Progress In Electromagnetics Research, Vol. 120, 263–292, 2011

PARALLEL IMPLEMENTATION OF A 3D SUBGRIDDING FDTD ALGORITHM FOR LARGE SIMULATIONS

A. Vaccari [1, *], A. Cala' Lesina [1], L. Cristoforetti [2], and R. Pontalti [1]

[1]FBK-irst, REET Department, Trento, Italy

[2]PAT, Department of Education, University and Research, Trento, Italy

Abstract—In a previous paper, we proposed and tested a robust and efficient three-dimensional (3-D) subgridding algorithm for the FDTD solution method of the Maxwell's curl PDEs system. Its characteristic feature is the *straight*, non-recursive, embedding of Yee grids — refined by factors of 3, 5, 7 and even larger — within coarser ones. There, the algorithm's implementation was described with the traditional serial programming approach. In the present paper, we propose and test its *parallel programming* implementation. The goal is to make it suitable and efficient for large scale electromagnetic simulations.

1. INTRODUCTION

The early Yee formulation [1] of the finite-difference time-domain (FDTD) method [2] and its successive enrichments, such as the pulsed FDTD method and the absorbing (ABC) or radiation (RBC) boundary conditions for appropriate mesh truncation, are a powerful tool in Computational ElectroMagnetics [3]. FDTD is a fully explicit numerical solution method of the hyperbolic first order Maxwell's curl PDEs system. All the spatial values of the electromagnetic field components are updated on the basis of their values at the previous time step. A frequency domain analysis is not precluded, because a semidiscrete Fourier Transform (DFT) can be performed in-line. Due to the large sizes of the models, a straight encoding of the FDTD method may result in long calculation time and thus demands the increase of the computational efficiency. The basic

Received 30 June 2011, Accepted 31 August 2011, Scheduled 6 September 2011

* Corresponding author: Alessandro Vaccari (vaccari@fbk.eu).

assumption is that increased mesh densities are introduced only in sub-regions where they are really needed (*mesh refinement* — or *subgridding* — algorithms) [4–7]. In a previous paper [8], we proposed an efficient, non-recursive, 3-D subgridding algorithm for the FDTD explicit method, incorporating a spatial filtering technique of the numerical signals. Our algorithm enables the straight embedding of finer meshes into coarse ones, which have larger space step by factors of 3, 5, 7 or greater, while maintaining fairly good accuracy and long term stability. Another step toward increased efficiency stands in the realm of the *High Performance Computing* (HPC) with the use of parallel or massive parallel machines which give access to larger RAM and storage resources. MPI realizes the *single-program multiple-data* (SPMD) programming approach: many instances of a single program, the *processes*, are executed *autonomously* on distinct physical processors of a cluster. This is a collection of interconnected SMP (Symmetrical MultiProcessor) *nodes*, each of them with a certain number of CPUs. Previous works on FDTD code parallelization by means of the MPI Library are reported in [9–13]. These papers describe 2-D or 3-D cartesian topologies of processes for the FDTD grid decomposition, along with user defined MPI data structures for data exchange at the subdomain boundaries. They also report some performance analysis. We start from Section 2, which gives details on how the MPI parallel version of our subgridding algorithm [8] is implemented, in particular it is described how to dynamically allocate the data structures in memory in order to have a fast access and an efficient management of the data transition at the coarse/coarse, refined/refined and coarse/refined domain interfaces. According to the intrinsic modularity of this algorithm, all the code is object-oriented C++ [14], which makes easier its understanding, development and maintenance. Our domain decomposition is through slices, i.e., 1-D. Section 3 analyzes quantitatively the performances of our parallel code, as applied to some FDTD test configurations involving the electromagnetic field human exposure. Section 4 deals with the application of the parallel code to a fairly large FDTD simulation. We assume cubic uniform grid cell shapes everywhere. The absorbing boundary conditions are those described in [15].

2. MPI PARALLELIZATION STRATEGY FOR THE SUBGRIDDING ALGORITHM

We start with a parallelepiped domain $\mathcal{D}$, the coarser one, sampled with a uniform *space step* δ and a corresponding *time step* $\tau = 0.5 \times \delta/c_o$ satisfying the Courant stability criterion [16–18], c_o being the vacuum

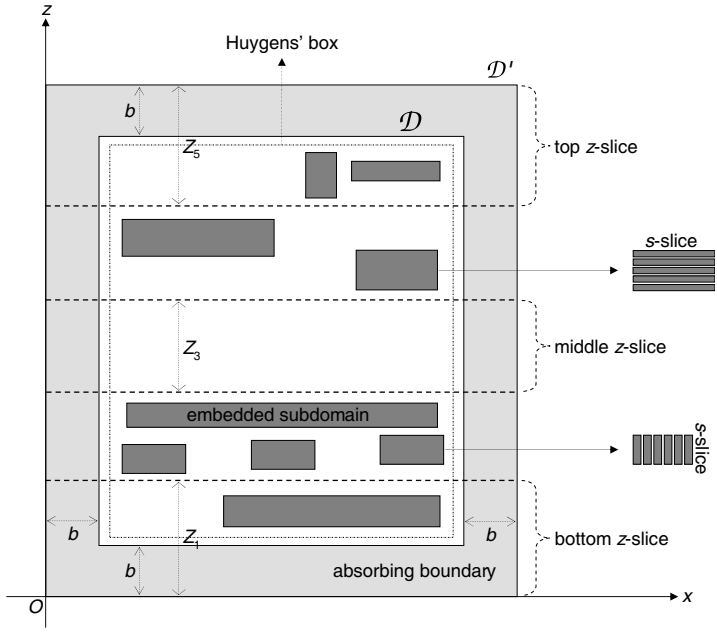

Figure 1. 2-D view example of a $\mathcal{D}'$ with $L = 5$. The $\ell = 3$ z-slice is empty.

light velocity. The resulting space lattice $\mathcal{D}$ consists of $N_x \times N_y \times N_z$ cubic Yee cells of edge size δ. $\mathcal{D}$ has to be augmented with extra cells of the same size to get a bigger $\mathcal{D}'$ parallelepiped lattice, in such a way that the set $\mathcal{D}' \setminus \mathcal{D}$ is an external shell used to accommodate the absorbing boundary conditions. We use a fixed number of $b = 20$ extra cells for each face of $\mathcal{D}$, so that the total memory cost for $\mathcal{D}'$ amounts to $(N_x + 2b) \times (N_y + 2b) \times (N_z + 2b)$ Yee cells. The $\mathcal{D}'$ domain is partitioned (see Fig. 1) into $L \geq 1$ subdomains, by "slicing" it along the z-axis through cutting planes parallel to the xy coordinate plane. Each one of the resulting L slices, referred to as a "z-slice", will have a thickness of given $Z_\ell > 0$ cells in the z direction ($1 \leq \ell \leq L$), such that:

$$Z_1 + Z_2 + \ldots + Z_L = N_z + 2b.$$

Into each z-slice with index ℓ ($1 \leq \ell \leq L$) a given number $M_\ell \geq 0$ of sublattices can be embedded. Any two of them do not intersect and all are strictly contained inside the z-slice intersection with $\mathcal{D}$. The various sublattices are characterized by a *mesh refinement factor* $R_{\ell m}$ ($1 \leq \ell \leq L$; $1 \leq m \leq M_\ell$). Their space and time steps are $\delta/R_{\ell m}$ and $\tau/R_{\ell m}$ respectively so that the Courant limit remains unchanged. $R_{\ell m}$

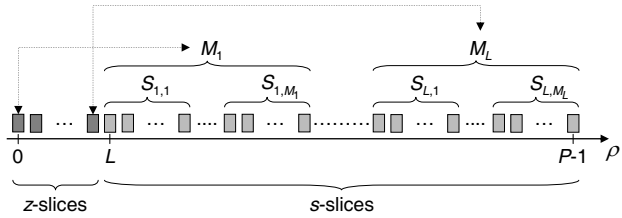

Figure 2. Scheme of the process rank ρ assignment for z-slices and s-slices.

are assumed to be odd integers: $3, 5, 7, \ldots$ without loss of generality. Every sublattice can be partitioned through cutting planes into $S_{\ell m} \geq 1$ slices (called "s-slice") along one, arbitrarily chosen, of the three coordinate directions (see Fig. 1). If the positive integers $T_{\ell mq}$ are the refined-cell thicknesses of the s-slices ($1 \leq q \leq S_{\ell m}$), they have to be chosen so that $R_{\ell m}$ is, for each q, a divisor of $T_{\ell mq}$ and consequently

$$C_{\ell m} = \sum_{q=1}^{S_{\ell m}} \frac{T_{\ell mq}}{R_{\ell m}}$$

gives the coarse-cell span, along the slicing direction, of the mth sublattice in the ℓth z-slice. Each slice, being it of the z-type or of the s-type, will be assigned to a single MPI process. The total number P of processes required is therefore given by (see Fig. 2):

$$P = L + \sum_{\ell=1}^{L} \sum_{m=1}^{M_\ell} S_{\ell m}.$$

Each process executes the usual FDTD "bulk" algorithm on its own lattice variables, inside a private local memory space. The overall space-grid continuity is achieved by *message passing* data at the various interfaces between the slices (cutting planes) and at the coarse/fine mesh interfaces. The task of data sharing is accomplished by means of MPI point-to-point communication routines. The FDTD bulk algorithm cannot advance to the next time step, until the data sharing in the whole space grid has been completed. In the MPI environment, the rank ρ of each of the P processes ($0 \leq \rho \leq P-1$) is assigned as follows: processes with $\rho = 0$ to $\rho = L-1$ accommodate the L z-slices; the process running the q' s-slice, of the m' sublattice, in the ℓ' z-slice ($1 \leq q' \leq S_{\ell' m'}$), has rank:

$$\rho = L + \sum_{\ell=1}^{\ell'-1} \sum_{m=1}^{M_\ell} S_{\ell m} + \sum_{m=1}^{m'-1} S_{\ell' m} + q'.$$

