ACES JOURNAL, VOL. 22, NO. 3, NOVEMBER 2007

# Electromagnetic Scattering Problems Utilizing a Direct, Parallel Solver

**William R. Dearholt**[1] **and Steven P. Castillo**[2]

[1] Box 1663, MS F644
Los Alamos National Laboratory
Los Alamos, New Mexico 87544
[2] College of Engineering
New Mexico State University
Box 30001, MSC 3449
Las Cruces, New Mexico 88003

**Abstract** - Finite-element discretization of the vector wave equation is a common method of analyzing the electromagnetic field scattered by an object. One of the most challenging aspects of this research concerns the solution of the system of equations resulting from the finite-element analysis. Advanced solution algorithms have enabled researchers to generate more realistic computational models for scattering problems. The work presented here represents what is believed to be a unique parallel algorithm offering researchers a method of solving large, sparse systems of equations with advantages that are not found in previously published works.

This research uses a parallel sparse matrix decomposition algorithm to solve very large algebraic systems arising from the finite-element solution of electromagnetic scattering problems. This article provides an overview of the scattering problem and how the direct, parallel algorithm offers an efficient method of solution.

## I. INTRODUCTION

The solution of large, sparse, irregular systems of equations is an important part of many computational tasks in science and engineering. In general, such a system can be solved iteratively or directly. The former path typically utilizes a Krylov-based method with preconditioning while the latter is some variation of Gaussian elimination. The convergence of the iteration procedure depends not only on the spectral content of the coefficient matrix but also on the right-hand-side excitation as well. Convergence difficulties have been reported in electromagnetics scattering problems similar to those discussed in this research [16]. In the case of problems with many excitation vectors, Krylov methods lose their advantage over Gaussian elimination methods since the iterative procedure typically has to be repeated for each right-hand side. The majority of the parallel, sparse matrix solvers are implemented with an iterative algorithm. They have been easier to code and they work well for many applications. A number of public domain codes using iterative techniques are available for download. For more information on iterative solvers that are available for public use, see references [2-5] or see the URL `http://www.netlib.org/utk/papers/iterative-survey`.

There are a number of parallel, direct solver algorithms available for sparse irregular systems. One that is currently being distributed is the PSPASES code from the University of Minnesota [7]. PSPASES uses a Cholesky algorithm for factorization of the coefficient matrix. A similar code is also available from the IBM Watson Research Center [8]. The discussion of their algorithm and a list of results were published in [9].

SUPERLU is a sparse matrix code that is available via the Gnu Public License. SUPERLU was initially a sequential code but was recently released to run on distributed-memory computers. While results are not available for problems with multiple excitation vectors, runtimes have been published in a number of papers and conference proceedings including [10, 11]. More information can be found at the URL `http://www.nersc.gov/ xiaoye/SuperLU/`.

References [13, 14] discuss the algorithm used as a basis for the numerical linear algebra in this paper. In [13, 14] , timing results are shown for solution of electrostatic problems. These papers summarize how the original parallel Cholesky factorization algorithm was modified to allow for indefinite and numerically nonsymmetric coefficient matrices. The solution domains for these problems were quite simple. The one-way dissection graph partitioning and the parallel LU factorization algorithm used at the time were sufficient to solve these problems. When more complex geometries were discretized with tetrahedral elements however, it was apparent that these methods of solution were not adequate. One fundamental problem was the one-way dissection algorithm used to subdivide the solution domain. The algorithm was not sophisticated enough for the finite-element problem of interest and resulted in poor load balancing. In addition, the early versions of the present algorithm were only able to solve a system with a single excitation vector which defeated the purpose for using the direct solution method.

## II. COMPUTATIONAL METHODOLOGY

A parallel, computational algorithm for the direct solution of large, sparse, irregular systems of equations generated by the finite-element method as applied to partial differential equations has been developed. The resulting linear systems may be definite or indefinite but are structurally symmetric and may contain many excitation vectors. The computer code is called *mp_solve.* The *mp_solve* code is currently used as a tool by researchers in science and engineering to solve these algebraic systems resulting from finite-element discretizations of field equations in areas including bioelectromagnetics, semiconductor device modeling, fluid flow, remote sensing, electromagnetic radiation, microwave circuit simulation and scattering problems. The current implementation of the *mp_solve* algorithm has the capability to solve a linear system for many excitation vectors as is commonly done for electromagnetic scattering problems. The focus of the research presented here is the utilization of the *mp_solve* software as a tool for solving large systems resulting from the finite-element discretization of the vector wave equation. In the following section, the computational techniques used in the *mp_solve* application are described in detail.

### A. Block LU Factorization of the System of Equations

The goal of the *mp_solve* software is to solve the linear system of equations,

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

where $\mathbf{A}$ is the coefficient matrix, $\mathbf{b}$ is the excitation vector and $\mathbf{x}$ is the solution vector.

A reordering scheme can be applied to matrix $\mathbf{A}$ to obtain the border-block diagonal matrix shown in Fig. 2. The border-block diagonal system is represented by four submatrices,

$$\mathbf{A} = \begin{bmatrix} \mathbf{B} & \mathbf{V} \\ \mathbf{Z^T} & \bar{\mathbf{C}} \end{bmatrix} \tag{2}$$

where $\mathbf{B}$ represents the diagonal blocks, $\mathbf{V}$ represents the upper-right border block, $\mathbf{Z^T}$ represents the lower-left border block and $\bar{\mathbf{C}}$ is the lower-right block.

The coefficient matrix $\mathbf{A}$ can be factored into upper and lower triangular matrices,

$$\mathbf{A} = \mathbf{LU} \tag{3}$$

where,

$$\mathbf{L} = \begin{bmatrix} \mathbf{L_B} & \mathbf{0} \\ \mathbf{W^T} & \mathbf{L_C} \end{bmatrix} \tag{4}$$

and,

$$\mathbf{U} = \begin{bmatrix} \mathbf{U_B} & \mathbf{G} \\ \mathbf{0} & \mathbf{U_C} \end{bmatrix}. \tag{5}$$

The submatrices $\mathbf{L_B}$ and $\mathbf{U_B}$ and $\mathbf{L_C}$ and $\mathbf{U_C}$ are the factors of the submatrices $\mathbf{B}$ and $\mathbf{C}$, respectively. The upper and lower factors of the diagonal blocks are computed,

$$u_{i,j} = b_{i,j} - \sum_{k=1}^{i-1} l_{i,j} u_{k,j} \tag{6}$$

and,

$$l_{i,j} = \frac{b_{i,j} - \sum_{k=1}^{j-1} l_{i,j} u_{k,j}}{U_{j,j}}. \tag{7}$$

The submatrix $\mathbf{C}$ is found by computing,

$$\mathbf{C} = \bar{\mathbf{C}} - \mathbf{Z^T U_B^{-1} L_B^{-1} V}. \tag{8}$$

With a few algebraic manipulations, the modification to submatrix $\bar{\mathbf{C}}$ becomes,

$$\mathbf{Z^T U_B^{-1} L_B^{-1} V} = \mathbf{Z^T U_B^{-1} L_B^{-1} L_B G} = \mathbf{Z^T G}. \tag{9}$$

To find the columns in the matrix $\mathbf{G}$, solve the system $\mathbf{U_B G} = \mathbf{G}$ for each column $\tilde{\mathbf{g}}_\mathbf{i}$ of the matrix $\mathbf{G}$ and finally write the modifications to submatrix $\bar{\mathbf{C}}$ as,

$$\mathbf{C} = \bar{\mathbf{C}} - \mathbf{Z^T} \tilde{\mathbf{G}}. \tag{10}$$

Each column of $\mathbf{Z^T}$ and $\tilde{\mathbf{G}}$ can be computed as they are needed and no two-dimensional arrays need to be stored thus cutting down significantly on memory requirements. When the modifications are complete, the subdomain-boundary block is factored using a block-column wrapped, dense LU factorization algorithm.

### B. Partitioning the Solution Domain

Prior to solving the system of equations on the parallel computer, the computational domain must be partitioned. The mesh connectivity data produced by the mesh generator is used to produce a graph. Graph partitioning software is then employed to subdivide the graph of the elements into a specified number of subdomains. The cuts (or subdomain boundaries) are made through the graph so that approximately the same number of elements are assigned to each subdomain while trying to minimize the number of elements which fall on the subdomain boundaries. The process of cutting the graph is repeated until the desired number of subdomains is obtained.

The graph formed by the elemental graph however is not a representation of the nonzero structure of the coefficient matrix. The unknowns in the problem are associated with the edges (and faces) of the mesh. Further work is needed to determine which edges (and faces) from each element reside on each subdomain.

Once this is known, another graph is produced representing the edge (and face) connectivity for each subdomain. The adjacency data in these new graphs indicate the locations of the nonzero entries in the coefficient matrix. This process results in a nested dissection of the coefficient matrix.

Figure 1 illustrates a mesh of rectangular elements partitioned into four subdomains. The edges which fall on the subdomain boundaries are shown with thick lines. The algorithm described in this research depends on the coefficient matrix being stored in the border-block diagonal form illustrated in Fig. 2. To obtain this matrix ordering, the edges that reside on the interior of each subdomain are numbered first and those residing on the subdomain boundaries are numbered last. The interactions between edges on the interior of each subdomain form the four diagonal blocks shown in Fig. 2. The interactions between interior edges and subdomain-boundary edges form the border-blocks on the upper-right and lower-left and the interactions between subdomain-boundary edges form the darkest block on the lower-right of Fig. 2. Note that this ordering results in a matrix of the form shown in equation (2) of the previous section. Details regarding this border-block diagonal matrix ordering can be found in George and Liu [2].

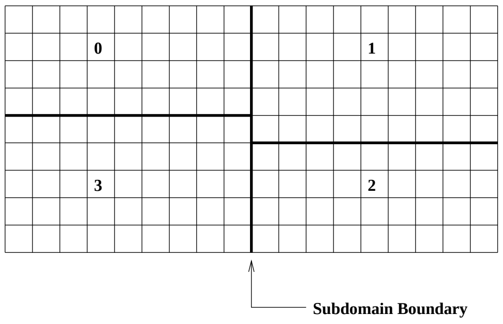

Fig. 1. Graph of solution domain partitioned into four subdomains.

