Parallel High-Order EM-FVTD on an Unstructured Mesh

Ian Jeffrey, Dmitry K. Firsov, Colin Gilmore, Vladimir Okhmatovski and Joe LoVetri

Department of Electrical and Computer Engineering
University of Manitoba, Winnipeg, MB, R3T 5V6, Canada
ijeffrey@ee.umanitoba.ca, okhmatov@ee.umanitoba.ca, lovetri@ee.umanitoba.ca

Abstract: We present a summary of the steps taken to parallelize a high-order, unstructured mesh, EM-FVTD solver for distributed parallel systems. We use a modified version of ORB for domain partitioning on an arbitrary number of processors, and layer-by-layer halo element acquisition suited for a wide variety of flux-integration schemes. The resulting code is capable of solving Maxwell's Equations with high-order MUSCL or polynomial interpolation flux-integration, high-order absorbing boundary conditions and up to fourth-order time-integration. The parallel algorithm shows excellent scalability and is capable of solving in excess of four million unknowns per processor.

Keywords: Distributed computing, Finite-volume time-domain, Maxwell's equations.

1. Introduction

Over the past two years, the Computational Electromagnetics (CEM) group at the University of Manitoba has created a very flexible, object-oriented, C++ finite-volume time-domain (FVTD) electromagnetics solver that we now call SUMO-FIVE (Serial University of Manitoba Object-oriented FinIte-Volume Engine). Our engine solves Maxwell's equations using a cell-centered characteristic approach on unstructured meshes [1-4].

The purpose of this paper is to describe the steps taken in order to parallelize our serial algorithm for a distributed parallel system. We call our parallel implementation PUMA-FIVE (Parallel University of Manitoba Accelerated-FIVE). We begin with a brief general introduction to FVTD in Section 2, followed by an overview of characteristic based FVTD for Maxwell's equations on unstructured grids in Section 3. In Section 4, we devote attention to the various high-order boundary conditions and flux- and time-integration techniques available to SUMO-FIVE in order to clarify the parts of the current algorithm that require special care during parallelization. Section 5 briefly summarizes the current state of parallel FVTD algorithms in the CEM community, while Section 6 elaborates on the steps taken to parallelize our FVTD code. Finally, Section 7 presents a summary of the performance of the resulting algorithm.

2. Background

Since the early 1990s, the electromagnetics community has been making use of FVTD for solving Maxwell's equations [5-8], an algorithm adopted from the field of Computational Fluid Dynamics (CFD) [9]. Prior to FVTD being introduced into the CEM community, the finite-difference time-domain (FDTD) algorithm was the time-domain method of choice for CEM researchers [10]. Even today, when increased computational speed and resources make the FVTD algorithm a viable alternative, FDTD dominates the CEM community due to its simplicity.

As FDTD usually requires a rectangular, interlaced spatial grid, standard FDTD implementations are plagued by their inability to accurately model curved surfaces and volumes. As a result, complex geometries are typically approximated by "stair-stepping" the physical parameters associated with the standard FDTD

grid. For accurate modelling of such geometries, the FDTD algorithm requires a finely resolved grid which, in accordance with a well known stability criterion, decreases the maximum allowable time-step for the algorithm. Thus, both increased memory requirements and increased computational time are required to reproduce accurate fields in the presence of complex geometries. Although several successful attempts to modify the FDTD algorithm for non-rectangular boundaries have been made, they have proven to be difficult to implement [10].

Modelling complex geometries is where the FVTD algorithm excels. Geometry discretization is performed accurately in one of two ways: first, many implementations of the algorithm use a structured, volumetric, body-fitted curvilinear mesh [11-13]. The primary drawback of such body-fitted meshes is that mesh generation is difficult for highly intricate geometries and often over-set meshes are required [11]. The second way that FVTD algorithms can describe the problem geometry is through an unstructured volumetric mesh [14] and SUMO-FIVE adopts this approach. Typical implementations use first-order tetrahedral elements which can provide a piece-wise linear, first-order approximation to a curved surface or volume. FVTD solvers that make use of unstructured grids, such as our own implementation, are easy to implement provided that an efficient representation of the unstructured volumetric mesh is available [2]. We have previously shown the benefits of FVTD for solving scattering from a PEC sphere, where a more accurate solution was obtained from the FVTD algorithm with fewer than 10% of the unknowns required for the FDTD solution [3].