Processes could be grouped into $1 + \sum_{\ell=1}^{L} M_\ell$ disjoint subsets. There is "intra-communication" between their members, but no "inter-communication" between any two of the subsets. One of such subsets is formed from the first L processes running the various z-slices; the others are formed from the various $S_{\ell m}$ processes running all the s-slices of the mth generic sublattice in the ℓth generic z-slice. Inside each of the subsets, a process ρ communicates with its two neighbors $\rho-1$ and $\rho+1$ only. It must share the *tangential* electric (E) and magnetic (H) fields components of the Yee cells lying on its two boundary cutting planes: the upstream one, π_-, and the downstream one, π_+. More precisely, it *sends* H from π_- to $\rho-1$ and E from π_+ to $\rho+1$; it *receives* E in π_- from $\rho-1$ and H in π_+ from $\rho+1$ (see Fig. 3). To avoid deadlocks, this task is best accomplished by calling the Sendrecv MPI routine. If ρ implements a first or last s-slice, then *MPI::PROC_NULL* is passed to the routine, instead of $\rho-1$ or $\rho+1$ respectively. The subgridding algorithm [8] is now described through the following cyclic sequence of steps (E and H denote the fields in the coarse grid, e and h denote the fields in the refined grid):

(1) Storing of the current time step tangential coarse E field

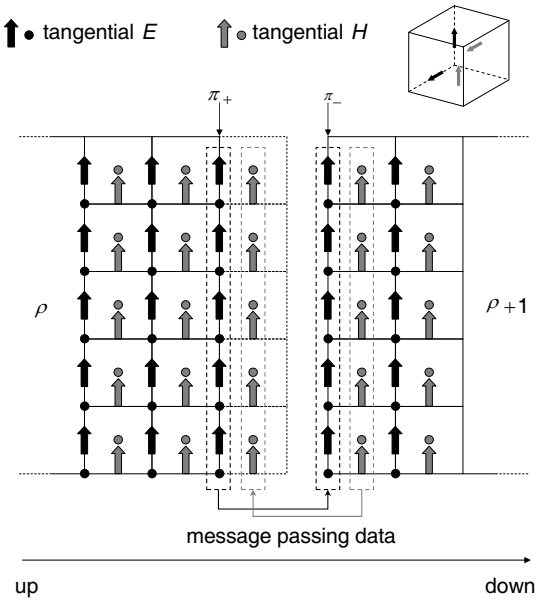

Figure 3. Data sharing at the cutting plane between two s-slices (2D view).

components.

(2) Updating of the coarse bulk E field values (using also the H values in step (10) below). Add excitations if any.

(3) Updating of the coarse E field Discrete Fourier Transforms.

(4) Updating of the coarse bulk H field values (normal components to the coarse/refined interfaces included). Add excitations if any.

(5) Updating of the interior refined e field values.

(6) Calculation of the refined tangential e field components by bilinear space interpolation (needed to fill in missing refined locations) and linear time inter/extra-polation (needed to get values at refined times). The valued stored in step (1) and those obtained in step (2) are used.

(7) Updating of the refined e field Discrete Fourier Transforms.

(8) Updating the refined h field values.

(9) Repeat steps from (5) to (8) R times (R being the refinement factor).

(10) Application of the spatial filtering technique to a subset (corresponding to the coarse grid sampling) of the refined tangential h field components and storing of the values.

(11) Repetition of steps from (1) to (10): an FDTD iteration.

Extra communication is then required at the coarse/fine grid interfaces (see Fig. 4). To this end, each z-slice manages the data sharing for all the s-slice interfaces of every sublattice it contains. In turn, a process ρ running a s-slice has to know the rank of the z-slice in which it is contained. Tangential E values on the interfaces have to be passed from the coarse side to the refined one and a space-time interpolation must be performed. In our parallelized implementation, the interpolation task is left to the process receiving the data, after the program has returned from the MPI *Recv* call. Moreover, the refined side has to pass *part* of its tangential H interface values, back to the coarse one. Such values have to undergo a "spatial" filtering procedure, according to formula (5) of [8], before they can be used in the FDTD bulk algorithm of the coarse side. Such task is left again to the refined side and then the program calls the MPI *Send* function. The functional scheme for the parallel version of the subgridding algorithm [8] is shown in Fig. 5 (labels 1, 2, $1'$, $2'$, A, B, A', B' in the figure will be referenced later on in Subsections 3.3 and 3.4).

Before the FDTD algorithm is entered, initial data for antenna structures or target complex permittivities are loaded from binary files. There is a data file for every sublattice. The loaded data are partitioned and dispatched to the appropriate s-slice processes by

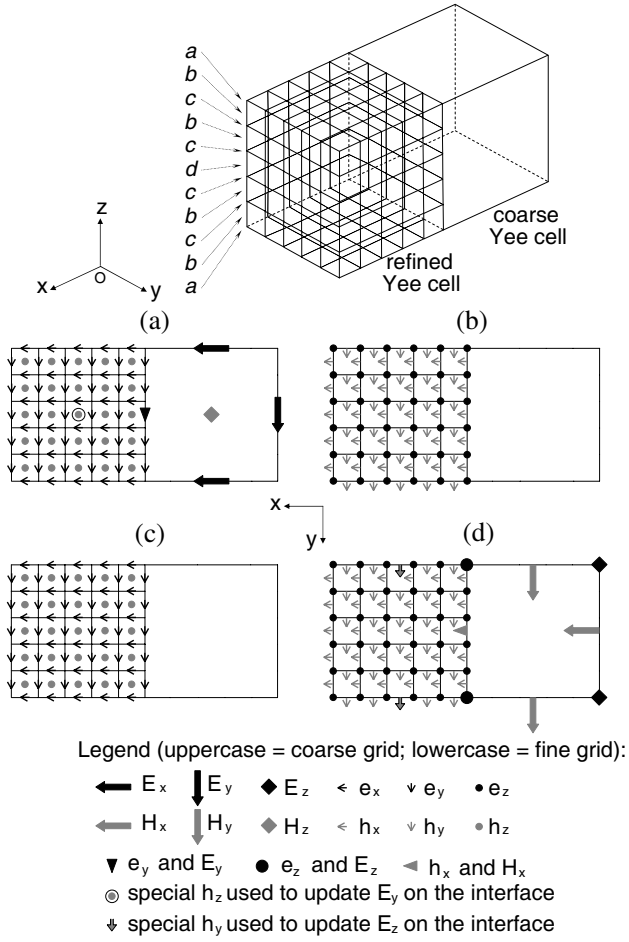

Figure 4. Fields layout at a coarse/fine grid interface in the yz-plane with a mesh refinement factor $R = 5$.

means of pointers. Additional initial data files are loaded, one for each z-slice: they contain parameters for the numbers of cells, numbers of sublattices, numbers of slices, placement of the sublattices, filtering order, and so on. At the end, the resulting data are collected from the various slices and packed, in such a way that there is a file of results for the $\mathcal{D}$ domain and each of its refined subdomains. As far as accuracy is concerned, we use the $\lambda_{\min}/10 \div \lambda_{\min}/20$ criterion [3], where $\lambda_{\min}$ is the shortest wavelength of interest in the response spectrum. This should assure an accuracy amounting to at least 5%. In early subgridding tests with experimental to analytical comparison, we found [8] an

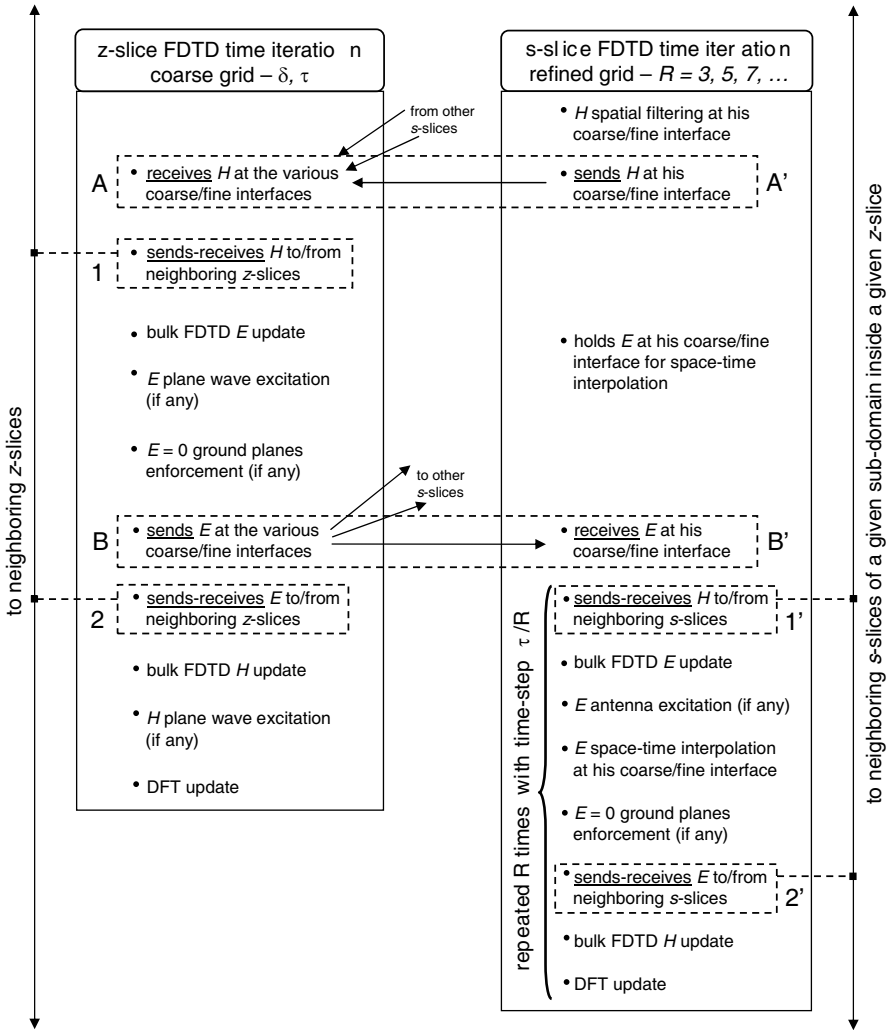

Figure 5. Parallel FDTD functional scheme for the subgridding algorithm [8].