## C. Implementation of the Solution Domain Partitioner

The pre-processing phase of this application consists of subdividing the finite-element mesh into the number of subdomains desired by the user. The *Chaco* (http://www.cs.sandia.gov/CRF/chac.html) graph partitioner obtained from Sandia National Laboratory is used to perform a partitioning into a specified number of subdomains. Because of the recursive partitioning of the graph that is performed by *Chaco*, the number of processors required will be $2^p$ where $p$ is the number of bisections performed by *Chaco*.

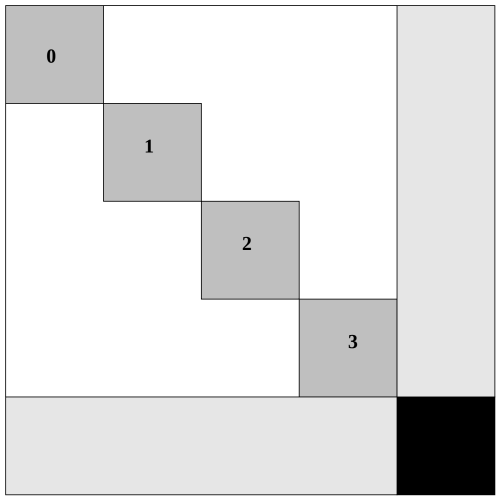

Fig. 2. Border-block diagonal system.

Alternative graph partitionings could be considered to remove the power-of-two restriction although that has not been explored in this research. Figure 1 shows the graph of a matrix partitioned into four subdomains.

The order in which the unknowns are numbered is of crucial interest for our application. One goal of the nested dissection is to obtain a border-block diagonal system (see Fig. 2) which can be distributed to the processors with the number of processors used equal to the number of diagonal blocks. Each diagonal block is mapped to a different processor. The data on the off-diagonal borders is mapped to the same set of processors depending on which processor the interior unknowns reside on. The subdomain boundary block is distributed among the same processors in a block-column wrapped format. Figure 3 illustrates the distribution of the block-columns of the subdomain boundary block to all of the processors.

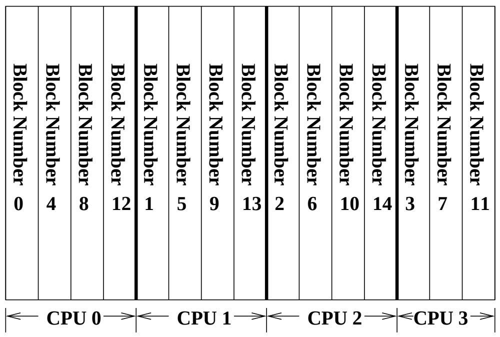

Fig. 3. Block-column distribution for the subdomain-boundary block.

## D. Implementation of the Parallel, Direct Solver Application

The primary goal of this research was to develop and implement a parallel, sparse solver for linear algebraic systems. Figure 4 shows the program flow for *mp_solve*. The grey-shaded boxes indicate operations requiring interprocessor communications. The boxes without shading indicate computations done in parallel with no interprocessor communications necessary.

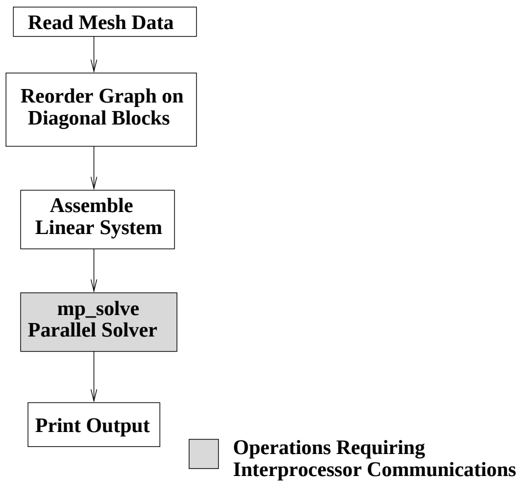
Read Mesh Data
Reorder Graph on Diagonal Blocks
Assemble Linear System
mp_solve Parallel Solver
Print Output
Operations Requiring Interprocessor Communications

Fig. 4. Flowchart of the parallel application including *mp_solve*.

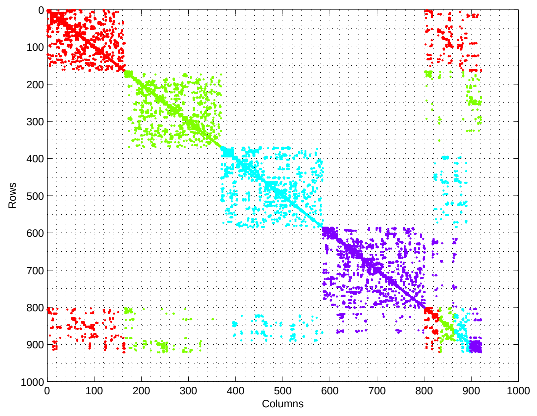

Fig. 5. Nonzero pattern of a four-subdomain border-block diagonal system.

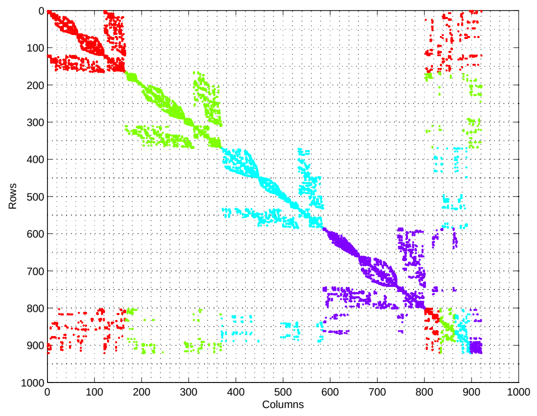

Fig. 6. Nonzero pattern of matrix shown in Fig. 5 after one-way dissection reordering for each subdomain block.

**One-Way Dissection Reordering** The first operation in this parallel application is a one-way dissection reordering of the graph representing the nonzero structure of the diagonal blocks [2]. This is the second reordering performed in the solution process; the first being a global reordering for the purpose of obtaining a border-block diagonal system. The reason for this second, local reordering on the diagonal blocks is to minimize the number of nonzeros that have to be stored in the global system. An example of the border-block diagonal system produced by the graph partitioning software prior to reordering the diagonal blocks is shown in Fig. 5. This mesh was subdivided into four subdomains hence the four diagonal blocks. Using a one-way dissection reordering, the rows in each diagonal block are permuted in such a way to move the first nonzero closer to the diagonal. This reordering minimizes the memory usage and computation time. Figure 6 illustrates the results of reordering the rows of the matrix shown in Fig. 5. The differing shades of grey shown in both figures indicate nonzero matrix entries assigned to different processors.

The graph of each diagonal block is cut into several subgraphs as determined by the reordering algorithm. The edges within each subgraph are numbered first and those edges forming the cuts are numbered last. This numbering scheme is analogous to the global reordering by the code which uses the *Chaco* graph partitioning software. This results in a border-block diagonal pattern on each diagonal block. In the small example matrix shown in Figs. 5 and 6, the total memory needed to store all of entries in the global system was reduced from $1,878,912$ bytes to $710,720$ bytes due to the one-way dissection reordering.

**System Assembly** The next function in the parallel application is the global system assembly. The matrix assembly process inserts the coefficient matrix and excitation vector(s) values in the locations specified by the one-way dissection reordering performed in the previous step. Each processor fills only the part of the global system assigned to it and no interprocessor communication is necessary.

Different parts of the global coefficient matrix are stored differently depending on the density of a particular portion of the system. First, consider the nonzero entries in each reordered diagonal block. The edges which fell on the interior of the subgraph during the one-way dissection are numbered first so they form the "diagonal" portion of each diagonal block. For each row along this diagonal, all numbers are stored starting with the first nonzero through up to the diagonal including any intervening zeros. An accompanying array of indices is used to access specific entries in the envelope. The same index array is used to locate entries in the coefficient matrix above the main diagonal since it is assumed that the matrix is structurally symmetric. More information on this storage scheme can be found in George and Liu [2].

The off-diagonal nonzeros in both the upper and lower borders of each diagonal block are stored in a compressed row-column format. These values represent interactions between edges on the interior of the subgraph and on the boundaries between adjacent subgraphs. An index array is used to keep track of the number of nonzeros in a row of the off-diagonal block and an accompanying array indicates which columns of the off-diagonal block contains the nonzero values. No zeros are stored in the border block portions of the coefficient matrix since this part of the system is very sparse. This method of storage closely follows that found in [2].

The subdomain-boundary block is shown in black in Fig. 2. This portion of the coefficient matrix is stored in a dense, block-column wrapped format. The number of the columns comprising a block is specified by the code accompanying the graph partitioning software. Since this part of the coefficient matrix is assumed to be dense, no special indexing arrays are needed to keep track of nonzeros. The only information needed for retrieving a particular block of the matrix is a global-to-local mapping to determine which processor is storing a particular block of columns. This mapping can be computed easily on each processor since it is known how many processors are in the job, the number of columns per block and the number of subdomain-boundary unknowns in the problem. Each block of columns of the subdomain-boundary matrix are distributed among all of the processors in the job in a "round-robin" ordering as shown in Fig. 3.

*mp_solve* **Software** When the linear system is assembled, the parallel solver *mp_solve* is called. The solver software is passed the coefficient matrix and the excitation vector(s) from the calling program. The *mp_solve* function returns a matrix of solutions; one column for each excitation vector. A flowchart showing the sequence of steps is shown in Fig. 7 and a detailed description of *mp_solve* is presented in the following section.

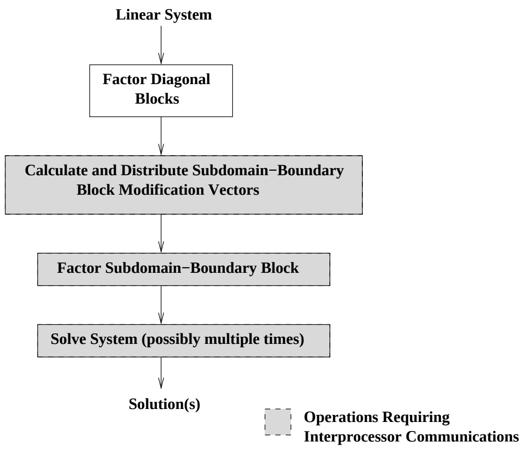
Linear System
Factor Diagonal Blocks
Calculate and Distribute Subdomain−Boundary Block Modification Vectors
Factor Subdomain−Boundary Block
Solve System (possibly multiple times)
Solution(s)
Operations Requiring Interprocessor Communications

Fig. 7. Flowchart of the *mp_solve* algorithm.