Finally, the mesh representation adopted for our solver is not limited to tetrahedral elements but may also include a mixture of prisms, pyramids and hexahedrals. Further, the implementation of our algorithm is easily modified for higher-order elements and includes high-order boundary truncation schemes in addition to high-order time- and flux-integration methods which will be discussed in Section 4.

3. FVTD Formulation of Maxwell's Equations

The FVTD algorithm is generally used to solve problems involving *conservation laws*. As an example, consider a scalar function $u(\boldsymbol{x}, t)$ that satisfies:

$$\int_{T_i} \partial_t u(\boldsymbol{x}, t) dV + \int_{\partial T_i} \boldsymbol{f}(u(\boldsymbol{x}, t)) \cdot d\boldsymbol{s} = \int_{T_i} S(\boldsymbol{x}, t) dV \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^3$, the flux vector $\boldsymbol{f}$ is some function of u, $S(\boldsymbol{x}, t)$ is a source term and integration is over an arbitrary volume T_i, bounded by ∂T_i with $d\boldsymbol{s}$ being the outward directed surface element vector. To apply FVTD to Maxwell's equations, we closely follow [5] beginning with Maxwell's two curl equations in SI units:

$$\begin{cases} \varepsilon \partial_t \boldsymbol{E} - \nabla \times \boldsymbol{H} + \sigma \boldsymbol{E} = -\boldsymbol{J} \\ \mu \partial_t \boldsymbol{H} + \nabla \times \boldsymbol{E} = 0 \end{cases} \tag{2}$$

Employing the 3×3 matrix operator

$$\boldsymbol{S}(\boldsymbol{x})\boldsymbol{b} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \boldsymbol{x} \times \boldsymbol{b} \tag{3}$$

the curl of a vector $\boldsymbol{x} \in \mathbb{R}^3$ can be expressed in terms of the divergence of a matrix operating on the vector as follows:

$$\nabla \times \boldsymbol{x} \equiv (\mathrm{div}\boldsymbol{S}(\boldsymbol{x}))^{\mathrm{T}} \tag{4}$$

where $\mathrm{div}\boldsymbol{S}(\boldsymbol{x})$ is defined to be a row vector whose j^{th} entry is the divergence of the j^{th} column of $\boldsymbol{S}(\boldsymbol{x})$. Now, we may write Maxwell's equations as:

$$\partial_t \boldsymbol{u} + \alpha^{-1}\boldsymbol{K}\boldsymbol{u} = \alpha^{-1}(\boldsymbol{G} + \boldsymbol{B}\boldsymbol{u}) \tag{5}$$

with

$$\boldsymbol{u} \triangleq \begin{bmatrix}\boldsymbol{E}\\ \boldsymbol{H}\end{bmatrix},\ \alpha \triangleq \begin{bmatrix}\varepsilon & \boldsymbol{0}\\ \boldsymbol{0} & \mu\end{bmatrix},\ \boldsymbol{K}\boldsymbol{u} = \begin{bmatrix}-(\mathrm{div}\boldsymbol{S}(\boldsymbol{H}))^{\mathrm{T}}\\ (\mathrm{div}\boldsymbol{S}(\boldsymbol{E}))^{\mathrm{T}}\end{bmatrix},\ \boldsymbol{B} \triangleq \begin{bmatrix}-\sigma & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix},\ \boldsymbol{G} \triangleq \begin{bmatrix}-\boldsymbol{J}\\ \boldsymbol{0}\end{bmatrix}. \tag{6}$$