accuracy of about 3% or better for canonical objects 20 space step apart from the outer boundary of the FDTD grid, where Higdon-Mur absorbing boundary conditions [19] were imposed. We also did not found any relevant effect of the coarse/refined transition boundaries, if an adequate spatial filtering order is used, provided that the stencil of the filtering procedure involves homogeneous Yee cells only. For

problems of electromagnetic field interaction with complex structures then, the subgridding should be applied only to those space regions where the local value of the relative permittivity reduces the local wavelength and thus the local space step value needed for a satisfactory accuracy.

2.1. Dynamic Memory Allocation and User Defined MPI Types for the Parallel Subgridding Algorithm

This subsection gives basic information about memory management specific of our MPI parallel implementation of our subgridding algorithm [8]. In the C++ encoding we use pointers and dynamic memory allocation to implement the data structures for the electric and magnetic field components lattice variables. In fact, no previous knowledge of the actual size of the model is available before compilation. Let then F be a generic field component and suppose we want for it a 3-D lattice of $(M+1)\times(N+1)\times(P+1)$ points along the x, y and z axes respectively. As has been seen in Section 2, data corresponding to the faces of parallelepipeds embedded in the F lattice have to be exchanged between pairs of processes at the coarse/fine grids interfaces. From the viewpoint of either the sender or receiver, the best way to do this is by *defining* new MPI types [20, 21], derived from the original *MPI::FLOAT* one. Such definitions require additional parameters, used by MPI to recognize regular patterns of subsets of F and to group and collect them out of F in a *Send* routine, or to fill in F appropriately with an incoming data flush in a *Recv* routine. In defining those new types, it is important to keep in mind the original ordering of the F array, that is to say the contiguity pattern of its memory locations (by appropriately changing the above lines of code, one could equally well allocate memory for F to accommodate different orderings).

• For a face parallel to the xy-plane, consisting of $m \times n$ lattice points and lower left vertex at I, J, K $(0 < I,\ J;\ I+m < M;\ J+n < N;\ 0 < k < P)$, one can create the derived type by invoking the method *Create_vector* with parameters values: $count = n$ blocks, each of $blocklength = m$ and with a $stride = M$ of old types between the blocks. The whole face can then be sent or received in a single call, starting at the address $\&F[K][J][I]$.

• For a face parallel to the xz-plane, consisting of $m \times p$ lattice points and lower left vertex at I, J, K $(0 < I,\ K;\ I+m < M;\ K+p < P;\ 0 < j < N)$, one can invoke again the *Create_vector* method, with parameters values: $count = p$ blocks, each of $blocklength = m$ and with a $stride = M \times N$ of old types between the blocks. The whole face can then be sent or received in a single call, starting at the address

$\&F[K][J][I]$.

• Data in a face parallel to the yz-plane, having $n \times p$ lattice points and lower left vertex at I, J, K ($0 < J,\ K;\ J + n < N;\ K + p < P;\ 0 < i < M$), could be sent in p chunks, each starting at the address $\&F[K + k][J][I]$, $0 \leq k \leq p$. Each chunk would correspond to an MPI derived type with $count = n$, $blocklength = 1$ and $stride = M$. The drawback of such a choice is that it requires p calls to the *Send* and *Recv* routines, with the bad side effect of amplifying the latency time by the same factor p. A better way is to create a derived MPI type by invoking the method *Create_indexed* [20, 21]. *Create_vector* does not work here, because the parameters must be constant integers. With *Create_indexed* one can specify an array of *blocklengths* and, instead of a constant *stride*, an array of corresponding *displacements*, in units of the old type from the starting location.

3. PARALLEL SUBGRIDDING ALGORITHM PERFORMANCE ANALYSIS

To analyze the parallel subgridding algorithm implementation performances, we defined three 3D FDTD test configurations, here conventionally referred to as T_1, T_2 and T_3.

• T_1 has an outer $\mathcal{D}'$ domain with a single z-slice ($L = 1$). $\mathcal{D} \subset \mathcal{D}'$ has $N_x = 200$, $N_y = 100$, $N_z = 100$ coarse Yee cells with a space step of 1.4 cm. It embeds a single subdomain ($M_1 = 1$) made of $875 \times 252 \times 147$ Yee cells with a space step of 0.2 cm., corresponding to a refinement factor $R_{1,1} = 7$. The latter models a standing human male body through the complex permittivity (dielectric constant in Farad/m and electric conductivity in Siemens/m) values, at a frequency of 2 GHz, of its anatomical structures. The human has its feet on a ground yz plane and is exposed to a x linearly polarized electromagnetic plane pulse of that frequency, impinging on it along the decreasing z direction. Such a model, without subgridding, was developed and used in [22]. The refined space step value has been chosen to get a good accuracy at the anatomical space scale for the given wavelength. If one uses this space step value for the entire model, there would be an increase by a factor of about 18 in the memory and simulation time requests.

• T_2 has the same outer $\mathcal{D}'$ domain than T_1. It embeds two subdomains ($M_1 = 2$). One modeling a commercial Radio Base Station antenna, made of $700 \times 154 \times 112$ Yee cells with a space step of 0.2 cm. The antenna operates at a frequency of 1862.5 MHz and is made of a double array (for a double polarization of $\pm 45°$) of radiating elements, each array consisting of 8 dipole pairs. Moreover, there is a metallic shield in the back. The other subdomain models the above

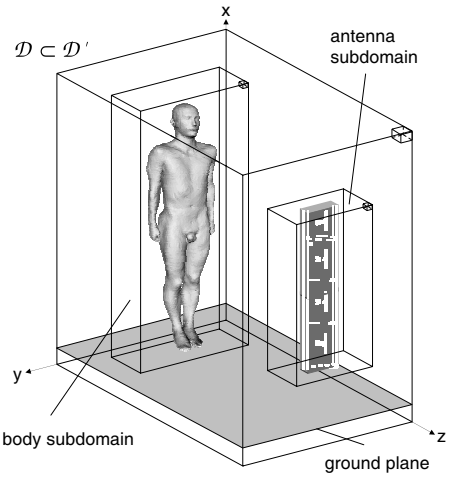

Figure 6. Schematic of the T_2-T_3 FDTD test configurations.

mentioned standing human male body on a grounded yz plane, made of $875 \times 252 \times 147$ Yee cells with a space step of 0.2 cm. Both sublattices have the same refinement factor: $R_{1,1} = R_{1,2} = 7$. T_2 increases the complexity of T_1 by adding an e.m. source-containing sublattice. Now the human body is not in a plane wave regime, but exposed to the antenna electromagnetic near field (see Fig. 6).

• T_3 is the same as T_2, but realized with two z-slices ($L = 2$): one for each of the sublattices ($M_1 = M_2 = 1$). From a serial programming point of view, there is no difference between T_2 and T_3. However, in parallel programming, T_3 adds complexity to T_2, because one more process is required in the $\mathcal{D}'$ modelization. T_3 has been introduced here to test the z-slice communication, which also affects the absorbing boundary conditions [15] implementation.

The parallel runs of T_1, T_2 and T_3 are made with variable s-slice numbers ($S_{\ell m}$) for the embedded sublattices. Also, various possible slicing directions (along the x or y or z axes) are taken into account. The code was compiled and executed on three types of systems, here conventionally referred to as WS, SP5 and BCX.

• WS is a dual processor Linux workstation, with two 3 GHz Xeon^{TM} EM64T Hyper-Threading Technology^{TM} CPUs, having a cache size of 1024 kB each and sharing a 6 GB memory space. Each CPU is treated by the operating system as two virtual processors. Installed are the GNU C++ ver. 3.4.6 compiler and the MPICH2 ver. 0.4 implementation [23] of the Message Passing API.

• SP5 is an IBM SP Cluster 1600, made of 64 SMP nodes

interconnected through a Federation High Performance Switch with a bandwidth of 2 GB/s. Each node contains 8 IBM Power5 1.9 GHz processors, for a total of 512 CPUs: 60 of the nodes have 16 GB of shared memory, the remaining 4 have 64 GB. Installed are the IBM AIX ver. 5.3 operating system and the IBM *xlC* C++ serial compiler. SP5 has the IBM Parallel Operating Environment (POE) and the LoadLeveler job management system to submit and schedule batch jobs. The SP5 peak performance amounts to 3.89 Tflop/s.