The *mp_solve* function called after the global system has been filled. It accepts the linear system as input and returns the solution vector(s). The first computation executed by *mp_solve* is the diagonal block factorization. This factorization is shown mathematically in equations (6) and (7). This step is performed in parallel with no interprocessor communications necessary.

The next operation in the *mp_solve* code is determined by the number of processors used to solve the problem. If a serial job was specified, i.e., one processor is used, then the forward and backward solves are performed and the solution is returned to the calling program. While the solution phase of the serial job takes advantage of the sparsity of the system, the operations are relatively simple and can be found in most linear algebra texts.

If multiple processors are used to solve the problem, then the solution process is more complicated. First, modifications have to be performed on the subdomain-boundary block prior to its factorization. The modification computations are outlined in equations (8) through (10). To increase the efficiency of the modification communications, a mapping of the modification vectors to their destination processors is made prior to beginning the computations. Once the communication pattern is known for each modification vector, the computations take place and the data can be sent to other processors as needed.

The next step in a multiple-processor job is to factor the subdomain-boundary block into upper and lower factors. The first part of the factorization function performs some "bookkeeping" operations so that a mapping is made of which processor stores each column and each block of columns The subdomain-boundary block factorization algorithm utilizes the BLAS-3 functions to enhance the computational performance. The blocks typically contained 32 columns apiece which was shown to run most efficiently on a Hewlett-Packard parallel server in earlier timing tests. During the factorization computations, local pivoting is performed to ensure numerical stability.

When the factorization is complete, a parallel solution function is called to perform a forward and backward solve for each excitation vector. One advantage of the direct solver is the ability to solve for many excitation vectors efficiently. The parts of the solution computations involving the diagonal blocks are performed completely in parallel but the portion of the solution involving the subdomain-boundary block requires a number of communication function calls. This is due to the fact that the dense block is stored on all of the processors in the job. Each processor performs solution operations on the diagonal blocks for one excitation vector, repeating the steps for each excitation vector. The solution operations which take place on the subdomain-boundary block handle as many excitation vectors simultaneously as there are processors in the job. Figure 8 illustrates the sequence of steps taken to perform the solve on the subdomain-boundary block for four excitation vectors.

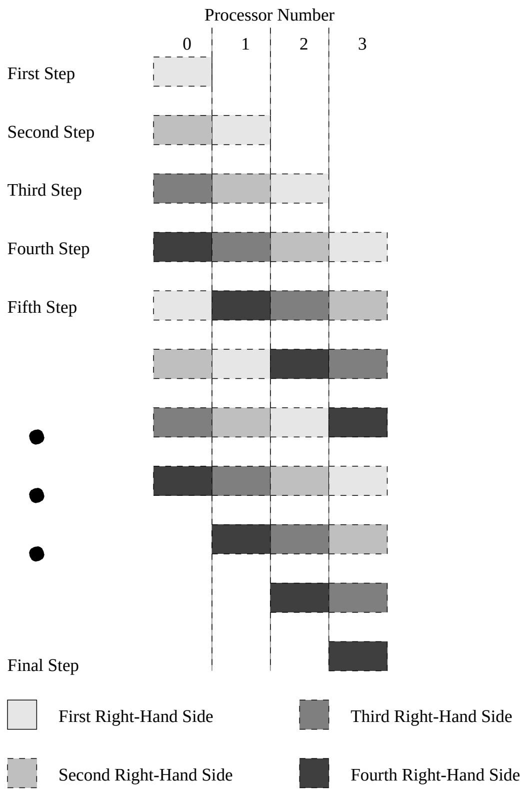

Fig. 8. Subdomain-boundary block solve.

The first step in the parallel solution on the subdomain-boundary matrix is that processor 0 starts with the solve on the portion of the matrix in its memory. When CPU 0 has done all of the computations it can on one row of the matrix, the data is passed to CPU 1 and CPU 0 starts working with the next excitation vector. When CPU 1 is done with its computations on the first right-hand side, that data is passed to processor 2, processor 1 receives the data concerning the second right-hand side and processor 0 begins working on data associated with the third right-hand side. These operations continue on the subdomain-boundary block for all four excitation vectors until this phase of the solve is complete. If there are more excitation vectors than there are processors in the job, the solution functions are called until all solutions have been computed.

## III. MATHEMATICAL DEVELOPMENT OF THE ELECTROMAGNETIC SCATTERING PROBLEM

The primary reason for the development of the *mp_solve* algorithm is for solving large systems of equations resulting from the finite-element analysis of the vector wave equation. Electromagnetic scattering problems arise when analyzing radar signature of aircraft or missiles as well as when searching for buried objects such as land mines or industrial waste. The following sections give an overview of the mathematical derivations behind the scattering problem of interest in this research.

### A. Overview of the Finite-Element Derivation

Finite-element discretization of the vector wave equation has been covered in a number of publications, therefore only a brief derivation will be shown here [3, 17]. Begin with the vector wave equation,

$$\nabla \times \bar{\bar{\mu}}_r^{-1} \times \bar{E} - \beta_o^2 \bar{\bar{\epsilon}}_r \bar{E} = 0 \tag{11}$$

and substitute $\bar{E}^s + \bar{E}^i$ for $\bar{E}$ in equation (11) to obtain the scattered field formulation,

$$\nabla \times \bar{\bar{\mu}}_r^{-1} \nabla \times (\bar{E}^i + \bar{E}^s) - \beta_o^2 \bar{\bar{\epsilon}}_r (\bar{E}^i + \bar{E}^s) = 0. \tag{12}$$

Collect the unknown scattered field on the left and the known incident field on the right, equation (12) becomes,

$$\nabla \times \bar{\bar{\mu}}_r^{-1} \nabla \times \bar{E}^s - \beta_o^2 \bar{\bar{\epsilon}}_r \bar{E}^s = \beta_o^2 \bar{\bar{\epsilon}}_r \bar{E}^i - \nabla \times \bar{\bar{\mu}}_r^{-1} \nabla \times \bar{E}^i. \tag{13}$$

Now apply the Method of Weighted Residuals using a vector weighting function $\bar{T}$ to equation (13),

$$\int_V \bar{T} \cdot (\nabla \times \bar{\bar{\mu}}_r^{-1} \nabla \times \bar{E}^s - \beta_o^2 \bar{\bar{\epsilon}}_r \bar{E}^s) dV = \int_V \bar{T} \cdot (\beta_o^2 \bar{\bar{\epsilon}}_r \bar{E}^i - \nabla \times \bar{\bar{\mu}}_r^{-1} \nabla \times \bar{E}^i) dV. \quad (14)$$

The final governing equation is,

$$-\oint_S \bar{T} \times (\bar{\bar{\mu}}_r^{-1} \nabla \times \bar{E}^s) \cdot d\bar{s} + \int_V (\bar{\bar{\mu}}_r^{-1} \nabla \times \bar{E}^s) \cdot (\nabla \times \bar{T}) dV - \beta_o^2 \int_V \bar{T} \cdot \bar{\bar{\epsilon}}_r \bar{E}^s dV = \beta_o^2 \int_V \bar{T} \cdot \bar{\bar{\epsilon}}_r \bar{E}^i dV - \int_V \bar{T} \cdot (\nabla \times (\bar{\bar{\mu}}_r^{-1} \nabla \times \bar{E}^i)) dV. \quad (15)$$

A mesh generator is used to discretize the solution domain into finite elements,

$$V = \sum_{N_e} V_e \quad (16)$$

where $N_e$ is the number of elements. On each element, the FEM approximation for the scattered field is,

$$\bar{E}^s = \sum_{j=1}^{N_u^e} \bar{N}_j^e (E_j^s)^e \quad (17)$$

where $N_u^e$ is the number of unknowns on this particular element and $\bar{N}_j^e$ is the shape function. The Galerkin Method stipulates that the weighting functions are equal to the shape functions,

$$\bar{T}_i = \bar{N}_i. \quad (18)$$

Substitute these summations into the volume integral terms on the left side of equation (15) to obtain,

$$\sum_{N_e} \sum_{j=1}^{N_u^e} (E_j^s)^e \left( \int_{V_e} (\bar{\bar{\mu}}_r^{-1} \nabla \times \bar{N}_i^e) \cdot (\nabla \times \bar{N}_j^e) dV^e - \beta_o^2 \int_{V_e} \bar{N}_i^e \cdot \bar{\bar{\epsilon}}_r \bar{N}_j^e dV^e \right), \quad i = 1, ..., N_u^e. \quad (19)$$

The excitation vectors are computed by summing the contribution of each element in the solution domain. Using the right-hand side of equation (15),

$$\sum_{N_e} \left( \beta_o^2 \int_{V_e} \bar{N}_i^e \cdot (\bar{\bar{\epsilon}}_r \bar{E}^i) dV_e - \int_{V_e} \bar{N}_i^e \cdot (\nabla \times \bar{\bar{\mu}}_r^{-1} \nabla \times \bar{E}^i) dV_e \right), i = 1, ..., N_u^e. \quad (20)$$

Substituting equations (19) and (20) into equation (15) yields a linear system of equations of the form,

$$\mathbf{Ax} = \mathbf{b}. \quad (21)$$

where $\mathbf{A}$ is the coefficient or stiffness matrix, $\mathbf{b}$ is the vector representing the discretization of the forcing function and $\mathbf{x}$ is the vector of unknowns.

The software developed for this research utilizes first- and second-order tetrahedral finite elements. The first-order tetrahedrons have one vector basis function (and hence one unknown) lying along each edge. The second-order tetrahedrons have two vector basis functions along each edge and two vector basis functions on each face for a total of twenty unknowns per element. Detailed development of the elemental matrix entries can be found in the computational electromagnetics text by Peterson [17].

## B. Mesh Termination

There are a number of methods available for terminating computational domains for electromagnetic scattering problems. Commonly-used techniques include radiation boundary conditions and integral equations terminations. In the past several years, attention has been given to ficticious materials surrounding the scatterer that causes the scattered wave to be attenuated. These are often referred to as "Perfectly Matched Layers" or PML for short. A PML layer is simple to implement and when the material parameters are chosen correctly, the PML is an effective mesh termination technique. The material parameters for each element in the PML are stored as diagonal tensors of the form,

$$\bar{\bar{\mu}}_r = \bar{\bar{\epsilon}}_r = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} \quad (22)$$

where $a$, $b$, and $c$ are complex. The outer surface of the PML is terminated by a perfect electric conductor. Further information and derivations can be found in a number of publications including [16, 18, 19].