Similar to the scalar case, we integrate (5) over a finite-volume denoted by T_i with boundary ∂T_i. We assume that discrete solution values $\boldsymbol{u}_i$ are associated with the centroid of each finite-volume in the mesh. Applying the divergence theorem we obtain

$$\int_{T_i} \partial_t \boldsymbol{u}_i dV - \int_{\partial T_i} \alpha^{-1}\boldsymbol{A}\boldsymbol{u}_i d\boldsymbol{s} = \int_{T_i} \alpha^{-1}\boldsymbol{G} dV + \int_{T_i} \alpha^{-1}\boldsymbol{B}\boldsymbol{u}_i dV, \qquad \boldsymbol{A} = \begin{bmatrix}\boldsymbol{0} & -\boldsymbol{S}(\hat{\boldsymbol{n}})\\ \boldsymbol{S}(\hat{\boldsymbol{n}}) & \boldsymbol{0}\end{bmatrix} \tag{7}$$

where $\hat{\boldsymbol{n}}$ is the outward directed surface normal. For first-order volumetric elements ∂T_i is simply a set of N_i plane-facets *i.e.*, $\partial T_i = \bigcup_{n=1}^{N_i} \partial T_{i_n}$ where ∂T_{i_n} denotes the n^{th} facet of element i such that (7) becomes:

$$\int_{T_i} \partial_t \boldsymbol{u}_i dV - \sum_{n=1}^{N_i} \mu(\partial T_{i_n})\alpha^{-1}\boldsymbol{A}_n\boldsymbol{u}_i = \int_{T_i} \alpha^{-1}\boldsymbol{G} dV + \int_{T_i} \alpha^{-1}\boldsymbol{B}\boldsymbol{u}_i dV \tag{8}$$

where $\mu(\partial T_{i_n})$ is the facet area and $\boldsymbol{A}_n$ is appropriately constructed for the n^{th} facet. Next, we adopt a flux-splitting representation where the matrix $\boldsymbol{A}$ is split according to incoming and outgoing characteristics such that $\alpha^{-1}\boldsymbol{A}(\hat{\boldsymbol{n}}) = \tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{+} + \tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{-}$, where the appropriate α is employed for each side of the facet. Details are available in [5]. Consequently, the surface integral summation in (8) can be written as:

$$\sum_{n=1}^{N_i} \mu(\partial T_{i_n})\alpha^{-1}\boldsymbol{A}_n\boldsymbol{u}_i = \sum_{n=1}^{N_i} \mu(\partial T_{i_n})(\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{+} + \tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{-})\boldsymbol{u}_i = \sum_{n=1}^{N_i} \mu(\partial T_{i_n})(\tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{+}\boldsymbol{u}_{i_n}^{*} + \tilde{\boldsymbol{A}}(\hat{\boldsymbol{n}})^{-}\boldsymbol{u}_{i_n}^{**}), \tag{9}$$

where $\boldsymbol{u}_{i_n}^{*}$ is the value of $\boldsymbol{u}_i$ in the limit as we reach the facet ∂T_{i_n} from the interior of the element, and $\boldsymbol{u}_{i_n}^{**}$ is the value of the unknown on the same facet but in limit as we reach the facet from its neighbour across the facet.

This completes a preliminary description of FVTD for Maxwell's equations. Equation (8) must be solved for each finite-volume T_i in the mesh description of the problem. What remains is to select a method for approximating the time-derivative in (8) as well as methods for interpolating the solution from the stored values at cell-centres to facets. These schemes are often referred to as time- and flux-integration respectively.

4. Parallel Considerations: High Order Time- and Flux-Integration and Mesh Truncation Schemes

SUMO-FIVE makes use of various time- and flux-integration schemes in addition to an array of mesh truncation schemes in an object-oriented manner [1-4]. Currently, the code is capable of first-order upwinding, a second order Monotone Upstream-centred Scheme for Conservation Laws (MUSCL), and variable order polynomial interpolation for flux-integration. The details of each scheme are available in [3]

and are omitted here for brevity. For time-integration, we currently consider explicit schemes, namely first-order Euler, second-order predictor-corrector and up to fourth order Runga-Kutta. The currently available mesh-truncation schemes are first-order absorbing boundary conditions, second-order absorbing boundary conditions based on MUSCL interpolation, a modification of Liao's absorbing boundary conditions and a high-order integral equation truncation scheme [3-4].