• BCX is an IBM BCX Cluster, made of 1280 SMP nodes, each containing 2 AMD Opteron^{TM} 2.6 GHz Dual Core processors (i.e., 4 cores per node) and with 8 GB of shared memory, for a total of 5120 cores. The nodes are interconnected through an Infiniband network with a bandwidth of 5 GB/s. Installed is the Intel C++ serial compiler *iCC*. The OpenMPI (not to be confused with OpenMP) implementation of the Message Passing API can be used [24]. The queuing system is LSF. The BCX peak performance amounts to 26.6 Tflop/s.

3.1. Profiling the Serial Version. OpenMP Loop-level Parallelism

The T_1 and T_2-T_3 test configurations ran for 1200 time iterations using the *serial* code version of the subgridding algorithm [8] on a single processor (remember that T_2 and T_3 coincide in this case). T_1 required about 1.8 GB of memory, while T_2-T_3 took about 2.4 GB of memory. The running times reported in Table 1 refer to the program calculation kernels only. In fact, times for initial data loading and final results output are negligible. A lot of time is spent (the top of list) in the space loop structures which update, in the *refined* grid, the: a) time domain values of the electric fields by the finite difference equations (44.66% of the total execution time); b) time domain values of the magnetic fields by the finite difference equations (25.41% of the total execution time); c) Discrete Fourier Transform values of the electric field (19.45% of the total execution time). The results obtained varying the number of threads are listed in Table 2. Inside each SMP node,

Table 1. Mean serial execution time for 1200 FDTD time iterations (sec).

	SP5	BCX	WS
T_1	3.0×10^4 ($\simeq$ 8.3 h)	4.9×10^4 ($\simeq$ 13.6 h)	3.7×10^4 ($\simeq$ 10.3 h)
T_2-T_3	4.0×10^4 ($\simeq$ 11.1 h)	6.3×10^4 ($\simeq$ 17.6 h)	4.6×10^4 ($\simeq$ 12.8 h)

Table 2. Mean SP5 OpenMP execution times for 1200 time iterations (sec).

Nr. of threads	T_1	T_2-T_3
2	1.5×10^4 ($\simeq$ 4.2 h)	2.1×10^4 ($\simeq$ 5.8 h)
3	1.1×10^4 ($\simeq$ 3.1 h)	1.8×10^4 ($\simeq$ 5.0 h)
4	9.8×10^3 ($\simeq$ 2.7 h)	1.3×10^4 ($\simeq$ 3.6 h)
5	8.8×10^3 ($\simeq$ 2.4 h)	1.1×10^4 ($\simeq$ 3.1 h)
6	8.0×10^3 ($\simeq$ 2.2 h)	9.1×10^3 ($\simeq$ 2.5 h)
7	6.6×10^3 ($\simeq$ 1.8 h)	8.7×10^3 ($\simeq$ 2.4 h)
8	5.9×10^3 ($\simeq$ 1.6 h)	8.4×10^3 ($\simeq$ 2.3 h)

the CPUs can *share* a common address space using the multi-threading OpenMP method [25, 26]. So one can specify more processes than the physical processors for tuning up the parallel code before running it on a bigger machine.

3.2. MPI Parallel Runs. Communication Overhead

The T_1, T_2 and T_3 test configurations ran for 1200 time iterations using the previously described *parallel* MPI implementation of the subgridding algorithm [8], with a varying number of processors and different sublattice decompositions, i.e., slicing directions. Actually, the sublattice decompositions here considered are a "longitudinal" one, made of s-slices which cut the subdomains through planes *perpendicular* to the body or antenna height (yz planes in Fig. 6, corresponding to a x slicing direction) and a "transversal" one, made of s-slices which cut the subdomains through planes *parallel* to the body or antenna height (xy planes in Fig. 6, corresponding to a z slicing direction). The former, hereafter referred to as the Longitudinal Slicing (LS, see left of Fig. 7), involves an interface exchange area between neighboring s-slices of 252×147 (human) or 154×112 (antenna) Yee cells. The latter, hereafter referred to as the Transversal Slicing (TS, see right of Fig. 7), involves an interface exchange area between neighboring s-slices of 875×252 (human) or 700×154 (antenna) Yee cells. As to the coarse cell s-slice thicknesses $\mathcal{T}_{\ell mq} = T_{\ell mq}/R_{\ell m}$, their values are all chosen to be about the same, within ± 1 or ± 2. This way, we are trying to avoid large imbalances in the processor loading. For example, in the TS slicing of the human body sublattice with $S_{\ell m} = 5$ s-slices, we alternate 3 s-slices of thickness 5 with 2 s-slices of thickness 3. This gives a total of $C_{\ell m} = 21$ coarse cells which amount,

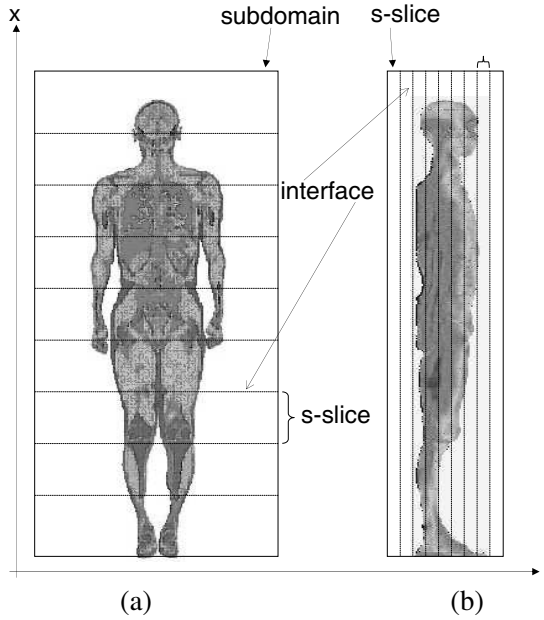

Figure 7. Schematic of the human male model (a) LS and (b) TS slicings.

for a refinement factor of $R_{\ell m} = 7$, to the given 147 refined cells along the z-axis. In the human body LS slicing with $S_{\ell m} = 35$ s-slices, we shuffle 20 s-slices of thickness 4 with 15 s-slices of thickness 3, for a total of $C_{\ell m} = 125$ coarse cells which amount, for a refinement factor of $R_{\ell m} = 7$, to the given 875 refined cells along the x-axis. There is, however, a lower limit for the s-slice thickness. A thickness $\mathcal{T}_{\ell mq} < 2$ ($1 \leq q \leq S_{\ell m}$, for fixed ℓ and m) would make unapplicable the spatial filtering procedure [8] at the coarse/fine interface. As can be seen from Fig. 8 the T_1 configuration has been tested for a wide range of s-slice numbers and for both LS and TS decompositions. A number of s-slices equal to 0 means *serial* execution: the values are taken from Table 1. Parallel MPI executions start with a number of s-slices at least equal to 1, which means a single embedding coarse FDTD lattice — the z-slice — and an embedded *uncut* refined FDTD sublattice, i.e., two processes which communicate between them at the coarse/fine interface. One sees LS is faster than TS. This is due to the bigger amount of data passed between processes in TS slicing. From Fig. 8, it is also apparent the saturation effect due to the communication overhead with increasingly P. In Fig. 9 the T_2 and T_3 models are

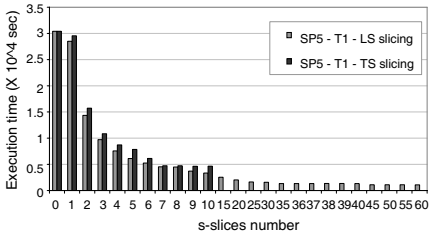

Figure 8. LS-TS comparison for the T_1 test model on SP5.

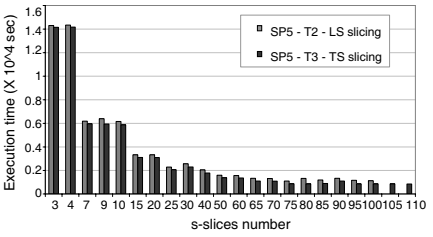

Figure 9. T_2-T_3 test models comparison for LS on SP5.

compared for LS only. They are more complex than T_1, including the extra antenna subdomain. For T_2, P is given by the number read on the horizontal axis plus *one* and for T_3, P is given by the horizontal axis value plus *two*. This explains the faster behavior of the T_3 model compared with that of T_2. Both Figs. 8 and 9 evidence fluctuations due to the non-exclusive access to the computing machine.