*mp_fem* **Software** The *mp_fem* application is a parallel electromagnetic scattering code which utilizes the *mp_solve* software. The sequence of steps in the *mp_fem* code is shown in Fig. 4 with the only difference being in how the entries in the linear system are filled.

The system assembly for *mp_fem* is done on an element-by-element basis with the boundary conditions and material properties accounted for as each elemental matrix is filled. When the elemental matrix entries have been computed, the elemental matrix data is assembled into the global system. Since *mp_fem* uses the *mp_solve* software, the system storage scheme is the same as described earlier with the diagonal blocks stored in an envelope format, the off-diagonal blocks stored in a compressed row-column format and the subdomain-boundary block stored as a dense matrix. No interprocessor communication is needed during matrix assembly because the preprocessing software provides all of the elemental data needed for each processor to completely build it's portion of the global system.

Currently, the *mp_fem* software contains code for 1st- and 2nd-order, vector-based tetrahedral elements though other element types could be added. The finite-element computations for the elemental matrix entries closely follow those found in [17].

## IV. RESULTS

This section outlines the results gathered from a variety of electromagnetics problems using the *mp_fem* software. The results serve two purposes: (1) to test the efficiency of the parallel solver as a tool for solving this class of problems and (2) to verify that this software models the electromagnetic scattering problems accurately. The efficiency of the parallel algorithm is checked by running scaled and unscaled speedups. The accuracy of the electromagnetics modeling is verified by comparing the solution that *mp_fem* returns to the solution of a problem with a known result.

### A. Parallel Computer Architecture

A 256-CPU linux cluster housed at the University of Michigan was used to obtain the results shown in this section. Each node on the computer has two AMD processors with a clock speed of 2 gigahertz. Each CPU has access to one gigabyte of memory. The nodes are interconnected with 2 gigabit/second Myrinet used for interprocessor communication. The CPUs are allocated to each user during a run so that only one job runs on each CPU at a time. This eliminates the problems associated with swapping jobs among users while attempting to get reliable parallel speedup results.

### B. Parallel-Plate Waveguide

Three geometries were used to verify correct electromagnetic solutions. The first geometry of interest is a parallel-plate waveguide with one end containing PML. While this is not an electromagnetic scattering problem, this simple geometry is convenient for verifying the that the finite element operations are being done correctly and for testing the parallel performance of the code. Figure 9 illustrates the geometry for this problem.

The top and bottom plates of the waveguide, at $z = 0$ and $z = .2\ m$, are perfect electric conductor (PEC). The right end of the waveguide ($x \geq 1.2\ m$) is filled with PML material which is backed by PEC at the

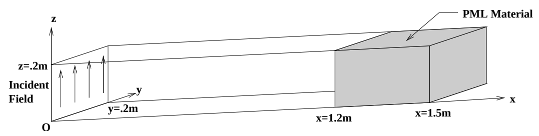

Fig. 9. Parallel-plate waveguide.

$x = 1.5\ m$ plane. The diagonal terms for the material tensor of the PML was chosen to be $1.5 - j1.5$. The natural boundary condition

$$\bar{H}^{tan} = 0 \qquad (23)$$

is employed on the sides of the waveguide in the planes $y = 0$ and $y = .2\ m$. For all of the results shown in this section, the waveguide was excited at the plane $x = 0$ by an incident field,

$$\bar{E}^i = \hat{z} E_o e^{-j\beta_x x} \qquad (24)$$

with a frequency of 300 MHz.

Several meshes of different densities were run to verify the results for first- and second-order tetrahedrons. Figure 10 shows the magnitude and phase for the wave propagating down the parallel plate waveguide. This particular mesh has 360 first-order tetrahedrons and 682 edges. The edge length is specified in the mesh generator to be 0.083333333 $m$ or approximately 12 edges per wavelength.

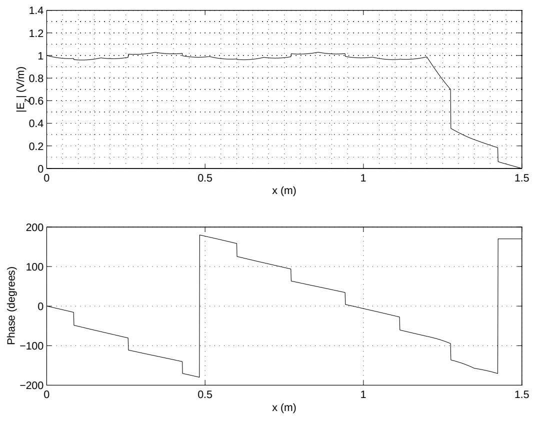

Fig. 10. Magnitude and phase plots for the first-order mesh, parallel-plate waveguide.

Figure 11 illustrates the magnitude and phase results using second-order tetrahedrons in the parallel-plate waveguide. The edge length was again specified to be 0.083333333 $m$ and, like the first-order mesh, there are 360 elements. However, due to the fact that there are two basis functions for each edge and face in the mesh, this problem has 3108 unknowns.

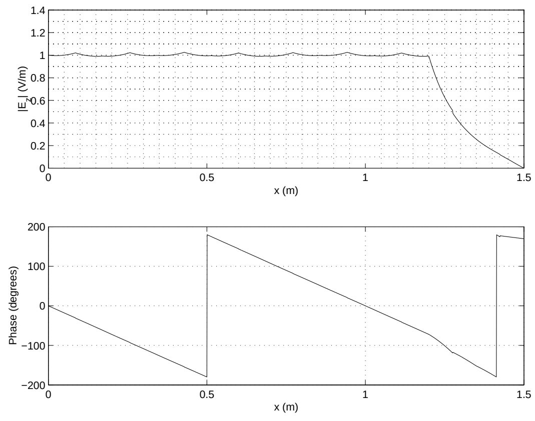

Fig. 11. Magnitude and phase plots for the second-order mesh, parallel-plate waveguide.

Several mesh densities were tested for both first- and second-order meshes on the parallel-plate waveguide in order to draw conclusions on accuracy. For both element orders, mesh densities from six through twenty edges per wavelength were run. The average magnitude and phase error was computed from each of a 1000 points longitudinally along the center of the waveguide. For first-order elements at a mesh density of six edges per wavelength, the magnitude error was approximately 1.5% and the phase error was approximately 40%. These numbers decreased, respectively, to .28% and 2% at a mesh density of twenty edges per wavelength. The second-order elements performed considerably better as expected. At a mesh density of six edges per wavelength, the error in both the magnitude and phase was approximately 1%. At a mesh density of twenty edges per wavelength, the error decreases to about 0.05% for the magnitude and to 0.1% for the phase.

**Unscaled Speedup** The first test of the scalability of the *mp_fem* algorithm is the unscaled speedup. In this test, a problem of a fixed size is run on an increasing number of processors. However, as the number of subdomains is increased, the amount of time spent on operations involving the subdomain-boundary block increases so dramatically that there is no longer a benefit to using more CPUs.

The first-order mesh using the parallel-plate waveguide geometry was run using a fixed problem size of 72,330 tetrahedral elements containing 81,249 edges. The first test was run with one processor and increased by a power-of-two until 32 CPUs were used to solve the problem. The runtimes for the 32-CPU job indicate that partitioning this mesh into more subdomains did not decrease the runtime so no further tests were conducted. Figure 12 illustrates the runtimes in seconds for all of the jobs in this test.

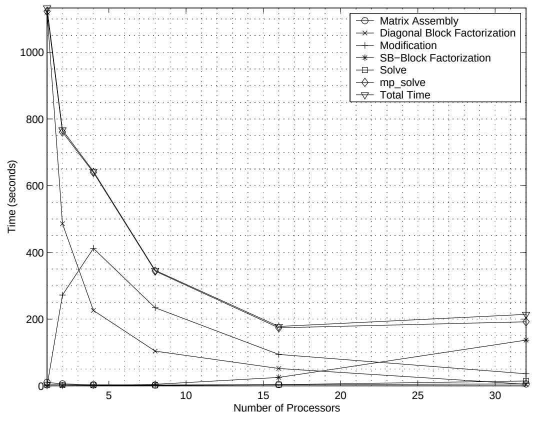

Fig. 12. Wall-clock times for fixed problem size, 1st-order mesh, parallel-plate waveguide.

A similar unscaled timing test was run on the waveguide geometry using second-order tetrahedrons. This mesh is constructed of 10,422 elements which are composed of 74,322 edges and faces. Similar to the first-order unscaled test, this problem was run on one CPU and then the problem was decomposed by a power of two until 32 processors were used. Because the total runtime was increasing when 32 processors were used, no further unscaled tests were done. Figure 13 illustrates the runtimes in seconds for all of the jobs in the test.

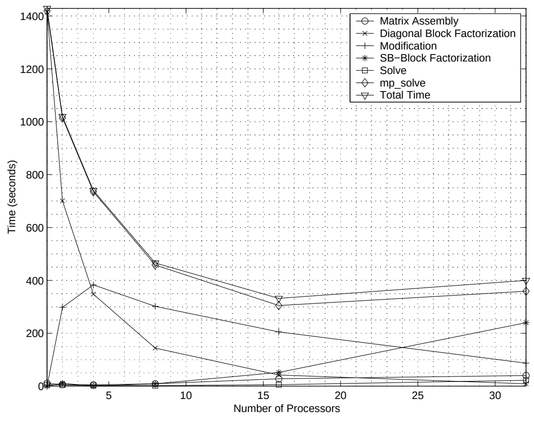

Fig. 13. Wall-clock times for fixed problem size, 2nd-order mesh, parallel-plate waveguide.

The scaling problems appear when modifying and factoring the subdomain-boundary block on multi-processor runs. Both of these operations require extensive interprocessor communications. The modification portion of the solver executes as many forward and backward solves as there are subdomain-boundary unknowns. In addition, interprocessor communications are necessary to send the modification vector to the appropriate processor to execute the subtraction in equation (10). For both orders of finite elements, the modifications performed worst on four processors and then proceeded to have a shorter runtime as more processors were added. As the size of the diagonal blocks decreased, the execution time for the forward and backward solves decreased substantially. While the number of interprocessor communications increased, the time required to perform the communications did not offset the gains made by the faster solve times. This results in a net improvement in the modification time as the number of processors in the job increases.

