INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 41, 759–776 (1998)

THE APPLICATION OF SCALABLE DISTRIBUTED MEMORY COMPUTERS TO THE FINITE ELEMENT MODELING OF ELECTROMAGNETIC SCATTERING

TOM CWIK*, DANIEL S. KATZ, CINZIA ZUFFADA AND VAHRAZ JAMNEJAD

Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA, 91109, U.S.A.

ABSTRACT

Large-scale parallel computation can be an enabling resource in many areas of engineering and science if the parallel simulation algorithm attains an appreciable fraction of the machine peak performance, and if undue cost in porting the code or in developing the code for the parallel machine is not incurred. The issue of code parallelization is especially significant when considering unstructured mesh simulations. The unstructured mesh models considered in this paper result from a finite element simulation of electromagnetic fields scattered from geometrically complex objects (either penetrable or impenetrable.) The unstructured mesh must be distributed among the processors, as must the resultant sparse system of linear equations. Since a distributed memory architecture does not allow direct access to the irregularly distributed unstructured mesh and sparse matrix data, partitioning algorithms not needed in the sequential software have traditionally been used to efficiently spread the data among the processors. This paper presents a new method for simulating electromagnetic fields scattered from complex objects; namely, an unstructured finite element code that does not use traditional mesh partitioning algorithms. © 1998 John Wiley & Sons, Ltd. This paper was produced under the auspices of the U.S. Government and it is therefore not subject to copyright in the U.S.

Int. J. Numer. Meth. Engng., **41**, 759–776 (1998)

KEY WORDS: finite elements; electromagnetics; parallel computing; scattering; sparse matrices

1. INTRODUCTION

Large-scale parallel computation can be an enabling resource in many areas of engineering and science. The available memory capacity and computational speed of large distributed memory machines can allow the simulation of complicated engineering components if the simulation algorithm attains an appreciable fraction of the machine peak performance, and if undue cost in porting the code or in developing the code for the parallel machine is not incurred. The issue of code parallelization is especially significant when considering unstructured mesh simulations. The unstructured mesh models considered in this paper result from a finite element simulation of electromagnetic fields scattered from geometrically complex objects (either penetrable or impenetrable.) The finite element model is used to capture the complex materials involved in the simulation, and to maintain fidelity of the structure's geometry. The unstructured mesh must be

*Correspondence to: T. Cwik, Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, M/S 168-522, Pasadena, CA 91109, U.S.A. E-mail: cwik@jpl.nasa.gov

CCC 0029–5981/98/040759–18$17.50
© 1998 John Wiley & Sons, Ltd.

Received 8 April 1997

distributed among the processors, as must the resultant sparse system of linear equations. Since a distributed memory architecture does not allow direct access to the irregularly distributed unstructured mesh and sparse matrix data, partitioning algorithms which are not needed in the sequential software have traditionally been used to efficiently spread the data among the processors. This paper presents a new method for simulating electromagnetic fields scattered from complex objects, namely, an unstructured finite element code that does not use traditional mesh partitioning algorithms. The complete software package is implemented on the Cray T3D massively parallel processor using both Cray Adaptive FORTRAN (CRAFT) compiler constructs to simplify portions of the code that operate on the irregular data, and optimized message passing constructs on portions of the code that operate on regular data and require optimum machine performance.

The finite element modeling software begins with mesh data constructed on a workstation using a commercially available CAD meshing package. Because the electromagnetic scattering simulation is an open region problem (scattered fields exist in all space to infinity), the mesh must be truncated at a surface that maintains accuracy in the modeled fields, and limits the volume of free space that is meshed. Local, absorbing boundary conditions can be used to truncate the mesh, but these may be problematic because they become more accurate as the truncating surface is removed from the scatterer, requiring greater computational expense, and they may be problem-dependent. The approach outlined in this paper solves the three-dimensional vector Helmholtz wave equation using a coupled finite element-integral equation method. A specific integral equation (boundary element) formulation that efficiently and accurately truncates the computational domain is used. A partitioned system of equations results from the combination of discretizing the volume in and around the scatterer using the finite element method, and discretizing the surface using the integral equation method. This system of equations is solved using a two-step solution, combining a sparse iterative solver and a dense factorization method. The matrix equation assembly, solution, and the calculation of observable quantities are all computed in parallel, utilizing varying number of processors for each stage of the calculation.