3.3. Scalability Analysis

This Subsection deals with the analysis of the expected performances of the proposed MPI parallel implementation of the subgridding algorithm [8] and their comparison with the numerical experiments in the previously two Subsections. Let Ω be the *input size* [20] of the FDTD model, which in our case is given by the pair:

$$\Omega = (\Omega_c\,, \Omega_s) = \left(\Omega_c\,, \sum_{i=1}^{\mathcal{N}} \Omega_{s,i}\right)$$

where Ω_c refers to the outer embedding coarse grid size and $\Omega_{s,i}$ to the ith sublattice refined grid size with $\mathcal{N} \geq 1$ ($\mathcal{N} = \sum_{\ell=1}^{L} M_\ell$) the total sublattice number inside it. Then:

$$\Omega_c = N'_x \times N'_y \times N'_z$$

where primed quantities include the extra $20 + 20$ cells needed for the absorbing boundary conditions [15], and:

$$\Omega_{s,i} = n_{x,i} \times n_{y,i} \times n_{z,i}$$

where $n_{\rho,i}$ are refined sizes ($\rho = x, y, z$ and $i = 1, \ldots, \mathcal{N}$). Let $T_\sigma(\Omega)$ and $T_\pi(\Omega, P)$ be the serial and parallel running times respectively (the latter referring to the slowest of $P \geq 2$ processes on separate physical processors) of the subgridding algorithm [8]. Important parameters of

interest to measure performances ([20, 21, 27, 28]) are the *speedup* $\mathcal{S}$:

$$\mathcal{S}(\Omega, P) \equiv \frac{T_\sigma(\Omega)}{T_\pi(\Omega, P)} \tag{1}$$

and *efficiency* of process utilization $\mathcal{E}$:

$$\mathcal{E}(\Omega, P) \equiv \frac{\mathcal{S}(\Omega, P)}{P} = \frac{T_\sigma(\Omega)}{P \times T_\pi(\Omega, P)}. \tag{2}$$

Taking into account message-passing communication overhead but, as already pointed out, discarding any input/output or initialization code time contributions, we have [20]:

$$T_\sigma(\Omega) = T_\sigma^{calc}(\Omega) \tag{3}$$

and

$$T_\pi(\Omega, P) = T_\pi^{calc}(\Omega, P) + T_\pi^{comm}(\Omega, P)\,. \tag{4}$$

Moving data among processes is a task which is a priori more expensive than local calculation times. In fact, the cost of a single *Send/Recv* pair with α *floats* of data transmitted may be written as [20, 21]:

$$\tau_{la} + \alpha \times \tau_{bw} \tag{5}$$

where τ_{la} is the *latency time* (a start-up time needed to initiate a message) of the MPI communication system, while τ_{bw} depends inversely on the communication *bandwidth* and its units in (5) are $sec/float$. Commonly one has: $\tau_{la} \gg \tau_{bw} \gtrsim \tau_{fp}$, where τ_{fp} is the typical cost of an arithmetic floating point operation. It is also useful to introduce a P processes *total overhead* T_o with respect to ideal parallelism:

$$T_o(\Omega, P) \equiv T_\pi(\Omega, P) - \frac{T_\sigma(\Omega)}{P}\,. \tag{6}$$

Through T_o, (2) can be rewritten as:

$$\mathcal{E}(\Omega, P) = \left[1 + P\frac{T_o(\Omega, P)}{T_\sigma(\Omega)}\right]^{-1}\,. \tag{7}$$

According to [20] we will then say that a parallel program is *scalable* if it, as Ω and P vary, admits a suitable $T_o(\Omega, P)$ adjustment that keeps efficiency $\mathcal{E}$ constant and as close to 1 as possible. The complexity of the bulk FDTD algorithm plus the electric field DFT updating code is $O(N^3)$, where N is a typical grid linear size. Since 39 or 42 (single precision) floating point operations (sums and multiplications) are required per FDTD time iteration per vacuum and dielectric/metallic

cell respectively. We have for an average estimate of T_σ in (3) (accounting for about the 94% of the total computation time):

$$T_\sigma(\Omega) \equiv T_\sigma^{calc}(\Omega) \simeq \left[42 \sum_{i=1}^{\mathcal{N}} R_i \, \Omega_{s,i} + 39 \, \Omega_c\right] \tau_{fp} \, , \tag{8}$$

where R_i is the refinement factor of the ith sublattice (for each coarse FDTD time iteration, there are R_i FDTD time iterations inside the ith sublattice). Turning now to T_π in (4) and differentiating between coarse grid calculations (c extra subscript) and refined grids (i.e., sublattices) calculations (s extra subscript), for a parallel code running with P processes ($P \geq \mathcal{N} + L$) we can write:

$$T_{\pi,c}^{calc}(\Omega, P) \simeq \frac{39 \, \Omega_c \, \tau_{fp}}{L} \tag{9}$$

and

$$T_{\pi,s}^{calc}(\Omega, P) \simeq \frac{1}{P - L} \left[42 \sum_{i=1}^{\mathcal{N}} R_i \, \Omega_{s,i}\right] \tau_{fp} \tag{10}$$

respectively. By introducing the average number Γ of s-slices per sublattice:

$$\Gamma \simeq \frac{P - L}{\mathcal{N}} \, ,$$

(L will depend on P through $\mathcal{N}$ and Γ), we deduce the following three contributions for the communication time overhead per FDTD iteration. The first one:

$$T_{\pi,c}^{comm}(\Omega, P) = 2 \times 4 \, (L - 1) \left(\tau_{la} + N'_x \, N'_y \, \tau_{bw}\right) \tag{11}$$

is for the coarse grid computations (labels 1 and 2 in the left column of Fig. 5). The second one:

$$T_{\pi,s}^{comm}(\Omega, P) = 2 \times 4 \, (\Gamma - 1) \sum_{i=1}^{\mathcal{N}} R_i \, (\tau_{la} + n_{\alpha,i} \, n_{\beta,i} \, \tau_{bw}) \tag{12}$$

for the refined grid computations (labels $1'$ and $2'$ in the right column of Fig. 5). The third contribution is:

$$T_{\pi,both}^{comm}(\Omega, P) = 6 \left[\mathcal{N} \, \Gamma \, \tau_{la} + \sum_{i=1}^{\mathcal{N}} \left(\frac{n_{x,i} \, n_{y,i} + n_{x,i} \, n_{z,i} + n_{y,i} \, n_{z,i}}{R_i^2}\right) \tau_{bw}\right] \tag{13}$$

and affects both the coarse and refined grid computations (labels AA' and BB' in Fig. 5). As previously pointed out, these data are passed efficiently by means of suitable data templates. We thus have:

$$T_{\pi,c}(\Omega, P) = T_{\pi,c}^{calc}(\Omega, P) + T_{\pi,c}^{comm}(\Omega, P) + T_{\pi,both}^{comm}(\Omega, P)$$

and

$$T_{\pi,s}(\Omega,P) = T^{calc}_{\pi,s}(\Omega,P) + T^{comm}_{\pi,s}(\Omega,P) + T^{comm}_{\pi,both}(\Omega,P)\,.$$

We now formulate the general criterion which would give our parallel subgridding algorithm optimal scalability:

$$\boxed{T_{\pi,c}(\Omega,P) \overset{(1)}{=} T_{\pi,s}(\Omega,P) \overset{(2)}{=} \frac{T_\sigma(\Omega)}{P\,\mathcal{E}_t}} \tag{14}$$

where $\mathcal{E}_t$ is a target value (necessarily $\mathcal{E}_t < 1$), for the attainable efficiency $\mathcal{E}(\Omega,P)$ (2) or (7). The first equation of system (14) will make all of the P processes (of both kind, those running z-slice calculations and those running s-slice calculations) to take, on average, an equal running time. The second equation in (14), on the other hand, should reduce the overhead $T_o(\Omega,P)$ — as defined in (6) — according to the chosen value $\mathcal{E}_t$ for $\mathcal{E}(\Omega,P)$. Table 3 summarizes the values for the size parameters just introduced. From them one can easily deduce that (8) evaluates to:

$$T_\sigma(\Omega) \simeq 9.71 \times 10^{09} \times \tau_{fp} \tag{15}$$

for the T_1 test model. For the T_2-T_3 test model it evaluates to:

$$T_\sigma(\Omega) \simeq 1.33 \times 10^{10} \times \tau_{fp}\,. \tag{16}$$

Comparing with the results of Table 1, which hold for 1200 FDTD time iterations, empirical estimates for τ_{fp} are obtained. We also got values for τ_{la} and τ_{bw} by looking at elapsed times when running ad hoc MPI pieces of code. The values found are: We then proceed to extrapolate, accordingly to the above analysis, the expected performances of the algorithm proposed by applying it to the three general test configurations T_1, T_2 and T_3 previously mentioned. We point out here that, with the values in Table 3 and Table 4, we are not able to satisfy the first constraint in the general criterion (14) and always get:

$$T_{\pi,s}(\Omega,P) > T_{\pi,c}(\Omega,P)\,,$$

meaning the coarse embedding grid code runs faster than that for the refined grids, with a limiting parallel execution time, per FDTD iteration, given by the middle member $T_{\pi,s}$ of (14). For the T_1, T_2 cases it has the general form:

$$T_\pi(\Omega,P) \equiv T_{\pi,s}(\Omega,P) = \frac{a}{P-1} + b\,(P-1) + c\,, \tag{17}$$

while for the T_3 case it has the general form:

$$T_\pi(\Omega,P) \equiv T_{\pi,s}(\Omega,P) = \frac{a}{P-2} + b\,(P-2) + c\,. \tag{18}$$

Table 3. Size parameter values for the paper test models.

Size Param.	T_1	T_2	T_3
N'_x	$20 + 200 + 20$	$20 + 200 + 20$	$20 + 200 + 20$
N'_y	$20 + 100 + 20$	$20 + 100 + 20$	$20 + 100 + 20$
L	1	1	2
N'_z	$20 + 100 + 20$	$20 + 100 + 20$	$(20 + 50) + (50 + 20)$
$\mathcal{N}$	1	2	2
$n_{x,1}$	875	875	875
$n_{y,1}$	252	252	252
$n_{z,1}$	147	147	147
R_1	7	7	7
$n_{x,2}$	–	700	700
$n_{y,2}$	–	154	154
$n_{z,2}$	–	112	112
R_2	–	7	7

Table 4. Floating point, latency and bandwidth time.

	τ_{fp}	τ_{la}	τ_{bw}
SP5	$\simeq 2.5$ nsec	$\simeq 40$ µsec	$\simeq 2$ nsec/float
BCX	$\simeq 4.2$ nsec	$\simeq 200$ µsec	$\simeq 4$ nsec/float
WS	$\simeq 3.0$ nsec	–	–

where a, b, c are coefficients depending on the model size, on τ_{fb}, τ_{la} and τ_{bw}. Asymptotically both these two times grow linearly with P, thus indicating that the communication overhead ends up overwhelming the speedup achievable by reducing further and further the s-slice bulk sizes.