The subdomain-boundary block factorization time increases for both first- and second-order elements. This is a result of having a larger subdomain-boundary block as the solution domain is subdivided more times. In addition, increasing the number of CPUs increases the interprocessor communication needed to carry out the factorization. For jobs running on 16 or fewer processors, the subdomain-boundary block factorization time remains under 15% of the total runtime. However, on jobs distributed over 32 CPUs, the factorization time increase to over 50% of the total runtime.

Additional unscaled speedups were performed on larger waveguide problems to confirm performance data. Tests were done so that there were approximately the same number of unknowns for both 1st- and 2nd-order tetrahedral meshes. The 1st-order mesh had 477,274 edges requiring a minimum of 16 processors. Tests were also run on this mesh for 32 and 64 subdomains. Because the runtime increased on 64 processors, no further tests were run. The results are shown in Fig. 14.

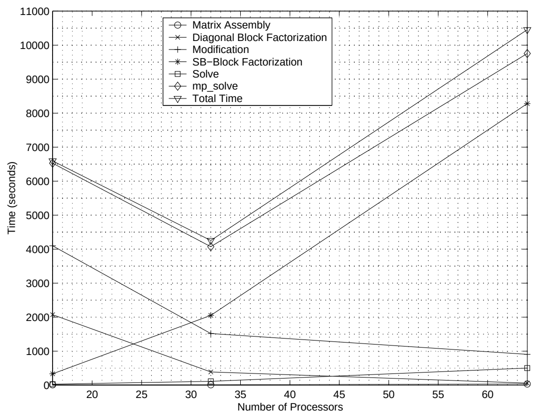

Fig. 14. Wall-clock times for fixed problem size, 1st-order mesh, parallel-plate waveguide (477,274 unknowns).

A similar test was done on a large parallel-plate waveguide geometry using 2nd-order elements. This mesh resulted in 469,192 edges and faces. Due to memory limitations, the problem could not be run on fewer than eight processors. This mesh was not successfully run on 64 processors because the subdomain-boundary block became too large and the computers operating system started to swap memory with disk space. Figure 15 illustrates the timings for the 2nd-order waveguide runs.

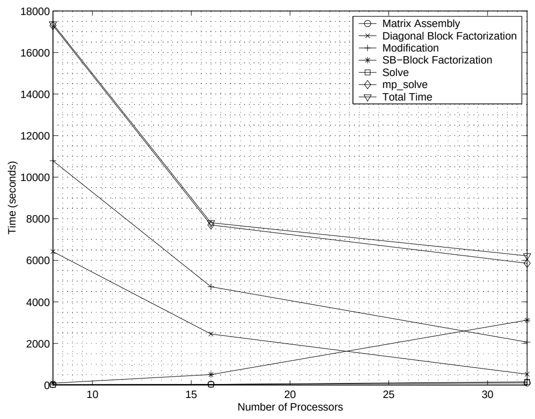

Fig. 15. Wall-clock times for fixed problem size, 2nd-order mesh, parallel-plate waveguide (469,192 unknowns).

Another way to examine the unscaled speedup is to consider the decrease in the runtime as the number of processors increase. Ideally, using two processors on a job should result in a runtime of half of that needed for solving the same problem on one processor. Likewise, running a problem on four processors should result in a runtime of one quarter of that required if the same problem was assigned to only one CPU. Figure 16 illustrates the unscaled speedup for the small first- and second-order parallel-plate waveguide results shown in Fig. 12 and 13.

Note that the plots in Fig. 16 for both 1st- and 2nd-order elements, the speedup obtained from the *mp_fem* software differs greatly from the optimal unscaled speedup. For instance, in the 1st-order mesh, as the number of processors is increased from 1 CPU to 2 CPUs, ideally, the CPU time should divide in half. The speedup obtained through from the parallel software is only 0.66. The gain for 2nd-order problems is even worse with a speedup of only about 0.71 when spreading the problem over two processors.

There are a couple of likely causes in this lack of efficiency. The first problem is load imbalance (discussed below). The other problem is a lack of scalability in the portions of the code dealing with the subdomain-boundary block. In both unscaled speedup tests of the waveguide problem, the factorization of the subdomain-boundary block increases as the num-

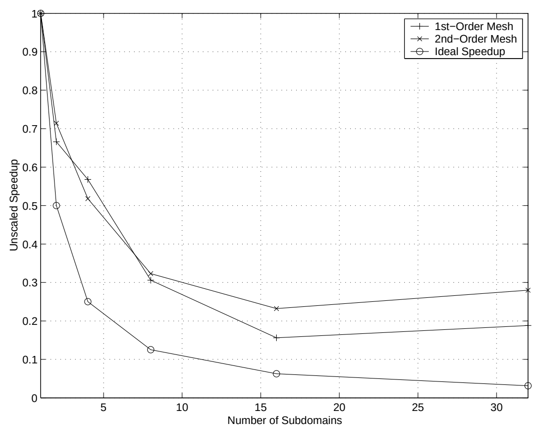

Fig. 16. Unscaled speedup for the 1st- and 2nd-order meshes, parallel-plate waveguide.

ber of subdomains increase. While the other operations such as the diagonal-block factorization and the modification of the subdomain-boundary block do not scale optimally, their decreasing runtimes do not have the impact on the overall runtime that the factorization of the subdomain-boundary block has. A torus-wrapped ordering for the subdomain-boundary block shows promise for improving scalability in this part of the code [20].

**Scaled Speedup** Another measure of parallel efficiency is the scaled speedup. In this series of runs, the problem size is increased proportionally to the number of processors in a job; i.e. if the problem size is doubled, then the number of processors is also doubled. Ideally, the execution time for each problem in the series of runs should remain constant no matter how large the problem. Due to increased communication costs as more CPUs are added to each run, the execution time is rarely ever constant.

The first scaled tests that were run involved the parallel-plate waveguide geometry meshed with first-order tetrahedrons. Figure 17 shows the scaled speedup for the first-order tetrahedrons. Each processor in each step in the test sequence was loaded with approximately 20,000 edges. Table 1 shows the problem sizes for each of the runs used for the first-order scaled speedup tests.

The second scaled speedup tests involved the same geometry but used the second-order elements. Figure 18 shows the scaling that was achieved in this sequence of runs. Table 2 shows the size of the problem for each run.

The diagonal-block factorization is a parallel operation that requires no inter-processor communication. For the scaled speedups, there were approximately the same number of edges on each CPU so the factorization time was kept relatively constant as shown in both Figs. 17 and 18.

The modification of the subdomain-boundary block requires considerable communication as well as many

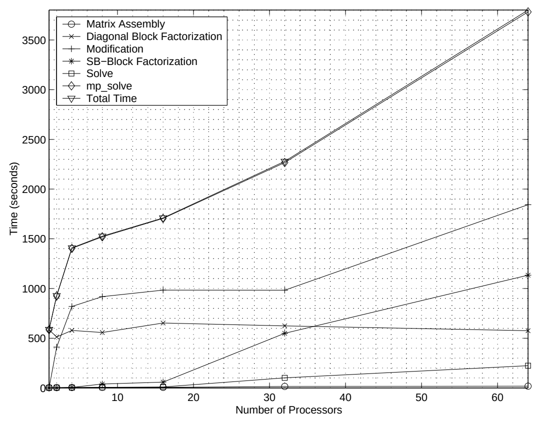

Fig. 17. Wall-clock times for scaled problem size, 1st-order mesh, parallel-plate waveguide.

Table 1. Problem sizes for the first-order scaled speedup tests.

| *Number of Processors* | *Number of Elements* | *Number of Unknowns* |
|---|---|---|
| 1 | 15,000 | 21,500 |
| 2 | 30,000 | 39,980 |
| 4 | 60,000 | 79,640 |
| 8 | 120,000 | 158,960 |
| 16 | 240,000 | 317,600 |
| 32 | 480,000 | 634,880 |
| 64 | 960,000 | 1,269,440 |

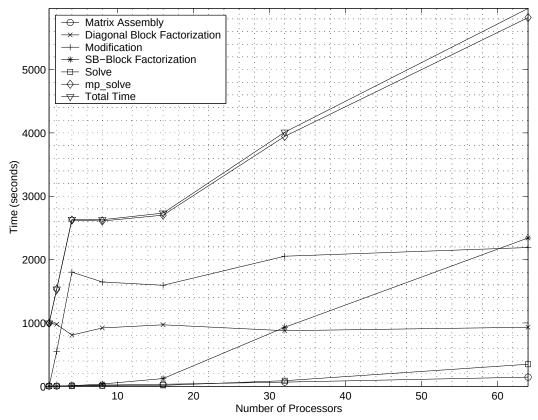

Fig. 18. Wall-clock times for scaled problem size, 2nd-order mesh, parallel-plate waveguide.

Table 2. Problem sizes for the second-order scaled speedup tests.

| *Number of Processors* | *Number of Elements* | *Number of Unknowns* |
|---|---|---|
| 1 | 2,880 | 20,768 |
| 2 | 5,940 | 42,426 |
| 4 | 11,880 | 84,468 |
| 8 | 23,760 | 168,552 |
| 16 | 47,520 | 336,720 |
| 32 | 95,040 | 673,056 |
| 64 | 190,080 | 1,345,728 |

forward and backward solves. The size of the diagonal blocks are kept nearly constant so the forward and backward solve times increase only slightly. This increase is due to that fact that with more processors in the job, more interprocessor communication is necessary to perform the forward and backward solves in the subdomain-boundary block.

The subdomain-boundary block factorization time increases as the number of unknowns in that part of the matrix increases. As the number of CPUs and the size of the subdomain-boundary block increases, the computation time and the communication time increase. This results in the rising curve shown in the subdomain-boundary block factorization time in both Figs. 17 and 18.

Another way of analyzing the scaled speedup data is to consider parallel efficiency. Hennigan defines the scaled speedup as,

$$S_s = p \times \frac{T(1,N)}{T(p,pN)} \tag{25}$$

where $N$ is the number of unknowns in the system and $p$ is the number of processors in the job [3]. Hennigan then goes on to define the parallel efficiency as,

$$\eta_p = \frac{S_s}{p}. \tag{26}$$

Figure 19 illustrates the parallel efficiency of the *mp_fem* algorithm for both 1st- and 2nd-order elements in the parallel-plate waveguide geometry. The runtimes shown in Figs. 17 and 18 were used to calculate the efficiency.

