω is triangulated into a set $\mathcal{T}_h$ of tetrahedra τ_i. We denote by $\mathcal{V}_i$ the set of indices of the elements which are neighbors of τ_i (i.e. sharing a face). In the following, to simplify the presentation, we set $\boldsymbol{J} = 0$. For a given partition $\mathcal{T}_h$, we seek approximate solutions to (1) in the finite element space $V_{p_i}(\mathcal{T}_h) = \{\boldsymbol{v} \in L^2(\Omega)^3 : \boldsymbol{v}_{|\tau_i} \in (\mathbb{P}_{p_i}[\tau_i])^3, \ \ \forall\tau_i \in \mathcal{T}_h\}$ where $\mathbb{P}_{p_i}[\tau_i]$ denotes the space of nodal polynomial functions of degree at most p_i inside τ_i. Following the discontinuous Galerkin approach, the electric and magnetic fields $(\mathbf{E}_i, \mathbf{H}_i)$ are locally approximated as combinations of linearly independent basis vector fields $\boldsymbol{\varphi}_{ij}$. Let $\mathcal{P}_i = \mathrm{span}(\boldsymbol{\varphi}_{ij}, \ 1 \leq j \leq d_i)$ where d_i denotes the number of degrees of freedom inside τ_i. The approximate fields $(\mathbf{E}_h, \mathbf{H}_h)$, defined by $(\forall i, \mathbf{E}_{h|\tau_i} = \mathbf{E}_i, \mathbf{H}_{h|\tau_i} = \mathbf{H}_i)$, are thus allowed to be completely discontinuous across element boundaries. For such a discontinuous field $\mathbf{U}_h$, we define its average $\{\mathbf{U}_h\}_{ik}$ through any internal interface a_{ik}, as $\{\mathbf{U}_h\}_{ik} = (\mathbf{U}_{i|a_{ik}} + \mathbf{U}_{k|a_{ik}})/2$. Because of this discontinuity, a global variational formulation cannot be obtained. However, dot-multiplying (1) by $\boldsymbol{\varphi} \in \mathcal{P}_i$, integrating over each single element τ_i and integrating by parts, yields a local weak formulation involving volume integrals over τ_i and surface integrals over $\partial\tau_i$. While the numerical treatment of volume integrals is rather straightfoward, a specific procedure must be introduced for the surface integrals, leading to the definition of a *numerical flux*. In this study, we choose to use a fully centered numerical flux, i.e., $\forall i, \forall k \in \mathcal{V}_i$, $\mathbf{E}_{|a_{ik}} \simeq \{\mathbf{E}_h\}_{ik}$, $\mathbf{H}_{|a_{ik}} \simeq \{\mathbf{H}_h\}_{ik}$. The local weak formulation can be written as:

$$\begin{aligned}
\int_{\tau_i} \boldsymbol{\varphi} \cdot \epsilon_i \partial_t \mathbf{E}_i =& \ \frac{1}{2}\int_{\tau_i} (\mathrm{curl}\boldsymbol{\varphi} \cdot \mathbf{H}_i + \mathrm{curl}\mathbf{H}_i \cdot \boldsymbol{\varphi}) - \frac{1}{2}\sum_{k \in \mathcal{V}_i} \int_{a_{ik}} \boldsymbol{\varphi} \cdot (\mathbf{H}_k \times \boldsymbol{n}_{ik}), \\
\int_{\tau_i} \boldsymbol{\varphi} \cdot \mu_i \partial_t \mathbf{H}_i =& -\frac{1}{2}\int_{\tau_i} (\mathrm{curl}\boldsymbol{\varphi} \cdot \mathbf{E}_i + \mathrm{curl}\mathbf{E}_i \cdot \boldsymbol{\varphi}) + \frac{1}{2}\sum_{k \in \mathcal{V}_i} \int_{a_{ik}} \boldsymbol{\varphi} \cdot (\mathbf{E}_k \times \boldsymbol{n}_{ik}).
\end{aligned} \tag{2}$$