Various approaches have been taken for parallel implementations of unstructured mesh simulations. A short and general overview of all stages in the simulation of high-temperature superconductors—mesh generation and refinement, domain partitioning, and linear system solution—can be found in Reference 1. Similarly, approaches have been reported for simulations in structural mechanics using a coarse-grain machine[2] and in a review article for simulations in fluid dynamics using a data parallel computer.[3] An early implementation of a nodal-based finite element implementation simulating scattered electromagnetic fields on a data parallel computer was given in Reference 4. An implementation on a shared virtual memory machine of a finite element method using absorbing boundary conditions, simulating scattered electromagnetic fields was outlined in Reference 5. The application of finite volume methods using unstructured meshes for electromagnetic modelling of both guided wave structures and scatterers was presented in References 6 and 7. These approaches either employ specific mesh partitioning algorithms to decompose the mesh onto the distributed memory machine, or machine architecture and compiler attributes specific to the computer. In References 1, 2, and 7, bisection partitioning algorithms were used. In References 3 and 4, using a data parallel computer, compiler constructs replaced the partitioning algorithms, and in Reference 5, a global address space available on the Kendall Square Research machine was used to distribute the mesh. A specialized partitioning algorithm for thin planar structures was employed in Reference 6.

Rather than employing mesh partitioning methods, the emphasis in this work is placed on decomposition of the resultant sparse matrix entries among the distributed memory processors.

Though there is a relationship between the geometric mesh data and the assembled sparse matrix entries, it is the sparse matrix that is operated on directly in the iterative solver used in most large finite element simulations. Specifically, a distributed sparse matrix-dense vector multiply is the computational component that must be efficiently computed at each step of the iterative algorithm. It is therefore essential that the decomposition of matrix elements be completed in a manner that allows an efficient matrix-vector multiplication. The row slab matrix decomposition used in this work strikes a balance between near perfect data and computational load balance among the processors, minimal but not perfectly optimal communication of data in the matrix-vector multiply operation, and scalability of simulating larger-sized problems on greater numbers of processors.

2. THE COUPLED FINITE ELEMENT-INTEGRAL EQUATION MODEL

To practically compute a solution to exterior electromagnetic scattering problems, the domain must be truncated at some finite surface where the Sommerfeld radiation condition is enforced, either approximately or exactly. Approximate methods attempt to truncate the mesh using only local field information at each grid point, whereas exact methods are global, needing information from the entire mesh boundary.[8] The global method used here couples a three-dimensional finite element solution interior to the bounding surface with an efficient integral equation solution that exactly enforces the Sommerfeld radiation condition. The problem domain is divided into interior and exterior regions, separated at the mesh boundary (Figure 1). The unknown sources in the integral equation are directly related to the tangential fields on the mesh boundary, and the radiation condition is implicitly enforced exactly through the use of the free-space Green's function. Fields in the two regions are coupled by enforcing boundary conditions on tangential field components at the mesh boundary, thereby producing a unique and exact solution to Maxwell's equations in both regions.

The bounding surface chosen is the minimal surface of revolution that fits around the scatterer. The integral equation is discretized using sub-domain basis functions along the surface of revolution generator, and Fourier harmonics azimuthally, to greatly limit the storage necessary in the integral equation component of the model. An outline of this formulation is presented below. A detailed presentation can be found in Reference 9, with further results presented in Reference 10. An extension to radiation modeling is given in Reference 11.