For a perfect scaled speedup, the efficiency is 1.0 for any number of processors and the corresponding problem size. Due to load imbalances and interprocessor communications however, the efficiency is rarely ever optimal. These problems along with the poor scalability of the subdomain-boundary block modification and factorization result in efficiencies that are less than one and continue to get worse as more processors are added to the job.

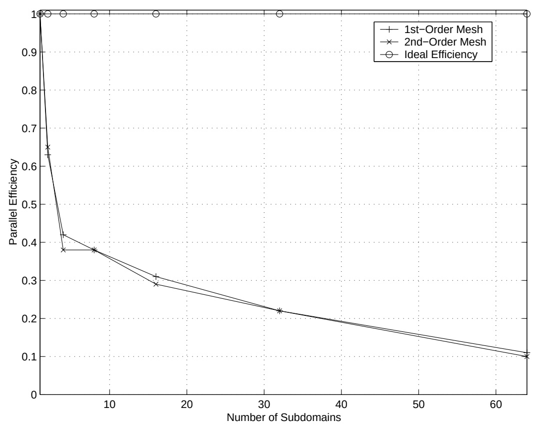

Fig. 19. Parallel efficiency for the 1st- and 2nd-order tetrahedral meshes.

**Load Balance** Obtaining a good load balance across all of the processors in a job is a primary concern of the graph partitioning software. Good load balancing means that each CPU will have the same number of computations so that no processor is sitting idle while other processors continue to work. Poor load balancing has a direct effect on parallel performance and can cause the parallel efficiency of an algorithm to be reduced considerably. While it is not possible to show here the load balance for every problem, a few decompositions are given.

The four-subdomain decomposition used in the unscaled speedup for the first-order element waveguide simulations is a matter of interest. Table 3 shows the load balancing results and diagonal-block factorization times for this decomposition.

Table 3. Load balance for four-subdomain decomposition of parallel-plate waveguide.

| *Processor Number* | *Number of Equations* | *Envelope Size* | *Memory Usage (MB)* |
|---|---|---|---|
| 1 | 19,764 | 2,978,154 | 93.3 |
| 2 | 20,030 | 2,868,948 | 95.5 |
| 3 | 20,292 | 3,675,486 | 121.3 |
| 4 | 20,489 | 3,026,150 | 100.5 |

It is evident from the table that while the number of equations on each processor vary by less than 3%, the envelope size varies by almost 22%. Since the speed of the factorization of the diagonal blocks depend on the size of the envelope, the processor with the largest envelope size in this problem requires 1.6 times the execution time to factor the matrix on its memory than the processor with the smallest envelope. For this problem, processor 2 sat idle for 135 seconds while processor 3 finished factoring its portion of the matrix.

This idle time accounts for approximately 18% of the total runtime for this problem. While only the discrepancies in the diagonal-block factorization times are shown here, modification of the subdomain-boundary block is also negatively effected by poor load balancing. The modification to the subdomain-boundary block requires many calls to a linear solve function that is dependent on the envelope size (see equations (9) and (10)). When the size of the envelope differs greatly among the processors, the time to perform these solves differs resulting in some processors sitting idle. The factorization of the subdomain-boundary block is not affected by load imbalance since the columns are distributed in a block-column wrapped manner to all of the processors in the job. Any differences in the number of columns assigned a processor has a negligible impact on the factorization time.

The load imbalance illustrated here points to the need for more thorough investigation into graph partitioning software issues as well as the need to explore alternatives to using the one-way dissection reordering on the diagonal blocks.

## C. Perfectly Conducting Cylinder

The first electromagnetic scattering problem of interest is an infinite perfectly conducting cylinder. This is a standard "textbook" problem with a well-known series solution composed of Bessel and Hankel functions.

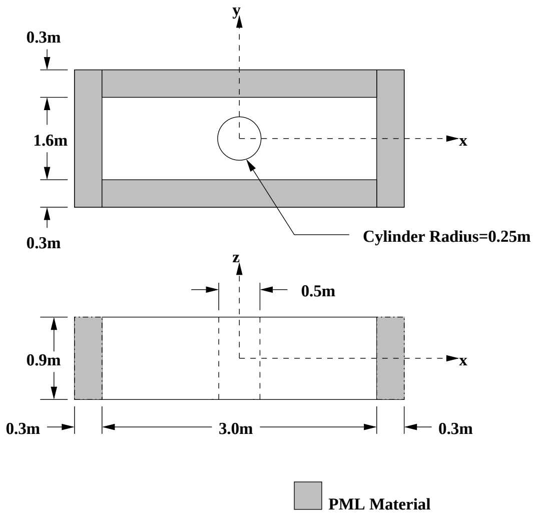

Fig. 20. Perfectly conducting cylinder.

The geometry consists of a box containing a cylinder with a diameter of 0.5 $m$. The center of the cylinder lies at the origin (see Fig. 20). The interior of the box is 3.0 $m$ long so as to capture the standing wave in the incident ($-x$ direction) and the shadow ($+x$ direction) regions of the solution domain. A layer of artificial absorber (PML) 0.3 $m$ thick is placed at the edges of the box surrounding the cylinder. The PML regions are shown in grey. The top and bottom planes of the box, at $z = -0.45$ $m$ and $z = +0.45$ $m$, are covered in PEC.

For the tests run in this section, the cylinder was illuminated by an incident plane wave of the form,

$$\bar{E}^i = \hat{z} E_o e^{-j\beta_x x} \tag{27}$$

at a frequency of 300 MHz.

Two initial tests were done with first- and second-order tetrahedral elements to ascertain that the physics is being modeled accurately. The first-order test was run with 66,213 first-order elements and 86,824 edges. The electric field was sampled in two directions shown by the heavy dashed lines in Fig. 21. The scale in Fig. 21 is the same as that shown in Fig. 20.

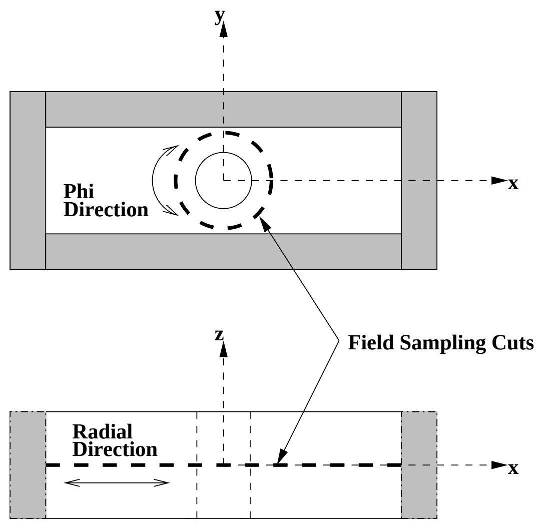

Fig. 21. Field sampling lines on the PEC cylinder.

The field in the radial direction is computed at 1000 places along the x-axis of the box containing the cylinder. The magnitude and phase components of the total field are shown in Fig. 22. The solid line is the solution computed using the *mp_fem* software while the dashed line represents the series solution.

The field in the $\phi$ direction is computed at 360 angles (one-degree increments) in a constant radius centered about the cylinders axis. The field points are computed at constant distance of 0.4 $m$ from the center of the cylinder. The magnitude and phase components of the total field are shown in Fig. 23. The solid line is the solution computed using the *mp_fem* software while the dashed line represents the series solution.

There are a couple of important considerations when determining how dense to make a mesh. The first issue is that the electric field has to be modeled correctly.

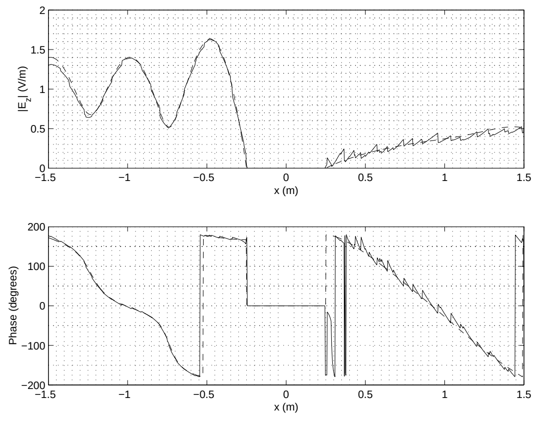

Fig. 22. Magnitude and phase plots for the first-order mesh, cylindrical scatterer (radial direction).

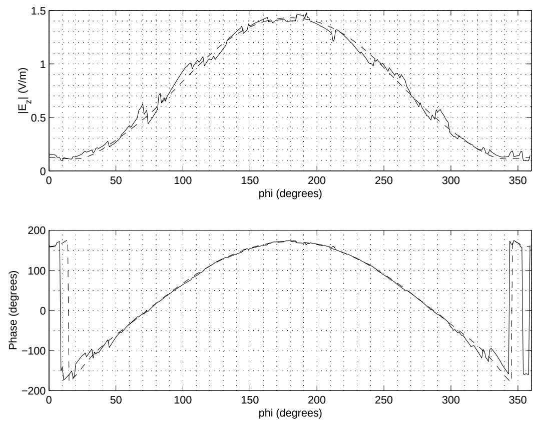

Fig. 23. Magnitude and phase plots for the first-order mesh, cylindrical scatterer (phi direction).

This issue was addressed with the parallel-plate waveguide and it was found that reasonably accurate answers were obtained using 12-14 edges per wavelength. Unfortunately, while a mesh with this density produces good answers on a rectangular geometry, it does not accurately model the curvature of the cylinder. The mesh generator used in this research allows a person to choose a different mesh densities in different places in the computational domain. For results shown in Figs. 23 through 25, a mesh density of 16 edges per wavelength was used near the surface of the cylinder while along the edges of the PML box, a mesh density of about 12 edges per wavelength was used.

Figure 24 shows the total field along the x-axis for the second-order mesh. The solid line is the field computed by the *mp_fem* software and the dashed line is the field computed using the series solution. In many places on the plot, they are indistinguishable.

Figure 25 shows the total field in a circle around

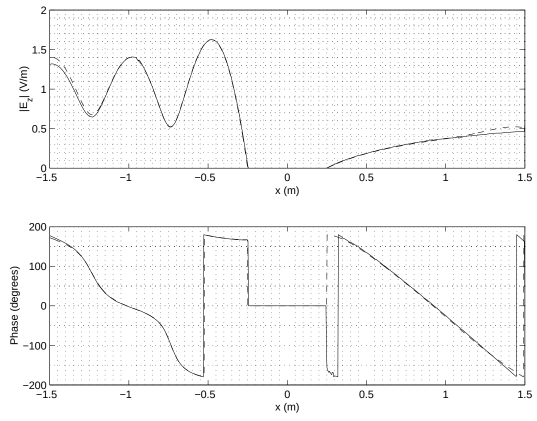

Fig. 24. Magnitude and phase plots for the second-order mesh, cylindrical scatterer (radial direction).

the cylinder at a distance of 0.4 $m$ away from the center. Again, the solid line is the field computed by the *mp_fem* software and the dashed line is the field computed using the series solution. On both plots, these lines are generally superimposed.