As a result of the multitude of available schemes, we desire a parallelization technique that maintains the functionality and performance of the sequential code, while offering good scaling performance over multiple processors. As we currently only use explicit time-integration schemes, a simple parallel approach is to adopt a global time-step common to all processors. In this manner, each processor is capable of performing time-integration locally without concern for the behaviour of other processors. Adopting variable time-steps on each processor subdomain will be considered in future work.

In contrast, flux-integration requires cooperation between processors. Specifically, in order to update volume T_i, we require knowledge of $\boldsymbol{u}_{i_n}^{*}$ and $\boldsymbol{u}_{i_n}^{**}$ at each facet n of each finite-volume T_i. For volumes who's neighbours share the same processor as the volume itself, this computation can be performed locally. When an element's neighbour happens to lie on a different processor, communication must be used to make the value of $\boldsymbol{u}_{i_n}^{**}$ available to volume T_i. In general, all flux-integration techniques under consideration can be viewed as requiring a stencil of elements surrounding the control volume T_i. For first-order schemes we require only the neighbours sharing common facets with element T_i. For MUSCL, we require two layers of neighbours and for third-order polynomial interpolation we require three layers of neighbours.

Finally, absorbing boundary conditions are also easily parallelized. As they depend only on local elements (and possible neighbours), any parallelization technique that satisfies flux-integration requirements will, by default, handle absorbing boundary conditions. In contrast, handling the integral equation truncation scheme is a slightly more complicated procedure which we will discuss in our future publications.

5. The Literature on Parallel FVTD for CEM

Parallel FVTD for CEM has been in the literature for over a decade. In 1995, Rowell *et. al.* presented a characteristic based parallel FVTD implementation on structured and unstructured grids using "geometry zoning" for load balancing [15] while in 1997, Blake and Buter reported parallel FVTD for an over-set body-fitted grid with 3rd-order accurate space- and second-order accurate time-integration [11]. In 2000, Shang *et. al.* provided an excellent report of their MPI-based "ghost-layer" implementation of parallel FVTD for a third-order accurate body-fitted curvilinear algorithm [12] and more recently, Baduel *et. al.* reported a second-order accurate leap-frog scheme for regular meshes [14]. Other implementations and details can be found in [13],[16-19].

As our FVTD code contains arbitrary order flux-integration schemes and high-order absorbing boundary conditions, many of the aforementioned parallel FVTD schemes are applicable, subject to minor modifications.

6. Design of the Parallel Algorithm

Our parallel FVTD algorithm is a single-program, multiple-data (SPMD) code written for a distributed cluster using standard Message Passing Interface (MPI) routines in C++. Our approach, like many of the approaches previously listed, is to obtain an *essential* set of information for updating finite-volumes on a given processor at each time-step. The algorithm can be broken into four parts: *mesh-reading, domain partitioning, stencil acquisition* and *processing*. We address each of these issues in detail.

MESH-READING

To run the algorithm on p processors we begin by performing a parallel mesh read. It is imperative when solving problems involving a large number of unknowns that a single processor is not responsible for

reading the entire mesh. I/O operations are costly, and when dealing with meshes including tens or even hundreds of millions of volumetric elements, a serial mesh-read can become computationally exhausting. This problem is rarely addressed in the CEM literature.

Currently, we operate with mesh files that first list a series of vertices, indexed by a global id, followed by a list of elements whose vertices are denoted by the global vertex ids. The total number of vertices and elements begin each list respectively. Most commercial meshing packages offer this format.

We begin the parallel mesh read operation by first assigning to each processor an equal sequential subset of the vertices to read. After reading the vertices, each processor then reads an equal sequential subset of the elements. Because each processor reads a sequential subset of the vertices, and because the vertices are listed sequential in terms of their global ids, each processor can next make a request to the appropriate processor for the vertices required to complete its local elements. At this point, domain partitioning can be performed.

Domain Partitioning

This step is accomplished using a slightly modified version of the method of Orthogonal Recursive Bisection (ORB) first introduced by Salmon [22] where partitioning is performed on the centroid of the volumetric element. Within the majority of engineering and CEM literature, a *k*-level ORB algorithm is described as producing 2^k equally balanced subdomains, limiting the number of processors to being a power of two. A small modification to the algorithm makes it suitable for an arbitrary number of processors.