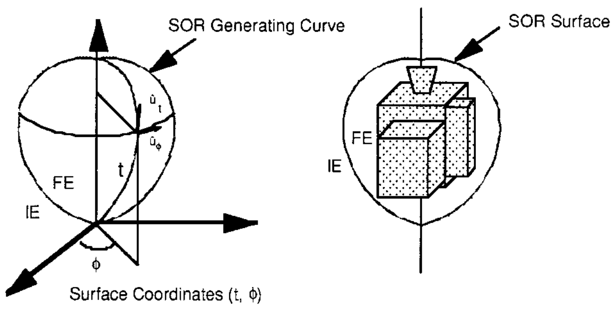

Figure 1. Geometry of scattering problem showing interior and exterior regions of model

2.1. Finite element representation

In the interior region, a finite element discretization of a weak form of the wave equation is used to model the geometry and fields, leading to

$$\frac{\eta_0}{jk_0}\iiint_V \left[\frac{1}{\varepsilon_r}(\nabla\times\bar{H})\cdot(\nabla\times\bar{W}^*) - k^2\mu_r\bar{H}\cdot\bar{W}^*\right]\mathrm{d}V - \iint_{\partial V}\bar{E}\times\hat{n}\cdot\bar{W}^*\,\mathrm{d}s = 0 \tag{1}$$

$\bar{H}$ is the magnetic field (the $\bar{H}$-equation is used in this paper; a dual $\bar{E}$-equation can also be written), $\bar{W}$ is a testing function, the asterisk denotes conjugation, and $\bar{E}\times\bar{n}$ is the tangential component of $\bar{E}$ on the surface of revolution S (∂V). Equation (1) represents the fields internal to and on the surface S. These fields will be modelled using a set of properly chosen finite element basis functions. In equation (1), ε_r and μ_r are the relative permittivity and permeability, respectively, k_0 and η_0 are free-space wave number and impedance, respectively.

A set of tetrahedral, vector edge elements (Whitney elements) are used to discretize (1),

$$\bar{W}_{mn}(r) = \lambda_m(r)\nabla\lambda_n(r) - \lambda_n(r)\nabla\lambda_m(r) \tag{2}$$

in which $\lambda(r)$ are the tetrahedral shape functions and indices (m, n) refer to the two nodal points of each edge. These elements are used for both expansion and testing (Galerkin's method) in the finite element domain.

2.2. Combined-field integral equation representation

In the formulation of the integral equation, fictitious electric ($\bar{J} = \hat{n}\times\bar{H}$) and magnetic ($\bar{M} = -\hat{n}\times\bar{E}$) surface currents, equivalent to the tangential magnetic and electric fields just on the exterior of the boundary surface, are defined on the boundary. These currents produce the scattered fields in the exterior region. A linear combination of the electric field integral equation (EFIE) and the magnetic field integral equation (MFIE) is used in this formulation, and it can be succinctly expressed as

$$Z_M[\bar{M}/\eta_0] + Z_J[\bar{J}] = V_i \tag{3}$$

where Z_M and Z_J are the integro-differential operators used in defining the combined field integral equation and V_i represents the incident field.

The integral equation on the surface of revolution is discretized by a set of basis functions with piecewise linear variation along the surface of revolution generator, and with an azimuthal Fourier modal variation. Applying Galerkin's method, both expansion and testing functions are given as

$$\begin{bmatrix}\bar{U}^t\\ \bar{U}^\phi\end{bmatrix} = \begin{bmatrix}\hat{t}\\ \hat{\phi}\end{bmatrix}\frac{T_k(t)}{\rho(t)}\mathrm{e}^{\mathrm{j}n\phi} \tag{4}$$

in which $T_k(t)$ is a triangle function spanning the kth annulus on the surface of revolution surface. The variables t and ϕ refer to the local surface of revolution co-ordinates, and ρ is the distance from the z-axis to a point on the surface of revolution. Each annulus spans two segments along the generator, each referred to as a strip. Adjacent triangles overlap on one segment.

2.3. *Enforcing boundary conditions*

At the artificial surface of revolution separating the interior and exterior regions, boundary conditions on the continuity of tangential field components must be enforced. Three equations are written for the three unknown field quantities of interest, the magnetic field $\bar{H}$ internal to the volume V and the electric and magnetic surface currents, $\bar{J}$ and $\bar{M}$, on the boundary. Continuity of the magnetic field across the boundary is enforced in a weak sense

$$\iint_{\partial V} (\hat{n} \times \bar{H} - \bar{J}) \cdot (\hat{n} \times \bar{U}^*) \, \mathrm{d}s = 0 \tag{5}$$

where $\bar{U}$ is a testing function. Continuity of the electric field across the boundary is made implicit in the finite element equation in the surface integral term $\hat{n} \times \bar{E}$ by replacing this term with $\bar{M}$.