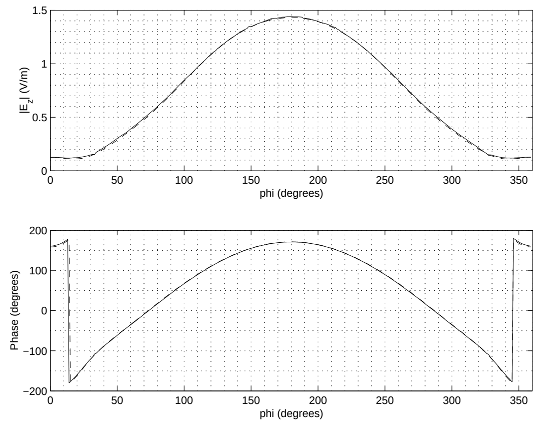

Fig. 25. Magnitude and phase plots for the second-order mesh, cylindrical scatterer (phi direction).

**Multiple Excitation Vectors** One of the strengths of a direct solver is the ability to amortize the high matrix factorization costs over the time required to perform a large number of solves for multiple excitation vectors. In this finite-element application, each new excitation vector represents a new angle from which the incident field illuminates the scatterer.

Two meshes were run for the purpose of obtaining the runtime statistics for multiple right-hand sides. The first-order mesh consisted of 125,000 tetrahedral elements containing 175,877 edges surrounding a cylindrical scatterer. The incident waves illuminated the cylinder from 360 angles at 1° increments. The time needed to solve for each excitation vector is shown in Fig. 26. In each of the three decompositions shown for first-order elements, the total solution time for 360 excitation vectors remained less than 40% of the overall wall-clock runtime for the *mp_fem* code.

The second-order mesh had approximately 28,000 elements containing 201,890 unknowns. The cylinder was illuminated at 1° increments from 360 angles. The time for each solve is shown by the upper line in Fig. 26. The percentage of the total runtime spent on solving for each excitation vector increased from approximately 26% on the 8-subdomain decomposition to 41% on the 32-subdomain decomposition. This increase in the percentage of runtime spent on the solve is primarily a reflection of the decrease in the overall runtime since the time required to solve for each right-hand side increases only slightly between 16− and 32−subdomain decompositions.

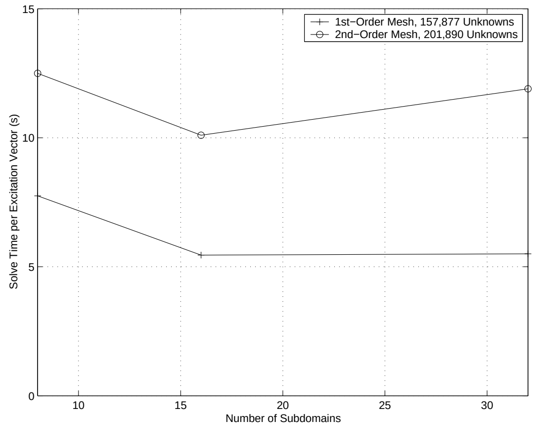

Fig. 26. Wall-clock times for each solve for multiple excitation vectors, 1st- and 2nd-order meshes, cylindrical scatterer.

## D. PEC Sphere

The second electromagetic scatterer of interest is the perfectly conducting sphere. Similar to the last problem, electromagnetic scattering from a sphere has a known series solution. The geometry for this problem is shown in Fig. 27. The PEC sphere is placed in a box with the center of the sphere at the origin. Unlike the cylindrical scatterer which can be modeled in two dimensions, the sphere is a true three-dimensional scattering problem and therefore the scatterer must be surrounded on all sides by absorbing material. The shaded areas in Fig. 27 around the sphere represent the PML absorber.

The interior of the box is 3.0 $m$ long so as to capture the standing wave in the incident region ($-x$ direction) and the shadow ($+x$ direction) region of the solution domain. The sphere has a diameter of 0.5 $m$. A layer of artificial absorber (PML) 0.3 $m$ thick is placed at the edges of the box surrounding the sphere. The PML regions are shown in grey in Fig. 27. For this set of results, the sphere was illuminated by the incident field,

$$\bar{E}^i = \hat{z}E_o e^{-j\beta_x x} \tag{28}$$

at a frequency of 300 MHz.

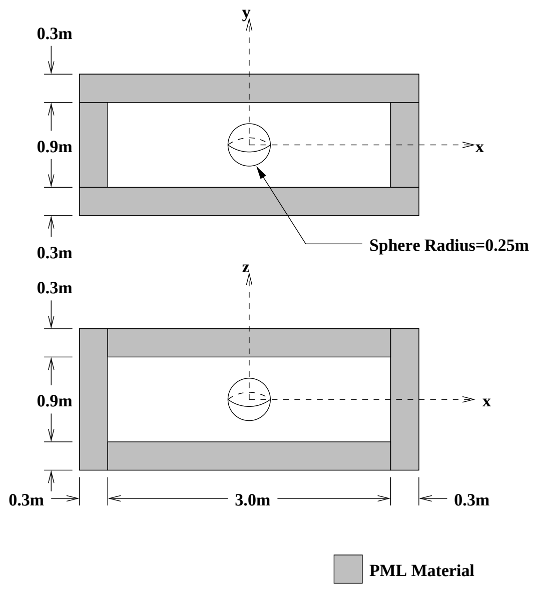

Fig. 27. Perfectly conducting sphere.

After the runs were completed, the magnitude and phase computations for the total field were performed at 1000 points along the bold, dashed line in the radial direction shown in Fig. 28. In addition, another set of field computations were taken in the $\phi$ direction at 360 angles.

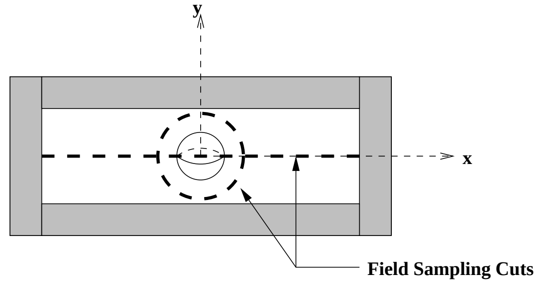

Fig. 28. Field sampling line on PEC sphere geometry.

Two meshes were run to verify that the code was correctly modeling the electromagnetic scattering problem. The first problem was a first-order mesh with

284,092 elements and 350,109 edges. The average edge length was approximately 32 edges per wavelength.

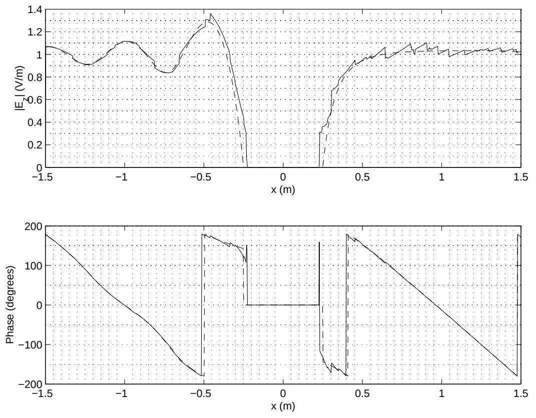

Fig. 29. Magnitude and phase plots for the 1st-order mesh, spherical scatterer (radial direction).

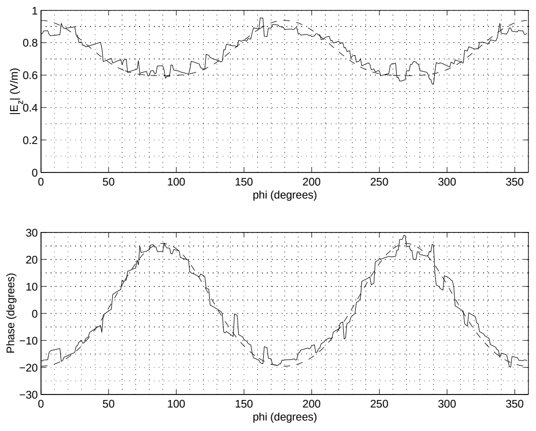

Fig. 30. Magnitude and phase plots for the 1st-order mesh, spherical scatterer (phi direction).

The solid lines in Figs. 29 and 30 are the fields in the radial and phi directions computed using the results of the *mp_fem* software. The dashed line in both plots are the field values computed using the Legendre solution. It is evident from the figures that the results obtained from the *mp_fem* software are very close to the series solution.

The second verification mesh for this geometry involved 2nd-order elements. The mesh density was set to 8 edges per wavelength resulting in 37,161 elements and 244,452 unknowns. The results for this run are shown in Fig. 31.

Similar to previous plots, the solid line is the field pattern computed using the *mp_fem* software and the dashed line is the field pattern computed using the series solution. Clearly, using a mesh with only eight

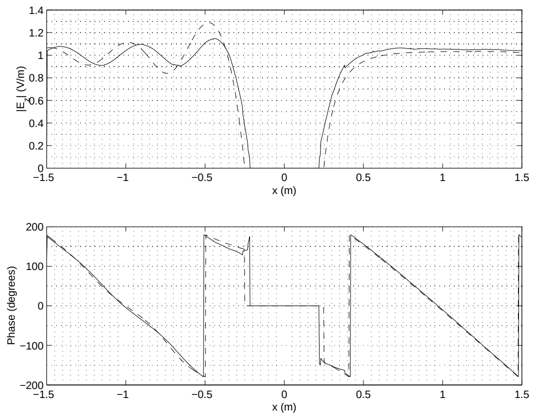

Fig. 31. Magnitude and phase plots for the 2nd-order mesh, spherical scatterer (radial direction).

edges per wavelength is not sufficient to model the spherical geometry. While the phase plot is very accurate, this coarse mesh does not resolve the field well in the incident side of the sphere. A considerable amount of computational resources were required to obtain this solution. The problem was run on 16 CPUs and required approximately 12 GB of memory. For comparison purposes, a 1st-order mesh with a density of 8 edges per wavelength requires about 750 MB of memory. Due to memory limitations, no other 2nd-order meshes were run with the spherical scatterer.