Assume that the initial domain, denoted D_0 contains M volumetric elements that we wish to divide equally among p processors. We begin by attributing a weight of p to the original domain. Partitioning of the domain D_0 will yield two new domains D_{00} and D_{01}. Instead of bisecting the domain into two equal subdomains, we assign weight $w_{00} = \lfloor p/2 \rfloor / p$ to D_{00} and weight $w_{01} = \lceil p/2 \rceil / p$ to D_{01}. We then partition the original domain such that D_{00} contains $w_{00}M$ elements while domain D_{01} contains $w_{01}M$ elements. We take special care to handle elements that lie directly on the partitioning plane such that the disbalance is at most ± 1 element. We then recursively divide D_{00} and D_{01} in the same manner using an alternate orthogonal direction. Once the weight for a given domain reaches $1/p$, its partitioning has been completed. This minor modification to ORB makes it a much more useful partitioning method for an arbitrary number of processors and does not seem to show itself in the engineering literature. The method proposed was developed independently from, but is generally the same as that proposed in [20]. Fig. 1 shows the partitioning of a 200,000 element spherical domain into seven balanced subdomains.

Stencil Acquisition

Once each element is transferred to its corresponding processor based on ORB partitioning, we require a final pre-processing step. It is desirable from a coding perspective to preserve as much of the serial nature of the code as possible. Ideally, we would like to have each processor run a serial FVTD code on the elements in its subdomain. Unfortunately, as discussed in Section 4, spatial interpolation techniques require information of neighbouring elements (and possibly the neighbours of neighbours etc.). As the different flux-integration techniques we have implemented depend on a different number of layers in the element stencils, we have chosen to build up these stencil elements on a layer-by-layer basis. Motivation for this approach stems from the fact that acquiring a single layer can be coded succinctly as its own function. Obtaining multiple layers can be achieved by repeated function calls. Also, for highly non-uniform meshes, neighbouring stencil elements may not be limited to a single neighbouring ORB domain. Our layer-by-layer

approach does not need to assume that all of the missing elements for a given ORB domain are situated in adjacent domains.

We now consider the case of elements lying on the boundaries of the ORB domains with neighbours on at least one other processor. We handle these boundary elements by providing each processor with the neighbours required to update all of its ORB elements. This results in duplication of a small percentage of the total mesh elements and an overlap in the ORB partitioning. An excellent account of the benefits and drawbacks of overlapping partitions for various computational algorithms is provided in [21]. Empirically, we have found that for large examples, this duplication leads to negligible memory overhead on a per-processor basis when compared to the actual memory required to store the ORB elements. In the literature, the duplicated elements are referred to as "ghost" elements [12] or "halo" elements.

To obtain the halo layers, each processor first searches through its ORB elements for missing neighbours. Once all of the facets that are missing connections have been determined, each processor requests the missing elements by means of these facets. Because a given processor does not necessarily know where the missing neighbouring elements lie, this request is made to all processors. Upon receiving the missing element requests, a given processor determines which of its elements are required for the halos of other processors. Two important steps must now be performed during the communication of the missing elements. First, the processor responding to the missing neighbour request maintains a record of its elements sent to other processors. Second, the receiving processor tracks the source of the elements it receives and flags them as being in *layer 1* of the halo. This procedure can be repeated to build the required number of halo layers. Of course at subsequent layers, only elements missing connections in the most recently received layer are considered. Fig. 2 shows the results of a two layer request for the MUSCL scheme for an ORB domain consisting of half of a 200,000 element sphere.

PROCESSING

Our implementation makes each processor responsible for updating its ORB elements, but not the halo elements. As each halo element is a unique ORB element on some processor, before each time step a single communication routine is used to update all halo elements from the computed ORB results. Of course, for high-order time-integration schemes, this information must be communicated each time the solution is updated. Upon receiving the updated solution for its stencil elements, each processor simply performs a serial FVTD algorithm for updating its ORB elements.