Eq. (2) can be rewritten in terms of scalar unknowns. Inside each element, the fields are re-composed according to $\mathbf{E}_i = \sum_{1 \le j \le d} E_{ij}\boldsymbol{\varphi}_{ij}$ and $\mathbf{H}_i = \sum_{1 \le j \le d} H_{ij}\boldsymbol{\varphi}_{ij}$ and let us now denote by $\mathbf{E}_i$ and $\mathbf{H}_i$ respectively the column vectors $(E_{il})_{1 \le l \le d_i}$ and $(H_{il})_{1 \le l \le d_i}$. Then, (2) is equivalent to:

$$M_i^{\epsilon}\frac{d\mathbf{E}_i}{dt} = \ K_i\mathbf{H}_i - \sum_{k \in \mathcal{V}_i} S_{ik}\mathbf{H}_k, \quad M_i^{\mu}\frac{d\mathbf{H}_i}{dt} = -K_i\mathbf{E}_i + \sum_{k \in \mathcal{V}_i} S_{ik}\mathbf{E}_k, \tag{3}$$

where the symmetric positive definite mass matrices M_i^{η} (η stands for ϵ or μ), the symmetric stiffness matrix K_i (both of size $d_i \times d_i$) and the symmetric interface matrix S_{ik} (of size $d_i \times d_k$) are given by:

$$(M_i^{\eta})_{jl} = \eta_i \int_{\tau_i} {}^t\boldsymbol{\varphi}_{ij} \cdot \boldsymbol{\varphi}_{il}, \quad (S_{ik})_{jl} = \frac{1}{2}\int_{a_{ik}} {}^t\boldsymbol{\varphi}_{ij} \cdot (\boldsymbol{\varphi}_{kl} \times \boldsymbol{n}_{ik}).$$
$$(K_i)_{jl} = \frac{1}{2}\int_{\tau_i} {}^t\boldsymbol{\varphi}_{ij} \cdot \mathrm{curl}\boldsymbol{\varphi}_{il} + {}^t\boldsymbol{\varphi}_{il} \cdot \mathrm{curl}\boldsymbol{\varphi}_{ij}.$$

The set of local systems of ordinary differential equations for each τ_i (3) can be formally transformed in a global system. To this end, we suppose that all electric (resp. magnetic) unknowns are gathered in a column vector $\mathbb{E}$ (resp. $\mathbb{H}$) of size $d_g = \sum_{i=1}^{N_t} d_i$ where N_t stands for the number of elements in $\mathcal{T}_h$. Then system (3) can be rewritten as:

$$\mathbb{M}^{\epsilon}\frac{d\mathbb{E}}{dt} = \ \mathbb{K}\mathbb{H} - \mathbb{A}\mathbb{H} - \mathbb{B}\mathbb{H} + \mathbb{C}_E\mathbb{E} \ , \ \mathbb{M}^{\mu}\frac{d\mathbb{H}}{dt} = -\mathbb{K}\mathbb{E} + \mathbb{A}\mathbb{E} - \mathbb{B}\mathbb{E} + \mathbb{C}_H\mathbb{H}, \tag{4}$$

where we emphasize that $\mathbb{M}^{\epsilon}$ and $\mathbb{M}^{\mu}$ are $d_g \times d_g$ block diagonal matrices. if we set $\mathbb{S} = \mathbb{K} - \mathbb{A} - \mathbb{B}$ then system (4) rewrites as:

$$\mathbb{M}^{\epsilon}\frac{d\mathbb{E}}{dt} = \mathbb{S}\mathbb{H} + \mathbb{C}_E\mathbb{E} \ , \ \mathbb{M}^{\mu}\frac{d\mathbb{H}}{dt} = -{}^t\mathbb{S}\mathbb{E} + \mathbb{C}_H\mathbb{H}. \tag{5}$$

Finally, system (5) is time integrated using a second-order leap-frog scheme as:

$$\begin{cases} \mathbb{M}^{\epsilon}\left(\dfrac{\mathbb{E}^{n+1}-\mathbb{E}^{n}}{\Delta t}\right) &= \quad \mathbb{S}\mathbb{H}^{n+\frac{1}{2}} + \mathbb{C}_E\mathbb{E}^{n}, \\ \mathbb{M}^{\mu}\left(\dfrac{\mathbb{H}^{n+\frac{3}{2}}-\mathbb{H}^{n+\frac{1}{2}}}{\Delta t}\right) &= -\,{}^{t}\mathbb{S}\mathbb{E}^{n+1} + \mathbb{C}_H\mathbb{H}^{n+\frac{1}{2}}. \end{cases} \tag{6}$$

The resulting discontinuous Galerkin time domain method (DGTD-$\mathbb{P}_{p_i}$ in the sequel) is analyzed in [1] where it is shown that, when $\Gamma_a = \emptyset$, the method is stable under a CFL-like condition.