The surface integral in (1) and the first component of the integral in (5) are termed the coupling integrals, since with a convenient choice of the unknown in the first and of the testing function in the second, they are made to couple interior and exterior field representations. To evaluate these terms, the finite element basis function $\bar{W}$ is approximately evaluated on the portion of surface of revolution projected from the triangular facet of the tetrahedron onto a strip. Such projections are curved triangles, curved quadrilaterals, or curved pentagons. The evaluation of the integrals are done numerically. These coupling integrals, as well as the discretization of the second surface integral in (5), complete the discretization of the problem.

2.4. *Numerical solution of the linear system*

Having introduced the basis and testing functions for the volume as well as the surface unknowns, substitution into the complete set of equations yields

$$\begin{vmatrix} \mathbf{K} & \mathbf{C} & \mathbf{0} \\ \mathbf{C}^{\dagger} & \mathbf{0} & \mathbf{Z}_0 \\ \mathbf{0} & \mathbf{Z}_M & \mathbf{Z}_J \end{vmatrix} \begin{vmatrix} \mathbf{H} \\ \mathbf{M} \\ \mathbf{J} \end{vmatrix} = \begin{vmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{V}_{\mathrm{i}} \end{vmatrix} \tag{6}$$

where

$$\begin{aligned} \mathbf{K} &= \langle K_p[\bar{W}_p] \cdot \bar{W}_q \rangle \\ \mathbf{C} &= -\eta_0 \langle \bar{U}_m \cdot \bar{W}_q \rangle \\ \mathbf{Z}_0 &= \eta_0 \langle \bar{U}_m \cdot [\hat{n} \times \bar{U}_n] \rangle \\ \mathbf{Z}_M &= \langle Z_{Mm}[\bar{U}_m] \cdot \bar{U}_n \rangle \\ \mathbf{Z}_J &= \langle Z_{Jm}[\bar{U}_m] \cdot \bar{U}_n \rangle \end{aligned} \tag{7}$$

The symbol † indicates the adjoint of a matrix. Note that both **K** and **C** are sparse, $\mathbf{Z}_0$ is tri-diagonal, and $\mathbf{Z}_M$ and $\mathbf{Z}_J$ are banded. In particular, the system is complex, non-symmetric, and non-Hermitian. The sparsity of the system (6) is shown in Figure 2 for a case with only several hundred finite element unknowns. For larger, representative cases, the number of finite element unknowns will grow into hundreds of thousands while the number of columns in **C** will be several hundred to several thousand.

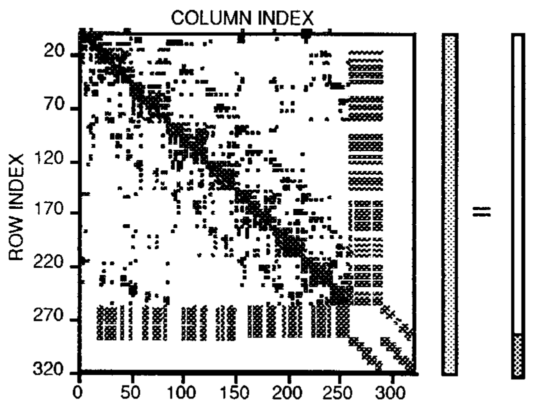

Figure 2. Scatter plot graphically showing structure of system of equations. Darkened spaces indicate non-zero matrix entries

The parallel solution to this matrix equation system is completed in two steps. Initially, **H** in the first equation in (6) is written as $\mathbf{H} = -\mathbf{K}^{-1}\mathbf{CM}$ and substituted into the second equation resulting in

$$\begin{vmatrix} \mathbf{Z}_K & \mathbf{Z}_0 \\ \mathbf{Z}_M & \mathbf{Z}_J \end{vmatrix} \begin{vmatrix} \mathbf{M} \\ \mathbf{J} \end{vmatrix} = \begin{vmatrix} \mathbf{0} \\ \mathbf{V}_{\mathrm{i}} \end{vmatrix} \tag{8}$$

where $\mathbf{Z}_K = -\mathbf{C}^{\dagger}\mathbf{K}^{-1}\mathbf{C}$. This relatively small system is then solved directly for **M** and **J**. By solving the system in two steps, the interior solution is decoupled from the incident field $\mathbf{V}_{\mathrm{i}}$, allowing for efficient solutions when many excitation fields are present as in monostatic radar cross-section simulations.