T_1 test model: here we have $\mathcal{N} = 1$ and $L = 1$ (thus implying $\Gamma = P - 1$), with $P \geq 2$. The coefficients in (17) amount to:

$$\begin{aligned} a &= 9.53 \times 10^9 \times \tau_{fp} \text{ sec.} \\ b &= (56 \times \tau_{la} + 2074464 \times \tau_{bw}) \text{ sec.} \\ c &= (-56 \times \tau_{la} + 10242 \times \tau_{bw}) \text{ sec.} \end{aligned}$$

By means of (15) and (17), expressions for the efficiency $\mathcal{E}(\Omega, P)$ and the speedup $\mathcal{S}(\Omega, P) = P\,\mathcal{E}(\Omega, P)$ can be calculated as functions of P, starting from their definitions (1), (2). Graphs of $T_\pi(\Omega, P)$, efficiency

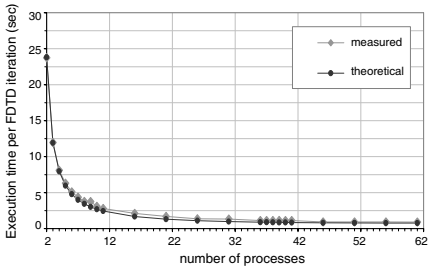

Figure 10. T_1 test model timing on SP5.

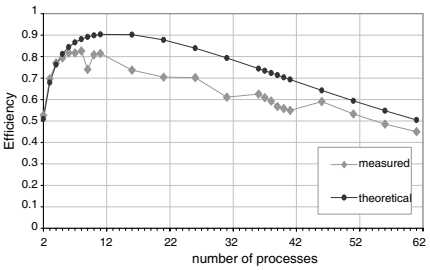

Figure 11. T_1 test model efficiency on SP5.

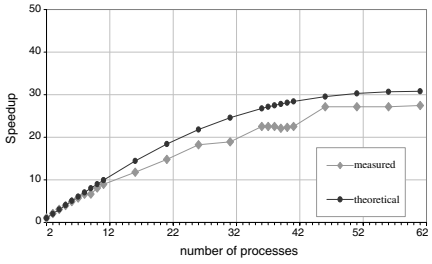

Figure 12. T_1 test model speedup on SP5.

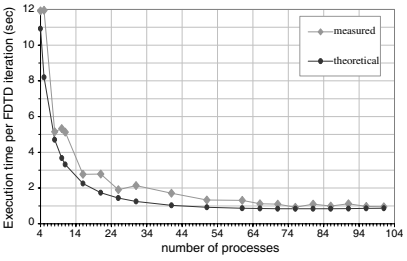

Figure 13. T_2 test model timing on SP5.

and speedup are shown in Figs. 10, 11 and 12 under the "theoretical" label. From those expressions, values for $P_{\mathcal{S}_{\max}}$, $\mathcal{E}_{\max}$, $P_{\mathcal{E}_{\max}}$ and $\mathcal{S}_{\max}$ are obtained by deriving with respect to P. Omitting cumbersome calculations, we get for the SP5 and BCX machines the following results:

	$P_{\mathcal{E}_{\max}}$	$\mathcal{E}_{\max}$	$P_{\mathcal{S}_{\max}}$	$\mathcal{S}_{\max}$
SP5	$\simeq 13$	$\simeq 0.91$	$\simeq 61$	$\simeq 30.8$
BCX	$\simeq 11$	$\simeq 0.89$	$\simeq 45$	$\simeq 23.0$

to which corresponds the following times per FDTD iteration:

	$T_\pi(P_{\mathcal{S}_{\max}})$	$T_\pi(P_{\mathcal{E}_{\max}})$
SP5	$\simeq 0.79$ sec	$\simeq 2.0$ sec
BCX	$\simeq 1.8$ sec	$\simeq 4.2$ sec

It would be possible to get a better efficiency, say $\mathcal{E} = 0.95$, by increasing ad hoc the size Ω_c of the coarse embedding grid and/or reducing the size $\Omega_{s,1}$ of the embedded refined sublattice, to force the $T_{\pi,s}$ and $T_{\pi,c}$ equalization in the first equation of system (14).

$\underline{\mathrm{T}_2 \text{ test model:}}$ here we have $\mathcal{N} = 2$ and $L = 1$ (thus implying $\Gamma = (P-1)/2$), with $P \geq 3$. The set of coefficients in (17) is now:

$$\begin{aligned} a &= 1.31 \times 10^{10} \times \tau_{fp}\,\text{sec.} \\ b &= (62 \times \tau_{la} + 1520176 \times \tau_{bw})\ \text{sec.} \\ c &= (-112 \times \tau_{la} - 2968154 \times \tau_{bw})\ \text{sec.} \end{aligned}$$

Going on as in the T_1 case, we get:

	$P_{\mathcal{E}_{\max}}$	$\mathcal{E}_{\max}$	$P_{\mathcal{S}_{\max}}$	$\mathcal{S}_{\max}$
SP5	$\simeq 16$	$\simeq 0.92$	$\simeq 78$	$\simeq 39.5$
BCX	$\simeq 13$	$\simeq 0.90$	$\simeq 56$	$\simeq 28.1$

	$T_\pi(P_{\mathcal{S}_{\max}})$	$T_\pi(P_{\mathcal{E}_{\max}})$
SP5	$\simeq 0.84$ sec	$\simeq 2.3$ sec
BCX	$\simeq 2.0$ sec	$\simeq 4.9$ sec

Relevant graphs are shown in Figs. 13, 14 and 15.

$\underline{\mathrm{T}_3 \text{ test model:}}$ here we have $\mathcal{N} = 2$ and $L = 2$ (thus implying $\Gamma = (P-2)/2$), with $P \geq 4$. The set of coefficients in (18) is now:

$$\begin{aligned} a &= 1.31 \times 10^{10} \times \tau_{fp}\,\text{sec.} \\ b &= (62 \times \tau_{la} + 1520176 \times \tau_{bw})\ \text{sec.} \\ c &= (-112 \times \tau_{la} - 2968154 \times \tau_{bw})\ \text{sec.} \end{aligned}$$

Going on as in the T_1 case, we get:

comm.	$P_{\mathcal{E}_{\max}}$	$\mathcal{E}_{\max}$	$P_{\mathcal{S}_{\max}}$	$\mathcal{S}_{\max}$
SP5	$\simeq 19$	$\simeq 0.87$	$\simeq 79$	$\simeq 39.5$
BCX	$\simeq 16$	$\simeq 0.84$	$\simeq 57$	$\simeq 28.1$

comm.	$T_\pi(P_{\mathcal{S}_{\max}})$	$T_\pi(P_{\mathcal{E}_{\max}})$
SP5	$\simeq 0.84$ sec	$\simeq 1.98$ sec
BCX	$\simeq 1.98$ sec	$\simeq 4.28$ sec

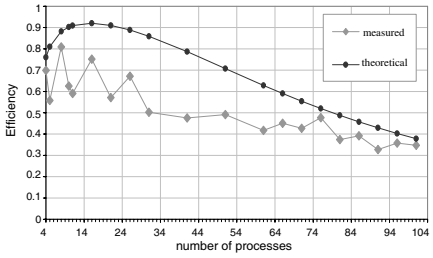

Figure 14. T_2 test model efficiency on SP5.

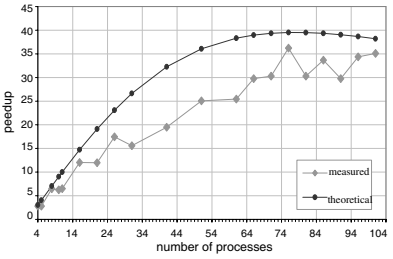

Figure 15. T_2 test model speedup on SP5.

Relevant graphs are shown in Figs. 16, 17 and 18. Also for T_2 and T_3 configurations it would be possible to calculate more efficient sizings, as has been mentioned for the T_1 case, by forcing $T_{\pi,s}$ and $T_{\pi,c}$ equalization in the first equation of system (14). To end this Subsection, we point out that the scattering and jaggedness in the measured "experimental" data shown in the graphs along with their "theoretical" counterparts are due to non-reproducibility, mainly from a lack of exclusive access to the computing machine. The constant loading of the computing machine by extraneous jobs concretizes in an extra amount of overhead time T_o, other than that arising simply from the non instantaneous communication between processes. Although unpredictable (and apart the possibly uncorrected estimates for τ_{la}, τ_{bw} and τ_{fp}), such effect let one to appreciate the fairly good efficiency and scalability of the proposed parallelization algorithm, once the most suitable amount of slicing has been chosen for a given geometry and model complexity.

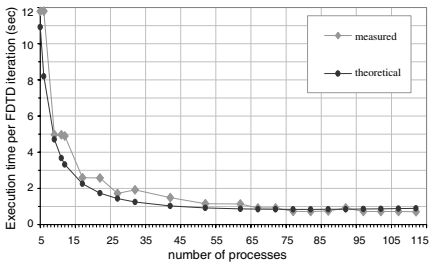

Figure 16. T_3 test model timing on SP5.

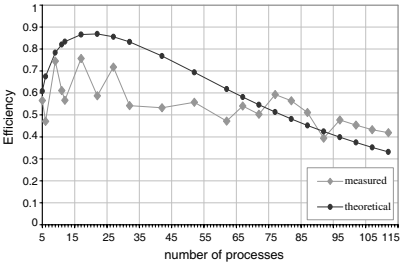

Figure 17. T_3 test model efficiency on SP5.