3 Implementation Aspects

3.1 DGTD CUDA Kernels

We describe here the implementation strategy adopted for the GT200 generation of NVIDIA GPUs and for calculations in single precision floating point arithmetic. We first note that the main computational kernels of the DGTD-$\mathbb{P}_{p_i}$ method considered in this study are the volume and surface integrals over τ_i and $\partial\tau_i$ appearing in (2). Moreover, we limit ourselves to a uniform order method i.e. $p \equiv p_i$ is the same for all the elements of the mesh, and we present experimental results for the values $p = 1, 2, 3, 4$. At the discrete level, these local computations translate into the matrix-vector products appearing in (3). The discrete equations for updating the electric and magnetic fields are composed of the same steps and only differ by the fields they are applied to. They both involve the same kernels that we will refer to in the sequel as `intVolume` (computation of volume integrals), `intSurface` (computation of surface integrals) and `updateField` (update of field components). All these kernels stick to the following paradigm: (1) load data from device memory to shared memory, (2) 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, (3) process the data in shared memory, (4) synchronize again to make sure that shared memory has been updated with the results, (5) write the results back to device memory. This paradigm ensures that almost all the operations on data allocated in global memory are performed in a coalesced way.

We outline below the main characteristics of these kernels and refer to [6] for a more detailed description. In our implementation, some useful elementary matrices, such as the mass matrix computed on the reference element, are stored in constant memory because they are small and are accessed following constant memory patterns. For the sequel, we introduce the following notations: NBTET is the number of tetrahedra that are treated by a block of threads. It depends of the chosen interpolation order and it is taken to be a multiple of 16 because of the way one load and write data to and from device memory; NDL is the number of degrees of freedom (d.o.f) in an element τ_i for each field component, for a given interpolation order; finally, NDF is the number of d.o.f on a face a_{ik} for each field component, for a given interpolation order.

Volume integral kernel : `intVolume`. This kernel operates on each d.o.f of a tetrahedron. Since the number of d.o.f increases with the interpolation order, resources needed by this kernel (registers and shared memory) also raise. Consequently, we wrote two versions of this kernel: one kernel for $p = 1$ and 2, and the other one for $p = 3$ and 4. However, these two versions have some common features. First, each thread computes one d.o.f of one tetrahedron. The second common feature is the data stored in shared memory, which are some geometrical quantities associated to a tetrahedron, and the field and the flux balance components. The last common feature is the number of tetrahedra operated by a block (i.e.NBTET). The main difference is that when in the low order version a block computes all the d.o.f (NDL) of the NBTET tetrahedra, the high order volume kernel only computes a certain number of d.o.f of the NBTET tetrahedra. Consequently, in the latter case, two or three instances of the kernel are necessary to compute all the d.o.f of all the tetrahedra. This approach induces a drawback because we have to load field data in two or three kernels instead of one. Indeed, the dimension of a block is NBTET*NDL which leads to blocks of more than 512 threads for high interpolation orders which is not possible in CUDA. However, there is also a benefit because computing a lower number of d.o.f in a kernel allows us to use less shared memory in the buffer storing field data and less registers in a kernel thus increasing the occupancy of the GPU.

Surface Integral kernel : `intSurface`. For this kernel, one thread works on one surface d.o.f of one tetrahedron. Similarly to the `intVolume` kernel, two versions of this kernel have been implemented. For the low order version, a thread applies the influence of its d.o.f to the four faces of its tetrahedron whereas for the high order version, a thread only works on one face of its tetrahedron. So, for the low order version, a block computes the numerical flux for four faces of NBTET tetrahedra instead of one face of NBTET tetrahedra for the high order version. Therefore, the high order version has to launch four kernels instead of one for the low order version. Here, we work on the surface d.o.f (NDF) but fields components are store using the volume d.o.f (NDL) so we need to use a permutation matrix to link these different local numberings of these d.o.f. Moreover, a face of a tetrahedron is also shared by another tetrahedron and the corresponding field values are needed in the computation of the elementary flux. Consequently, we cannot load field data in a coalesced way and we have to use texture memory. Field values are loaded before each face computation. Nevertheless, the high order version has a memory drawback compared to the lower one. Indeed, because there are four launches of the function, data are written four times to the flux table instead of once in the low order version.