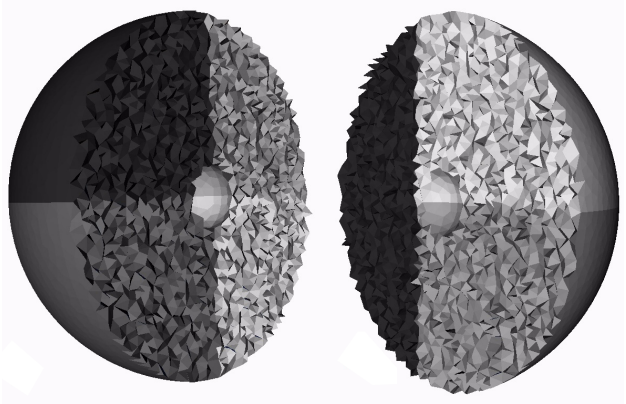

Figure 1. Modified ORB partitioning of a 200,000 element mesh enclosing a PEC sphere for 7 processors. Note that the volume on the left is 4/7 of the total volume while the volume on the right is 3/7 of the total.

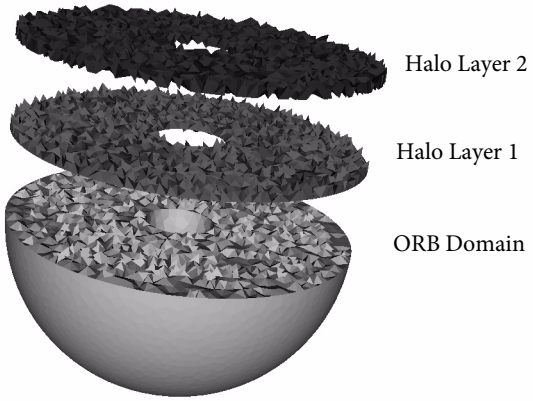

Figure 2. The two layers of stencil elements required for MUSCL on one of a two-processor partition for a 200,000 element volumetric mesh. Layer elements have been migrated for perspective.

7. Parallel Simulation Results

Previously, we have validated our sequential FVTD code for numerous geometries including scattering from a PEC sphere, PEC cube and dielectric cube [1-4]. Herein we are concerned only with parallel scalability of our parallel implementation PUMA-FIVE. The parallel algorithm was benchmarked on the University of Manitoba's ACN grid: a 10 node distributed cluster of Dual AMD Opteron 250 Sun servers (2.4 GHz) with 4 GB of memory per node. The nodes are connected via a gigabit switch. Three different mesh discretizations of a spherical volume surrounding a PEC sphere were tested using second-order space- and time-integration and first-order absorbing boundary conditions. The number of elements in each mesh was approximately 200 thousand, 1.6 million and 4 million finite-volumes. Fig. 3 below shows the desired scaling properties as the number of processors increases.

8. Conclusions

Our parallel FVTD solver, PUMA-FIVE, was easy to implement, demonstrates excellent scalability and is capable of solving problems involving a large number of unknowns. Specifically, the ACN grid nodes easily handled a problem involving 4 million unknowns on a single machine, requiring roughly 70 seconds per time-step and 2.45 GB of RAM. Following this trend, and given the size of available computing