The relative numbers of unknowns in **H** and **M** (or **J**) makes the calculation of $\mathbf{K}^{-1}\mathbf{C}$ the major computational expense. This operation is the solution of a system of equations, $\mathbf{KX} = \mathbf{C}$, where **C** is a rectangular matrix with a potentially large number of columns in the case of electrically large scatterers. The solution is accomplished by using a symmetric variant of the quasi-minimum residual iterative algorithm. The resulting overall matrix (8) is treated as being dense, and the solution of this second system is accomplished via a direct dense LU decomposition, since its size is relatively small.

3. UNSTRUCTURED SPARSE MATRIX DECOMPOSITION

The solution of the large sparse system is the central component of the finite element simulation. Traditionally, the dependence between mesh data and the resultant sparse matrix data has been exploited in the development of mesh partitioning algorithms.[12–15] These algorithms break the physical mesh or its graph into contiguous pieces that are then read into each processor of a distributed memory machine. The mesh pieces are generated to have roughly the same number of finite elements, and to some measure, each piece has minimal surface area. Since the matrix assembly routine generates non-zero matrix entries that correspond to the direct interconnection of finite elements (elements that do not physically touch do not generate a matrix entry), the mesh partitioning algorithm attempts to create a load balance of the sparse system of equations.

Processor communication in the algorithm that solves the sparse system is meant to be limited by the ability to minimize the surface area of each mesh piece.

The algorithm for mesh partitioning typically requires less computational time than the rest of the finite element simulation, but due to the complexity of the algorithm needed to create good load balance and minimal processor communication, the development of parallel partitioning codes can be quite expensive. The complexity results from the irregularity of mesh data inherent in volumetric finite element modeling. The strategy followed in this paper is to exploit the availability of a global address space by using compiler constructs to efficiently decompose the matrix data among processors of the Cray T3D. Because the amount of time needed to perform the matrix decomposition is a small fraction of the overall simulation time, any minor inefficiencies in using the shared memory compiler constructs are relatively unimportant. The matrix equation solution—the major time expense of the overall simulation—and the calculation of observables are accomplished using message passing algorithms. This strategy allows the use of global addressing constructs to simplify the high complexity but computationally inexpensive portion of the simulation, i.e. the parallel finite element matrix assembly from mesh data, and the use of message passing algorithms on the portions of the simulation that require high performance. The direct decomposition of the matrix entries also results in regular data structures that are exploited by efficient communication patterns in the iterative solver.

In the electromagnetic scattering application considered in this paper, the system of equations under consideration is complex-valued, symmetric and non-definite. Because the system has these properties, and because very large systems are considered (systems up to order one million) the quasi-minimum residual iterative algorithm is used to solve the system.[16] Each row (or column) of the matrix has a number of non-zero entries, typically 16 for the elements currently being used, and this number is constant, independent of the mesh size. The main expense of the solution algorithm is the sparse matrix-dense vector multiply that is inherent in this as in most other Krylov subspace iterative algorithms. The matrix decomposition used in this implementation is based on row slabs of the sparse reordered system. The reordering algorithm is used to minimize the bandwidth of the sparse system. As Section 4 will outline, this decomposition and reordering is chosen to minimize communication of the overlapping vector pieces in the parallel matrix-vector multiplication, reduce storage of the resultant dense vector pieces on each processor, and allow for load balance in storage and computation.