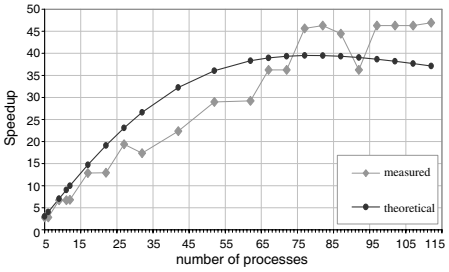

Figure 18. T_3 test model speedup on SP5.

3.4. Load Balancing. Nonblocking Communication

With a better *load balancing* — a fine tuning of the amount of work made by each process — one may hope to lower further the overhead time T_o (6), by reducing the *idle time* contribution due to a lack of synchronism between processes execution. A good way to achieve such a reduction is, if the program structure allows it, by overlapping computation and communication. This is accomplished by using the nonblocking *MPI::Isend* and *MPI::Irecv* routines [20, 21, 27, 28], were the I prefix stands for "Immediate" return, in place of the more common blocking ones, i.e., *MPI::Send* and *MPI::Recv*. As already pointed out, however, the FDTD algorithm iteration cycle is poorly prone to such an overlapping, because the field values are passed among the processes in a domain decomposition context, and are immediately used in the difference equations. As a consequence, a little gain is got from the use of the nonblocking communication functions. The better strategy remains an optimal choice of the size parameter values, so that the processes happen to be evenly loaded. The only point in the task list of Fig. 5 which could suit a "small" communication/computation overlapping is that labeled B' in the right column. After the *Recv* call, an R sequence of sub-cycles (R being the refinement factor) iterating on the refined sublattice grid is started. The *first* of such iterations could indeed start immediately without waiting the *Recv* completion, at least until the space-time interpolation routine is called. Hence the *Recv* completion can be slightly forward shifted. The better way to load balancing the code is by the fullfillment, as much as possible, of the criterion (14), particularly with respect to the size of the coarse grid relatively to the embedded refined ones, because a great amount of inefficiency is seen by processes running coarse grid computations spending a lot of time in communication. Keeping in mind that the subgridding is fixed, i.e., not dynamically varying during the code runs, a rough criterion for load balancing is to remember, given the refinement factor R, that the flop number in the refined grid is R^4 greater than the flop number in the coarse one.

4. A LARGE FDTD NUMERICAL EXPERIMENT

In this Section we apply the above described parallel version of the subgridding algorithm [8] to a moderately large FDTD model, made of five standing men on a perfectly conducting ground plane and exposed to the e.m. near field of an antenna (Fig. 19). The constitutive elements of the model are the same as those previously used in the test configurations T_1, T_2 or T_3, but now arranged to give an estimated *serial* execution time of $\simeq$ 216 sec per FDTD

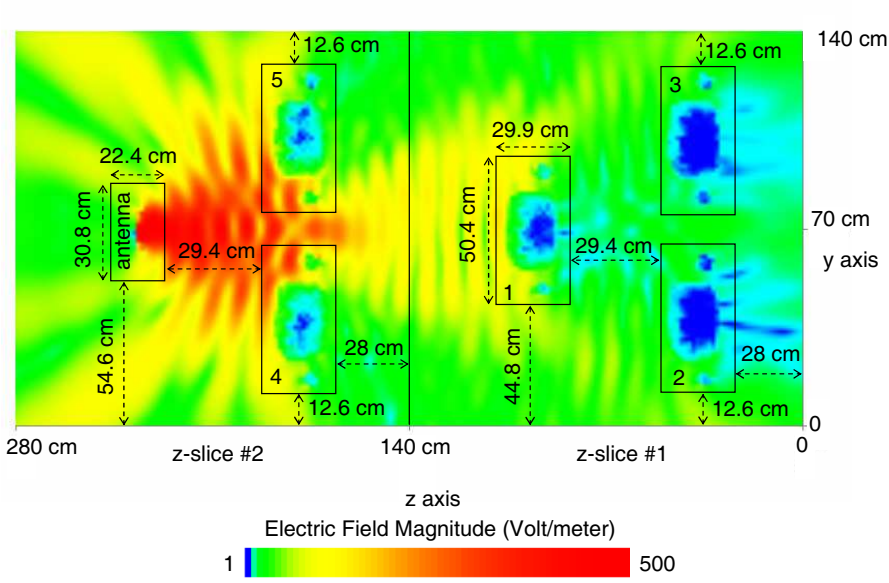

Figure 19. Top view of a plane from the outer coarse lattice with information about geometry and displacements of the various embedded refined sublattices for the antenna and the humans. Log. scale.

iteration on BCX, and of $\simeq$ 129 sec per FDTD time iteration on SP5. The modeled space volume has sizes of: 2.80 m (Height, along x-axis) $\times$1.40 m (Width, along y-axis) $\times$2.80 m (Depth, along z-axis) and thus, assuming that 1200 FDTD iterations suffice to stimulate every part of the computational domain by the excitation antenna signal — for good frequency domain results via Semi-Discrete Fourier transforms —, about 72 hours (3 days) on BCX and about 43 hours (1.8 days) on SP5 should be required for completing the calculations with conventional sequential programming. The *parallel* execution of the model uses 2 z-slices (like the T_3 test), each taking 100 coarse cells along the z-axis. Each z-slice contains 3 sub-lattices, each refined by a factor of 7 with respect to the coarse cubic cells which have an edge of 1.4 cm. Five sub-lattices contain, at a refined level, the relative dielectric constant and conductivity (Siemens/m) data values for the standing man ($875 \times 252 \times 147$ cubic cells), conformed to the anatomical structures as described in [22]. The remaining one contains data for the antenna metallic structures ($700 \times 154 \times 112$ cubic cells). On the final field values, we also perform some energetic checks, both to assess the reliability of the numerical calculations for dosimetric

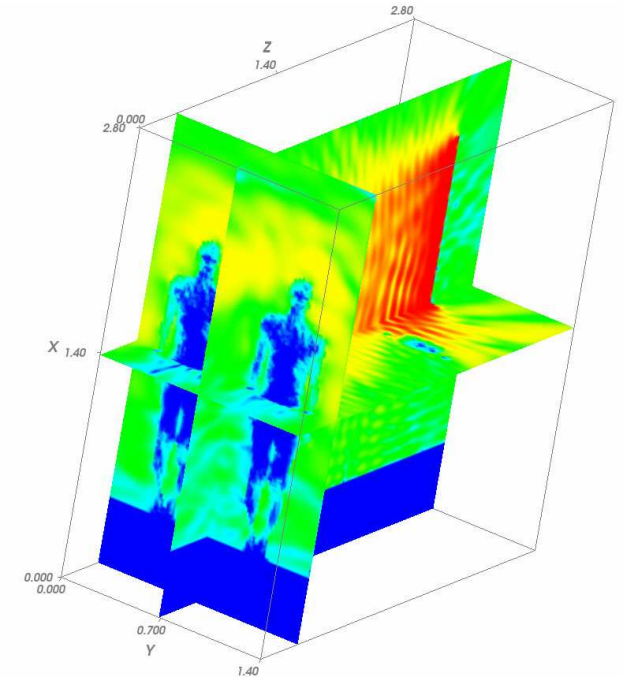

Figure 20. Three planes from the coarse grid. Perspective back view. Log. scale.

analysis and as an inspection of the goodness of the numerical solution, being lacking for such a complex model the analytic reference solution. For pure MPI code the running time results are shown in Table 5, from which we can see how the parallelization technique here proposed allows for a strong shortening of the execution FDTD times and with a fairly good scalability. In Table 6 it is summarized the mentioned power balance check at the antenna working frequency of 1862.5 MHz (human relative dielectric constants and conductivities are given for this frequency value too). The first row gives the total e.m. power emitted by the antenna, in equilibrium with the loads represented by the five humans in front to it. That value is the real part of the complex Poynting vector flux on a parallelepiped surface completely surrounding the antenna inside the refined sub-lattice (five refined cells apart from its outer surface). The second row gives the e.m. power leaving the computational domain. Of the two value in the row, the first is the real part of the complex Poynting vector flux on a parallelepiped surface completely surrounding the antenna and the humans inside the coarse lattice (five coarse cells apart from its outer

surface). The second value is obtained by means of a near-to-far field transformation — a Kirchhoff integral formula [29] implementation — and an integration over the 4π sterad angle at an extrapolated infinite distance from the center of the model. Rows from the third to the seventh have values of the e.m. power absorbed by the human loads. The absorbed power can be calculated in either of two ways.

The first value is obtained as the real part of the complex Poynting vector *inflow* on a parallelepiped surface completely surrounding the human inside the refined sub-lattice (five refined cells apart from its outer surface). The alternative is to calculate the power dissipation in the sub-regions where the electric conductivity $\sigma \neq 0$, as $\sigma \|\vec{E}\|^2/2$ and summing (integrating) up. The eighth row gives the total amount of e.m. absorbed power. The ninth row gives the sum of the e.m. powers absorbed and leaving the volume space under consideration, a value which has to equal the e.m. power emitted from the antenna. As can be seen, a fairly good result is obtained (within the 3% of the emitted power reference), despite the single precision floating point format, the absorbing boundary conditions effect, the coarse/refined

Table 5. MPI parallel execution times for 1200 FDTD iterations (sec).