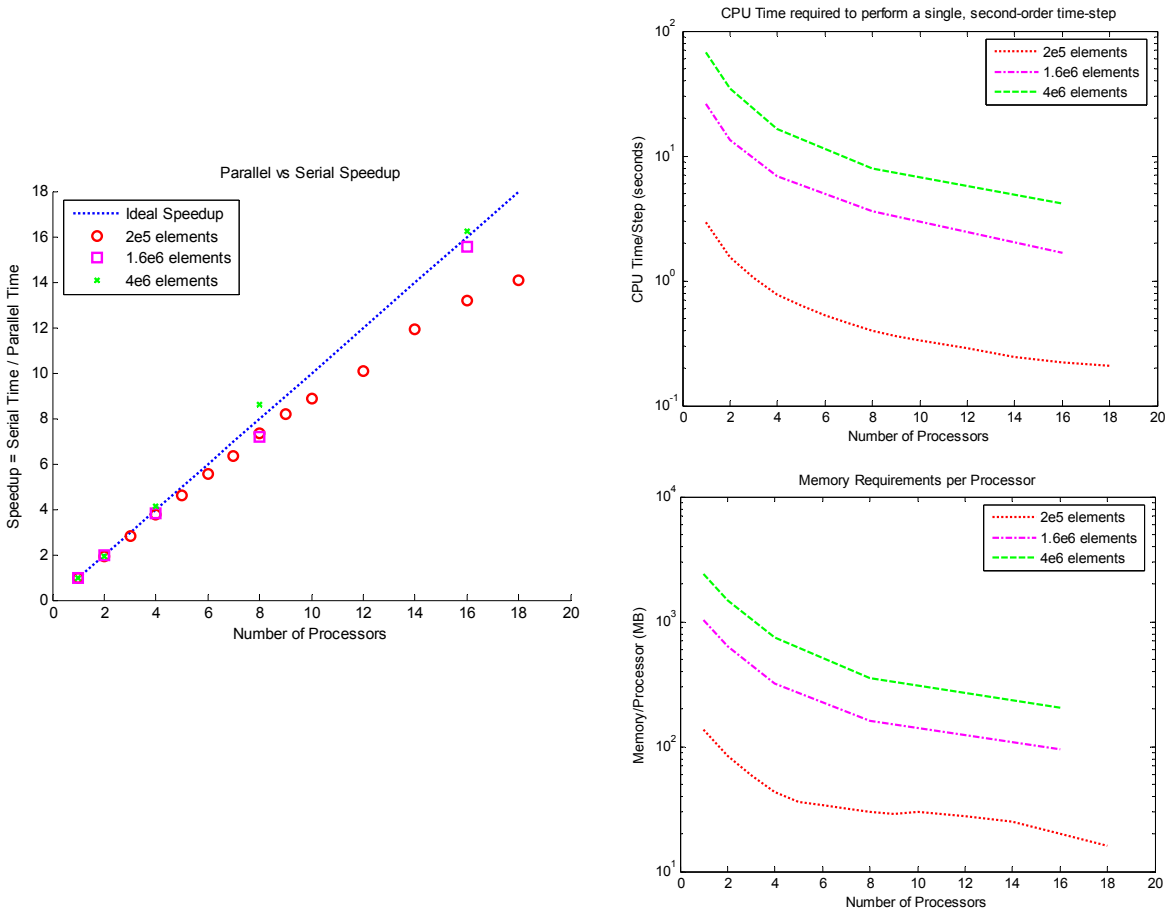

Figure 3. Parallel scaling of PUMA-FIVE. Speedup (left), CPU-Time per second-order time-step (top-right) and memory (bottom-right) for 200 thousand, 1.6 million and 4 million element meshes. The larger meshes were tested at 2, 4, 8 and 16 processors.

resources, PUMA-FIVE should easily be capable of solving problems involving hundreds of millions of unknowns. It remains to provide PUMA-FIVE with a parallel implementation of our integral-equation mesh-truncation scheme and dynamic load-balancing capabilities.

9. References

[1] D. K. Firsov, I. Jeffrey, J. LoVetri, W. Chamma, "High Order FVTD on Unstructured Grids," *2006 ACES Conference*, Miami, Florida, 2006.

[2] D. K. Firsov, I. Jeffrey, J. LoVetri and C. Gilmore, "An Object Oriented Finite-Volume Time-Domain Computational Engine", *2006 ACES Conference*, Miami, Florida, 2006.

[3] D. K. Firsov, J. LoVetri, I. Jeffrey, V. I. Okhmatovski, C. Gilmore, W. Chamma "High-Order FVTD on Unstructured Grids using an Object-Oriented Computational Engine", To appear in: *ACES Journal*, 2006.

[4] D.K. Firsov, J. LoVetri and V. I. Okhmatovski, "FVTD - Integral Equation Hybrid for Maxwell's Equations", *Frontiers in Applied Computational Electromagnetics (FACE)*, University of Victoria, Victoria, BC, 19-20 June, 2006.

[5] P. Bonnet, X. Ferrieres, P.L. Michielsen, P. Klotz, "Finite Volume Time Domain Method," in *Time Domain Electromagnetics,* S.M. Rao (editor), Academic Press, San Diego, 1999.