## V. CONCLUSIONS

The purpose of this research was to develop a parallel, direct solver and use it as a tool for use for electromagnetic scattering simulations. While the use of the finite-element method (FEM) is well known and has been documented extensively, the simulations of interest to researchers are often limited by the lack of memory and processing power. One goal of this research was to develop a parallel software application which would alleviate some of the constraints imposed by other available software and to include some capabilities not found elsewhere. While it is recognized that the scaling properties of the *mp_solve* software are not optimal, it has shown to be a useful tool in solving common problems of interest to the electromagnetics research community.

### A. The Parallel Solver

This research revolves around the parallel solver which was specifically developed for use with the scattering problems of interest. Several features were built into this code which are not found in other software packages including the ability to handle numerically unsymmetric matrices, complex numbers and multiple excitation vectors. Local pivoting is also included in the subdomain-boundary block factorization function to help improve numerical stability.

As was illustrated in the results section of this dissertation, the parallel algorithm scaling is less than ideal as the problem size or the number of processors are increased. A more sophisticated approach to the partitioning problem might produce better load balances with certain geometries. In addition, the simple block-column LU factorization used here had a severe impact on the runtime and on the parallel scaling.

While poor scaling is a serious drawback to any parallel algorithm, the *mp_solve* software remains a useful tool. The *mp_solve* software has been used to solve linear systems containing up to 1.35 million unknowns in just over an hour on 64 processors. An engineer or scientist can expect to use this software to solve large problems in a reasonable time.

### B. The Electromagnetic Scattering Simulation Software

Software has been developed to find solutions to the vector wave equation using the finite-element method. While this is a well-documented area of research in the electromagnetics community, a few words are in order here concerning the results.

The two orders of vector tetrahedral elements used offered both advantages and disadvantages. Code development for the first-order elements was relatively easy and the answers were found to be reasonably accurate; often within 10% of the accepted solutions. In addition, the memory requirements were considerably less than those required by the second-order elements.

If more accuracy is necessary, the second-order elements offer a substantial improvement for a given mesh density. Several of the scattering problems discussed in the previous section had accuracies within 1% of the accepted solutions. This increase in accuracy does not come without cost however. One problem with second-order elements is the memory required for a given mesh density. Experience in this research has shown that the second-order elements require seven to ten times more memory for a particular mesh density than the first-order elements. The memory consumption associated with the second-order elements limited the tests that could be run with this code. This problem was particularly evident when using second-order elements to mesh curved surfaces.

Another aspect of this research which warrants comment is the use of the Perfectly Matched Layer (PML) absorbing boundary condition. The PML was proven to be quite effective as shown by the accuracy of the results. While this research did not focus on finding optimal values for the PML material parameters, those used by Hennigan worked well in the scattering problems shown in the results section [3]. The PML domain termination conditions are easy to implenent and avoid the need for the dense matrix computations associated with the boundary-element method of solution domain termination. They also did not require the spherical terminating surface necessitated by the radiation boundary conditions.

## VI. ACKNOWLEDGEMENTS

Several government agencies and private companies contributed financial support and provided access to the computers necessary to see this project through. The National Aeronautical and Space Administration (NASA) sponsored this research through a Graduate Research Fellowship, Grant #5-50117 and through NASA/JPL FAR Grant #959913. CPU time was provided for this work by the Maui High Performance Computing Center (MHPCC), Start-up Grant #UNIVY-0279-U00 and by the National Partnership for Advanced Computing Infrastructure at the University of Michigan. Dr. Abijit Bose of The University of Michigan was instrumental in helping me obtain access to their parallel computer. The Hewlett-Packard Corporation provided a parallel server at no cost to our department. Much of the code used in this research was developed on this machine.

## References

[1] D. B. Davidson, “Large, parallel processing revisited: a second tutorial,” *IEEE Antennas and Propagation Magazine*, vol 34, pp. 9-21, 1992.

[2] A. George and J. Liu, *Computer Solution of Large, Sparse Positive Definite Systems*, Englewood Cliffs, N.J.: Prentice-Hall, 1981.

[3] G. L. Hennigan, *Open-Region Electromagnetic Finite-Element Scattering Calculations in Anisotropic Media on Parallel Computers*, Doctoral Dissertation, New Mexico State University, 1999.

[4] Y. Saad, A. Malevsky, G. C. Lo, S. Kusnetsov, M. Sosonkina, and I. Moulitsa, *A Portable Library of Parallel Sparse Iterative Solvers*, University of Minnesota, 1999.

[5] S. Balay, W. Gropp, L. C. McInnes, and B. Smith, *The Portable, Extensible Toolkit for Scientific Computation*, Argonne National Laboratory, 1999. URL:http://www-fp.mcs.anl.gov/petsc

[6] R. Tuminaro, J. Shadid, S. Hutchinson, L. Prevost, and C. Tong, *A Massively Parallel Iterative*

*Solver Library for Solving Sparse Linear Systems*, Sandia National Laboratory, 1999. URL: http://www.cs.sandia.gov/CRF/aztec1.html

[7] A. Gupta, M. Joshi, G. Karypis, V. Kumar, and F. Gustavson, *Scalable Parallel Direct Solver Library for Sparse Symmetric Positive Definite Systems*, University of Minnesota, 1999. URL: http://www-users.cs.umn.edu/mjoshi/pspases/index.html #people

[8] A. Gupta, M. Joshi, and V. Kumar, *Watson Symmetric Sparse Matrix Package*, IBM Watson Research Center, 1999. URL: http://www.research.ibm.com/mathsci/ams /ams_WSSMP.htm

[9] A. Gupta, M. Joshi, and V. Kumar, "A Highly Scalable, Parallel Algorithm for Sparse Matrix Factorization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 5, pp. 502-520, May 1997.

[10] J. Demmel and X. Li, "Making Sparse Gaussian Elimination Scalable by Static Pivoting," *Proceedings of Supercomputing '98*, Orlando, Florida, November 1998.

[11] X. Li and J. Demmel, "A Scalable Sparse Direct Solver Using Static Pivoting," *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[12] J. Demmel, S. Eisenstat, J. Gilbert, X. Li, and J. Liu, "A Supernodal Approach for Sparse Gaussian Elimination," *Technical Report UCB//CSD-97-943*, Computer Science Division, U.C. Berkeley, 1997.

[13] W. Dearholt, S. Castillo and G. L. Hennigan, "Solution of Large, Sparse, Irregular Systems on a Massively Parallel Computer," Third International Workshop, IRREGULAR '96, Santa Barbara, CA, August 1996, *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1996.

[14] W. Dearholt, S. Castillo and V. Bokil, "Solution of Large, Sparse Systems on a Parallel Computer," Conference Proceedings, *IEEE Antennas and Propagation Conference*, Baltimore, MD. July 1996.

[15] M. Hagger and L. Stals, *Domain Decomposition on Unstructured Grids*, Department of Mathematics, University of Bath, Claverton Down, Bath, BA2 7AY, England, URL: http://www.maths.bath.ac.uk/parsoft/doug

[16] J. Tang, K. Paulsen, and S. Haider, "Perfectly Matched Layer Mesh Terminations for Nodal-Based Finite-Element Methods in Electromagnetic Scattering," *IEEE Transactions on Antennas and Propagation*, vol. 46, pp. 507-517, April 1998.

[17] A. Peterson, S. Ray, and R. Mittra, *Computational Methods for Electromagnetics*, IEEE Press. New York, NY 1997.

[18] J. Y. Wu, D. M. Kingsland, J. F. Lee and R. Lee, "A Comparison of Anisotropic PML to Berenger's PML and Its Application to the Finite-Element Method for EM Scattering," *IEEE Transactions on Antennas and Propagation*, vol 45, pp. 40-50, January 1997.

[19] Z. Sacks, D. Kingsland, R. Lee, and J. F. Lee, "A Perfectly Matched Anisotropic Absorber for Use as an Absorbing Boundary Condition," *IEEE Transactions on Antennas and Propagation*, vol 43, pp. 1460-1464, December 1995.

[20] B. A. Hendrickson and D. E. Womble, "The torus-wrap mapping for dense matrix calculations on massively parallel computers," *Tech. Rep. SAND92-0792*, Sandia National Laboratories, 1992.

**Will Dearholt** received the B.S.E.E., M.S. and Ph.D. degrees from New Mexico State University in 1992, 1996 and 2003, respectively. He has been employed by NASA, IBM and Los Alamos National Laboratory as both a summer intern and a cooperative education student. He was an instructor at NMSU from 1997 to 2002. Dr. Dearholt received the NASA Graduate Research Fellowship in 1998 which funded the majority of the work described in this paper. Since graduating from New Mexico State University in 2003, he has been a Technical Staff Member at Los Alamos National Laboratory in Los Alamos, New Mexico working on mesh generation and geometric modeling for problems of interest to the computational physics community. Dr. Dearholt is a member of IEEE, ACES and Eta Kappa Nu.

**Steven Castillo** received the B.S.E.E. degree from New Mexico State University, Las Cruces, in 1982, and the M.S. and Ph.D. degrees in Electrical Engineering from the University of Illinois, Urbana, in 1984 and 1987, respectively, where he conducted research in electromagnetic analysis of high speed digital circuits. He has been employed at White Sands Missile Range, NASA Johnson Space Center, and Bell Telephone Laboratories. Since 1987, he has been on the faculty at New Mexico State University and is now an NMSU Regents Professor in the Klipsch School of Electrical and Computer Engineering where he served as Department Head from 1998 until 2004. He began his duties as Dean of Engineering July 1st, 2004. He has conducted research in computational methods in electromagnetics, electromagnetic theory and the solution of partial differential equations on parallel computers. Dr. Castillo is a member of the IEEE Antennas and Propagation, Electromagnetic Compatibility, and the Microwave Theory and Techniques societies. He currently serves on several boards and external committees including the Advisory Committee on Cyber Infrastructure for the National Science Foundation, the board of directors for the Center of Excellence for Hazardous Materials, the Computational Sciences review panel at Sandia National Laboratories, and the Advanced Simulation and Computing Program review at Sandia National Laboratories. He is a past Associate Editor of the IEEE Antennas and Propagation Transactions. He is a member of Eta Kappa Nu, Tau Beta Pi, Phi Kappa Phi and Sigma Xi and received the National Science Foundation Presidential Young Investigator Award in 1991.