Update kernel : `updateField`. There are four update kernels. First of all, update kernels are a bit different according to the field they are working on (electric or magnetic). Since in this case a thread works on one d.o.f of a tetrahedron, the dimension of a block is NBTET*NDL. Consequently, as for the `intVolume` kernel, we need a special version for the higher interpolation orders in order to avoid exceeding the maximum number of threads per block. In the high order version, we adopt an approach where a thread deals with two different d.o.f of a tetrahedron which allows a block to compute all the d.o.f for NBTET tetrahedra. This approach is less efficient for the lower interpolation orders. The two versions of the electric field update kernels need only one shared memory table. Indeed, in the first step, the flux computed by the previous kernels is loaded in this table, used to do some computations and then stored in a register. Therefore, the shared memory table is no longer used at the end of this part. In the second step, we load the previous values of the electric field in it in a coalesced way. In a third step, we update the value of the field in the shared memory, and in the last step, we write the new value of the field in the global memory. The update of the magnetic field follows the same pattern as the update of the electric field.

3.2 Multi-GPU Strategy

The multi-GPU parallelization strategy adopted in this study combines a coarse grain SPMD model based on a partitioning of the underlying tetrahedral mesh, with a fine grain SIMD model through the development of CUDA enabled DGTD kernels. A non-overlapping partitioning of the mesh is obtained using a graph partitioning tool such as MeTiS or Scotch and results in the definition of a set of sub-meshes. The interface between neighboring sub-meshes is a triangular surface. In the current implementation of this strategy, there is a one to one mapping between a sub-mesh and a GPU. Then the CUDA kernels described previously are applied at the sub-mesh level. The operations of the DGTD method are purely local except for the computation of the numerical flux for the approximation of the boundary integral over $\partial\tau_i$ in (2) which requires, for a given element, the values of the electromagnetic field components in the face-wise neighboring elements. For those faces which are located on an interface between neighboring sub-meshes, the availability of the electromagnetic field components on the attached elements is obtained thanks to point-to-point communications implemented using non-blocking MPI send and receive operations in order to overlap as much as possible communication operations by local computations of the volume integrals in (2). Moreover, we also overlap most of the PCI-express communications by using a CudaHostAlloc buffer which allows us to let the driver manage this CPU-GPU communication.

4 Performance Results

We first note that GPU timings (for all the performance results presented here and in the following subsections) are for single precision arithmetic computations and include the data structures copy operations from the CPU memory to the GPU device memory prior to the time stepping loop, and vice versa at the end of the time stepping loop. Numerical experiments have been performed on a hybrid CPU-GPU cluster with 1068 Intel CPU nodes and 48 Tesla S1070 GPU systems. Each Tesla S1070 has four GT200 GPUs and two PCI Express-2 buses. The Tesla systems are connected to BULL Novascale R422 E1 nodes with two quad-core Intel Xeon X5570 Nehalem processors operating at 2.93 GHz themselves connected by an InfiniBand network.

4.1 Weak Scalability

We first present results for the assessment of the weak scalability properties of the GPU enabled DGTD-$\mathbb{P}_p$ method. For that purpose, we consider a model test problem which consists in the propagation of a standing wave in a perfectly conducting unitary cubic cavity. For this simple geometry, we make use of regular uniform tetrahedral meshes respectively containing 3,072,000 elements for the DGTD-$\mathbb{P}_1$ and DGTD-$\mathbb{P}_2$ methods, 1,296,000 elements for the DGTD-$\mathbb{P}_3$ method and 750,000 elements for the DGTD-$\mathbb{P}_4$ method for the experiments involving one GPU. As usual in the context of a weak scalability analysis, the size of each mesh is increased proportionally to the number of computational entities. Moreover, since these meshes are regular discretizations of the cube, it is possible to construct perfectly balanced partitions and this is achieved here by constructing the tetrahedral meshes in parallel (i.e.on a sub domain basis) given a box-wise decomposition of the domain. Table 1 summarizes the measured timings measures for 1000 iterations of the leap-frog time scheme (6), and corresponding GFlops rates for 1 and 128 GPUs. These results illustrate an almost perfect weak scalability of the GPU enabled DGTD-$\mathbb{P}_p$ method with $p = 3$ and 4 for up to 128 GPUs. It also appears from these results that, for the proposed GPU implementation of the DGTD-$\mathbb{P}_p$ method and the hardware configuration considered in the above numerical experiments, the third-order scheme yields the best performance while, when increasing further the interpolation order, the sustained performance decrease due to bandwidth-bound effects.