[6] V. Shankar, A. H. Mohammadian, W. F. Hall, "A Time-Domain Finite-Volume Treatment for the Maxwell Equations," Electromagnetics, Vol. 10, No. 1-2, pp 127 - 145, 1990.

[7] S. Piperno, M. Remaki, L. Fezoui, "A Nondiffusive Finite Volume Scheme For The Three-Dimensional Maxwell's Equations On Unstructured Meshes," *SIAM J. Numer. Anal.*, Vol. 39, No. 6, pp. 2089–2108, 2002.

[8] J. S. Shang, "Characteristic-Based Algorithms for Solving the Maxwell's Equations in the Time Domain," I*EEE Antennas and Propagation Magazine*, Vol. 37, No. 3, pp. 15-25, June 1995.

[9] C. Hirsch, *Numerical Computation of Internal and External Flows, Vol. I: Fundamentals of Numerical Discretization*, John Wiley & Sons Ltd., New York, 1988.

[10] A. Taflove (editor), *Advances in Computational Electrodynamics: The Finite-Difference Time-Domain Method*, Artech House Inc, Boston, 1998.

[11] D. C. Blake, T. A. Buter, "Electromagnetic Scattering Simulations Using Overset Grids on Massively Parallel Computing Platforms", *IEEE Anten. and Propagat. Soc. Int. Symp,* Vol. 1, pp. 106-109, 1997.

[12] J.S. Shang, M. Wagner, Y. Pan, and D. C. Blake, "Strategies for Adopting FVTD on Multicomputers", *Computing In Science & Engineering,* Vol. 2, No. 2, pp 10-21, 2000.

[13] Y. Pan, J. S. Shang, M. Guo, "A scalable HPF implementation of a finite-volume computational electromagnetics application on a Cray T3E parallel system", *Concurrency Computat.: Pract. Exper.* 2003, vol. 15, pp. 607-621.

[14] L. Baduel *et. al.* "A Parallel Object-Oriented Application for 3D Electromagnetism" *18th IEEE Para. and Dist. Proc. Symp, 2000.*

[15] C. Rowell, V. Shankar, W. F. Hall and A. Mohammadian "Algorithmic Aspects and Computing Trends in Computational Electromagnetics Using Massively Parallel Architectures", *Algo. and Arch. for Para. Proc., IEEE First Int. Conf. on,* Vol. 2, pp. 770-779, 1995.

[16] J. A. Camberos and P. M. Wheat, "A Finite-Volume, Time-Domain CEM Code for Unstructured-Grids on Parallel Computers", *Comp. Electromag in Time-Domain, IEEE Workshop on,* pp. 40-43, 2005.

[17] V. Ahuju, L. N. Long, "Scattering From Complex Geometries Using a Parallel FVTD Algorithm", *ACES 1996,* Monterey, California, 1996.

[18] V. Shankar, C. Rowell, W. F. Hall, "An Unstructured Grid-Based Parallel Solver for Time-Domain Maxwell's Equations", *IEEE Anten. and Propagat. Soc. Int. Symp.,* Vol. 1, pp. 90-101, June 1999.

[19] J. Camberos, P. M. Wheat, S. H. Wong, "Development and Status of a Finite-Volume Time-Domain CEM Research Code with Multidisciplinary Applications", *Proc. of the Users Group Conf.* pp. 47-51, *IEEE*, 2005.

[20] F. Wolfheimer, E. Gjonaj, and T. Weiland, "A parallel 3D particle-in-cell code with dynamic load balancing", *Nucl. Instr. & Methods in Phys. Research,* Section A, Vol. 558, No. 1, pp 202-204, March 2006.

[21] J. Galtier and S. Lanteri "On Overlapping Partitions" *Para. Proc., IEEE Int. Conf. on,* pp. 461-468, 2000.

[22] J. K. Salmon, *Parallel Hierarchical N-Body Methods*, Ph.D. thesis, Caltech, 1990.

[23] H. D. Simon, "Partitioning of Unstructured Problems for Parallel Processing", *Comp. Sys. in Eng.,* Vol. 2, No. 2-3, pp. 135-48, 1991.