	# proc.s: $105 + 2$	# proc.s: $140 + 2$	# proc.s: $350 + 2$
BCX	$4,137$ ($\simeq 1.15$ h)	$3,862$ ($\simeq 1.07$ h)	$1,934$ ($\simeq 0.54$ h)
SP5	$2,470$ ($\simeq 0.69$ h)	N/A	N/A

Table 6. Power balance (Watt) for the large FDTD numerical experiment.

Emitted Poynting (antenna-refined)	79.96	79.96
Escaped Poynting/Kirchhoff (outer-coarse)	43.57	43.56
Absorbed Poyn./Dissip. (sublatt.1-refined)	6.03	6.02
Absorbed Poyn./Dissip. (sublatt.2-refined)	1.44	1.44
Absorbed Poyn./Dissip. (sublatt.3-refined)	1.55	1.55
Absorbed Poyn./Dissip. (sublatt.4-refined)	12.72	12.63
Absorbed Poyn./Dissip. (sublatt.5-refined)	12.22	12.13
Subtotal Absorbed (1 to 5)	33.96	33.77
Subtotal Absorbed + Escaped	77.53	77.33
Deviation from Emitted	2.43 (3%)	2.63 (3.2%)

mesh sizes coupling (with filtering) effect, the 1200 FDTD iterations (which become $1200 \times 7 = 8400$ inside the refined sub-lattices) cumulative truncation error effect. Fig. 20 is a perspective view of three simultaneous mutually perpendicular planes from the coarse domain. The test has given good accuracy and performance results and the algorithm could be proposed for solving accurately the problems described in [30] and [31] with bigger sizes.

5. CONCLUSION

The present paper starts on the ground of a subgridding algorithm for the three-dimensional (3-D) FDTD numerical solution method of the Maxwell's equations, proposed in a preceding paper [8] by the authors and there described using a serial programming approach. Now a parallel code version of the same algorithm is proposed, for very large size FDTD calculations. This parallel version of the subgridding algorithm exploits a C++ object-oriented programming approach, for ease of maintenance and improvement. The parallelization is made by means of an MPI API implementation. To keep the parallel part of the code at the speed level of the bulk FDTD calculations, suitable user defined MPI data types were introduced, along with an adequate domain decomposition of the coarse and refined meshes and their interfaces organization. The main concept of our subgridding algorithm is that a shorter space-step is chosen only in sub-regions where an accurate discretized description of the geometrical surface details is needed, or sub-regions where the wavelength is shortened by the presence of high permittivity media. It is assumed that such sub-regions are embedded in homogeneous, isotropic, air-like regions where the accuracy criterion is easily verified with a larger space step. The subgridding is thus confined to volumes of minimum possible extent. For a given coarse/fine ratio (which we assume the best from the viewpoint of the geometrical/electrical constraints), the shortest simulation time is achieved with an accurate load balancing obtained by an appropriate domain decomposition (slicing). The execution time depends also on the cluster network bandpass, due to data exchange between processes executing contiguous domain portions. There is no limitation on the number of slices decomposing a given domain, but it is shown that, for a given initial size, there is an optimum of slices beyond which communication overwhelms computation. Our parallel code has been tested with moderately large FDTD models and, although affected by a lack of exclusive access to the cluster, shown to give fairly good speedup and scalability, which are also analyzed from a theoretical point of view.

REFERENCES

1. Yee, K. S., "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Trans. Antennas Propagat.*, Vol. 14, 302–307, May 1966.
2. Taflove, A., "Application of the finite-difference time-domain method to sinusoidal steady-state electromagnetic penetration problems," *IEEE Trans. Electromagnetic Compatibility*, Vol. 22, 191–202, Aug. 1980.
3. Taflove, A. and S. C. Hagness, *Computational Electrodynamics: The Finite-difference Time-domain Method*, 3rd edition, Artech House, Norwood, MA, 2005.
4. Okoniewski, M., E. Okoniewska, and M. A. Stuchly, "Three-dimensional subgridding algorithm for FDTD," *IEEE Trans. Antennas Propagat.*, Vol. 45, 422–429, Mar. 1997.
5. Yu, W. and R. Mittra, "A new subgridding method for finite difference time domain (FDTD) algorithm," *Microwave Opt. Technol. Lett.*, Vol. 21, No. 5, 330–333, Jun. 1999.
6. Wang, B., Y. Wang, W. Yu, and R. Mittra, "A hybrid 2-D ADI-FDTD subgridding scheme for modeling on-chip interconnets," *IEEE Transactions on Advanced Packaging*, Vol. 24, No. 4, 528–533, Nov. 2001.
7. Marrone, M., R. Mittra, and W. Yu, "A novel approach to deriving a stable hybridized FDTD algorithm using the cell method," *IEEE International Symposium on Antennas and Propagation*, Columbus, OH, Jun. 2003.
8. Vaccari, A., R. Pontalti, C. Malacarne, and L. Cristoforetti, "A robust and efficient subgridding algorithm for finite-difference time-domain simulations of Maxwell's equations," *J. Comput. Phys.*, Vol. 194, 117–139, 2003.
9. Guiffaut, C. and K. Mahdjoubi, "A parallel FDTD algorithm using the MPI library," *IEEE Antennas and Propagat. Mag.*, Vol. 43, No. 2, 94–103, Apr. 2001.
10. Yu, W., Y. Liu, T. Su, N. Huang, and R. Mittra, "A robust parallel conformal finite-difference time-domain processing package using the *MPI* library," *IEEE Antennas and Propagat. Mag.*, Vol. 47, No. 3, 39–59, Jun. 2005.
11. Chen, X., M. Cracraft, Y. Zhang, J, Zhang, J. Drewniak, B. Archambeault, and S. Connor, "An Efficient Implementation of Parallel FDTD," *IEEE International Symposium on Electromagnetic Compatibility, 2007. EMC 2007*, Honolulu, HI, Jul. 2007.

12. Yu, W., X. Yang, Y. Lin, L. Ma, T. Su, N. Huang, R. Mittra, R. Maaskant, Y. Lu, Q. Che, R. Lu, and Z. Su, "A new direction in computational electromagnetics: Solving large problems using the parallel FDTD on the BluGene/L supercomputer providing teraflop-level performance," *IEEE Antennas and Propagat. Mag.*, Vol. 50, No. 2, 26–44, Apr. 2008.
13. Liu, Y., Z. Liang, and Z. Yang, "A novel FDTD approach featuring two-level parallelization on PC cluster," *Progress In Electromagnetics Research*, Vol. 80, 393–408, 2008.
14. Josuttis, N. M., *The C++ Standard Library. A Tutorial and Reference*, Addison-Wesley, 1999.
15. Gedney, S. D., "An anisotropic perfectly matched layer-absorbing medium for the truncation of FDTD lattices," *IEEE Trans. Antenna Propagat.*, Vol. 44, No. 12, 1630–1639, Dec. 1996.
16. Courant, R., K. O. Friedrichs, and H. Lewy, "Über di partiellen differenzenglei-chungen der mathematischen physik," *Math. Ann.*, Vol. 100, 32–74, 1928, German Original Paper.
17. Courant, R., K. O. Friedrichs, and H. Lewy, "On the partial difference equations of mathematical physics," *IBM Jour. of Res. and Dev.*, Vol. 11, 215–234, 1967, English Translation of the Original Paper.
18. Taflove, A. and M. E. Brodwin, "Numerical solution of steady-state electromagnetic scattering problems using the time-dependent Maxwell's equations," *IEEE Trans. Microwave Theory Tech.*, Vol. 23, 623–630, Aug. 1975.
19. Mur, G., "Absorbing boundary conditions for the finite difference approximation of the time domain electromagnetic field equations," *IEEE Trans. Electromagn. Comp.*, Vol. 23, 377–382, Nov. 1981.
20. Pacheco, P. S., *Parallel Programming with MPI*, Morgan Kaufmann Publishers, San Francisco, 1997.
21. Gropp, W., E. Lusk, and A. Skjellum, *Using MPI. Portable Parallel Programming with the Message-passing Interface*, 2nd edition, The MIT Press, Cambridge, Mass., 1999.
22. Mazzurana, M., L. Sandrini, A. Vaccari, C. Malacarne, L. Cristoforetti, and R. Pontalti, "A semi-automatic method for developing an anthropomorphic numerical model of dielectric anatomy by MRI," *Physics in Medicine and Biology*, Vol. 48, No. 19, 3157–3170, 2004.
23. http://www.mcs.anl.gov/mpi, for the MPICH Implementation of the MPI Libraries.

24. http://www.open-mpi.org/, the OpenMPI Implementation of the MPI Libraries Official Web Site.
25. http://www.openmp.org, the OpenMP Official Web Site.
26. Chandra, R., L. Dagun, D. Kohr, D. Maydan, J. McDonald, and R. Menon, *Parallel Programming in OpenMP*, Morgan Kaufmann Publishers, San Francisco, 2001.
27. Quinn, M. J., *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill, 2003.
28. Karniadakis, G. E. and R. M. Kirby II, *Parallel Scientific Computing in C++ and MPI*, Cambridge University Press, 2005.
29. Jackson, J. D., *Classical Electrodynamics*, 3rd edition, Wiley, 1998.
30. Hirata, A., H. Sugiyama, and O. Fujiwara, "Estimation of core temperature elevation in humans and animals for whole-body averaged SAR," *Progress In Electromagnetics Research*, Vol. 99, 53–70, 2009.
31. Islam, M. T., M. R. I. Faruque, and N. Misran, "Design analysis of ferrite sheet attachment for SAR reduction in human head," *Progress In Electromagnetics Research*, Vol. 98, 191–205, 2009.