4.2 Strong Scalability

We now consider a more realistic physical problem which corresponds to the simulation of the propagation of an electromagnetic wave in the head of mobile phone user. For this problem, compatible geometrical models of the head tissues have been constructed from magnetic resonance images. First, head tissues are segmented and surface triangulations of a selected number of tissues are obtained. In a second step, these triangulated surfaces together with a triangulation of the artificial boundary (absorbing boundary) of the overall computational domain are used as inputs for the generation of volume meshes. The exterior of the head must also be meshed, up to a certain distance and the

Table 1. Weak scalability assessment: timings and sustained performance figures

# GPU	DGTD-$\mathbb{P}_1$	DGTD-$\mathbb{P}_2$
1	104.7 sec/63 GFlops	325.1 sec/92 GFlops
128	104.9 sec/8072 GFlops	323.1 sec/11844 GFlops
# GPU	DGTD-$\mathbb{P}_3$	DGTD-$\mathbb{P}_4$
1	410.3 sec/106 GFlops	759.8 sec/94 GFlops
128	408.4 sec/13676 GFlops	763.6 sec/12009 GFlops

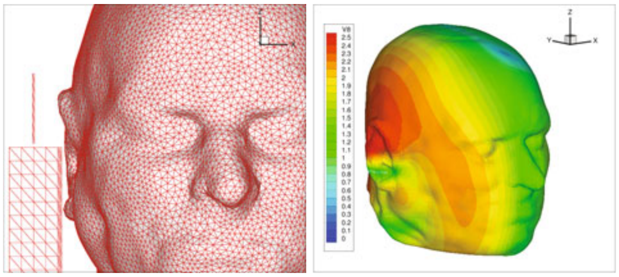

Fig. 1. Geometrical model of head tissues and computed contour lines of the amplitude of the electric field on the skin

Table 2. Characteristics of the fully unstructured tetrahedral meshes of head tissues

Mesh	# elements	L_{min} (mm)	L_{max} (mm)	L_{avg} (mm)
M1	815,405	1.00	28.14	10.69
M2	1,862,136	0.65	23.81	6.89
M3	7,894,172	0.77	22.75	3.21

Table 3. Head tissues exposure to an electromagnetic wave emitted by a mobile phone. Strong scalability assessment: mesh M1. Elapsed time on 16 CPUs: 715 sec (DGTD-$\mathbb{P}_1$ method) and 3824 sec (DGTD-$\mathbb{P}_2$ method).

# GPU	DGTD-$\mathbb{P}_1$			DGTD-$\mathbb{P}_2$		
	Time	GFlops	Speedup	Time	GFlops	Speedup
1	620 sec	32	-	2683 sec	60	-
16	35 sec	566	17.8	145 sec	1110	18.5

computational domain is artificially bounded by a sphere surface corresponding to the boundary Γ_a on which the Silver-Müller absorbing boundary condition is imposed. Moreover, a simplified mobile phone model (metallic box with a quarter-wave length mounted on the top surface) is included and placed in vertical position close to the right ear. The surface of this metallic box defines the boundary Γ_m. Overall, the geometrical models considered here consist of four tissues (skin, skull, CSF - Cerebro Spinal Fluid and brain). For the numerical experiments, we consider a sequence of three unstructured tetrahedral meshes whose characteristics are summarized in Table 2. The tetrahedral meshes are globally non-uniform and the quantities L_{min}, L_{max} and L_{avg} in Table 2 respectively denote the minimum, maximum and average lengths of mesh edges. Performance results are presented in Tables 3 to 5. For the coarsest mesh (i.e. mesh M1), the parallel speedup is evaluated for 16 GPUs relatively to the

Table 4. Head tissues exposure to an electromagnetic wave emitted by a mobile phone. Strong scalability assessment: mesh M2. Elapsed time on 64 CPUs: 519 sec (DGTD-$\mathbb{P}_1$ method) and 2869 sec (DGTD-$\mathbb{P}_2$ method).

# GPU	DGTD-$\mathbb{P}_1$			DGTD-$\mathbb{P}_2$		
	Time	GFlops	Speedup	Time	GFlops	Speedup
16	82 sec	699	-	407 sec	1137	-
32	46 sec	1239	1.8	201 sec	2299	2.0
64	33 sec	1747	2.5	116 sec	4007	3.5

Table 5. Head tissues exposure to an electromagnetic wave emitted by a mobile phone. Strong scalability assessment: mesh M3. Elapsed time on 64 CPUs: 2786 sec (DGTD-$\mathbb{P}_1$ method) and 6057 sec (DGTD-$\mathbb{P}_2$ method).

# GPU	DGTD-$\mathbb{P}_1$			DGTD-$\mathbb{P}_2$		
	Time	GFlops	Speedup	Time	GFlops	Speedup
32	162 sec	146	-	816 sec	2370	-
64	97 sec	2470	1.7	416 sec	4657	2.0
128	69 sec	3469	2.4	257 sec	7522	3.2

simulation time using one GPU. Although the number of elements of this mesh is well below the size of the mesh considered for the weak scalability analysis (i.e. 3,072,000 elements for the DGTD-$\mathbb{P}_1$ and DGTD-$\mathbb{P}_2$ methods), superlinear speedups are obtained. However, not surprisingly, the single GPU GFlops rates are lower than the corresponding ones reported in Table 1 (32 instead of 63 for the DGTD-$\mathbb{P}_1$ method, and 60 instead of 92 for the DGTD-$\mathbb{P}_2$ method). For the two other meshes (i.e. M2 and M3), as expected the DGTD-$\mathbb{P}_2$ method is always more scalable than the DGTD-$\mathbb{P}_1$ method because of a more favorable computation to communication ratio. Overall, acceleration factors ranging from 15 to 25 are observed between the multiple CPU and multiple GPU simulations. We note however that this comparison is made with a CPU version whose parallel implementation relies on MPI only. In particular, we have not considered a possible optimization to hybrid shared-memory multi-core systems combining the OpenMP and MPI programming models. Besides, an optimized CPU version in terms of simulation times can be obtained by computing the surface integrals over $\partial\tau_i$ in (2) through a loop over element faces and updating the flux balance of both elements τ_i and τ_j since the numerical flux between τ_j and τ_i is just the opposite of that from τ_i and τ_j. Such an optimization would lower the simulation times of the CPU version by approximately 30%. In the present implementation, each elementary numerical flux is computed twice (respectively for flux balances of τ_i and τ_j) for maximizing the floating point performance in the CUDA SIMD framework.

5 Conclusion

We have presented a high performance numerical methodology to simulate electromagnetic wave propagation in complex domains and heterogeneous media. This methodology is based on a high order discontinuous Galerkin time domain method formulated on unstructured tetrahedral meshes for solving the system of Maxwell equations. Due to its intrinsically local nature, this DGTD method is particularly well suited to distributed memory parallel computing. Besides, from the algorithmic point of view, the method mixes sparse linear algebra operations (as usual with classical finite element or finite volume methods) with dense linear algebra operations due to the use of a high order nodal interpolation method at the element level. Therefore, the method is an ideal candidate for exploiting the processing capabilities of GPU systems. In this work, this DGTD method has been adapted to multi-GPU parallel computing by combining a coarse grain SPMD programming model for inter-GPU parallelization and a fine grain SIMD programming model for intra-GPU parallelization. Numerical experiments presented in this paper clearly demonstrate the viability of the proposed parallelization strategy and open the route for further investigation especially in view of improving the GPU utilization as well as the overall scalability on systems consisting of several hundreds of GPU nodes.

Acknowledgments. This work was granted access to the HPC resources of CCRT under the allocation 2010-t2010065004 made by GENCI (Grand Equipement National de Calcul Intensif).

References

1. Fezoui, L., Lanteri, S., Lohrengel, S., Piperno, S.: Convergence and stability of a discontinuous Galerkin time-domain method for the 3D heterogeneous Maxwell equations on unstructured meshes. ESAIM: Math. Model. Num. Anal. 39(6), 1149–1176 (2005)
2. Gödel, N., Nunn, N., Warburton, T., Clemens, M.: Scalability of higher-order discontinuous Galerkin FEM computations for solving electromagnetic wave propagation problems on GPU clusters. IEEE. Trans. Magn. 46(8), 3469–3472 (2010)
3. Klöckner, A., Warburton, T., Bridge, J., Hesthaven, J.: Nodal discontinuous Galerkin methods on graphic processors. J. Comput. Phys. 228, 7863–7882 (2009)
4. Komatitsch, D., Erlebacher, G., Göddeke, D., Michéa, D.: High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster. J. Comput. Phys. 229(20), 7692–7714 (2010)
5. Komatitsch, D., Göddeke, D., Erlebacher, G., Michéa, D.: Modeling the propagation of elastic waves using spectral elements on a cluster of 192 GPUs. Comput. Sci. Res. Dev. 25, 75–82 (2010)
6. Cabel, T., Charles, J., Lanteri, S.: Multi-GPU acceleration of a DGTD method for modeling human exposure to electromagnetic waves. Tech. rep., INRIA Research eport RR-7592 (2011), `http://hal.inria.fr/inria-00583617`