Since the right-hand-side vectors in the parallel sparse matrix equation ($\mathbf{KX} = \mathbf{C}$) are the columns of $\mathbf{C}$, these columns are distributed as required by the row distribution of $\mathbf{K}$. When setting up the row slab decomposition, $\mathbf{K}$ is split by attempting to equalize the number of non-zeros in each processor's portion of $\mathbf{K}$ (composed of consecutive rows of $\mathbf{K}$). The rows in a given processor's portion of $\mathbf{K}$ determines the rows of $\mathbf{C}$ that processor will contain. As an example, if the total number of non-zeros in $\mathbf{K}$ is nz, a loop over the rows of $\mathbf{K}$ will be executed, counting the number of non-zeros of $\mathbf{K}$ in the rows examined. When this number becomes approximately nz/P (where P is the number of processors that will be used by the matrix equation solver), the set of rows of $\mathbf{K}$ for a given processor has been determined, as has the set of rows of $\mathbf{C}$.

The reordering is chosen to minimize and equalize the bandwidth of each row over the system.[17] Because the amount of data communicated in the matrix-vector multiplication will depend upon the equalization of the row bandwidth, different reordering algorithms have been examined. The generalized reverse Cuthill–McKee algorithm (in both the SPARSPAK[17] and the Gibbs–Poole–Stockmeyer[18] versions) produces an ordering that minimizes system bandwidth, and equalizes the bandwidth over each row of the matrix. Matrices resulting from objects that were long and thin, as well as those resulting from spherical objects have been examined. The nested dissection ordering in Reference 15 could produce a smaller profile of the reordered matrix, but equalization of the row bandwidth was not accomplished; row bandwidths even approaching the matrix order were found in a few rows of the matrix.

The matrix decomposition code, termed P_SLICE, consists of a number of subroutines. Initially, the potentially large mesh files are read (READ). Then the connectivity structure of the sparse matrix is generated and reordered (CONNECT), followed the generation of the complex-valued entries of **K** (FEM), building the connectivity structure and filling the **C** matrix (COUPLING). Finally, the individual files containing the row slabs of **K** and the row slabs of **C** must be written to disk (WRITE). For each processor that will be used in the matrix equation solver, one file containing the appropriate parts of both the **K** and **C** matrices is written.

3.1. Port to T3D using CRAFT

Cray Research Adaptive FORTRAN (CRAFT) is used for the matrix decomposition stage of the simulation. All large arrays are declared using *CDIR$* directives to be shared in either a block manner or a cyclic manner for the leading dimension, with non-leading dimension distributed degenerately. Using a block distribution of a matrix of size 256 on 4 processors leads to the first 64 elements residing on processor 0, the next 64 elements on processor 1, etc. A cyclic distribution would lead to processor 0 having elements (1, 5, 9, . . .), processor 1 having elements (2, 6, 10, . . .), etc. A two-dimensional array with a degenerate distribution of the second dimension leads to all elements of the array having a given index in the first dimension being on the same processor, regardless of the index in the second dimension. For example, a two-dimensional array of size (256, 10) distributed degenerately over the second dimension will have elements $((i, 1), (i, 2), \ldots, (i, 10))$ all located on the same processor. Which processor this will be is dependent on the value of i, and the method of distribution over the first dimension.

Routines which could be easily parallelized by CRAFT directives were FEM and part of COUPLING. The directive *CDIR$ DO SHARED* was added to the parallelizable loops to automatically distribute the work over all the processors. Other routines that could be executed in parallel with a combination of CRAFT and message passing included the READ and WRITE routines. The remaining routines (CONNECT, and a second part of COUPLING) are basically sequential routines, where only one processor is doing the majority of the work, while using data spread across many (usually all) processors.

Two files are read in the READ routine, one containing finite element data, and the other containing integral equation data. The finite element file is at least an order of magnitude larger than the integral equation file, and is read by 4 processors. By using these 4 processors, the time of the READ routine is reduced roughly by a factor of 3 as compared to reading the file with 1 processor. Further reduction in this time may be possible; however, this factor of 3 is currently sufficient. In the WRITE algorithm, data is assembled on each processing element and written to disk. On the T3D, it is faster to assemble a local array and write out that data than to write out a distributed array directly, since as the number of processors increases, more writes of smaller amounts of data are being performed, and disk and network contention develops. Scaling beyond this point quickly leads to diminishing returns from each processor.

Figures 3 and 4 show the performance of P_SLICE over varying numbers of processors for two different problems. The number of edges is the number of finite element unknowns in the problem. It may be observed that for the routines that have been parallelized, doubling the

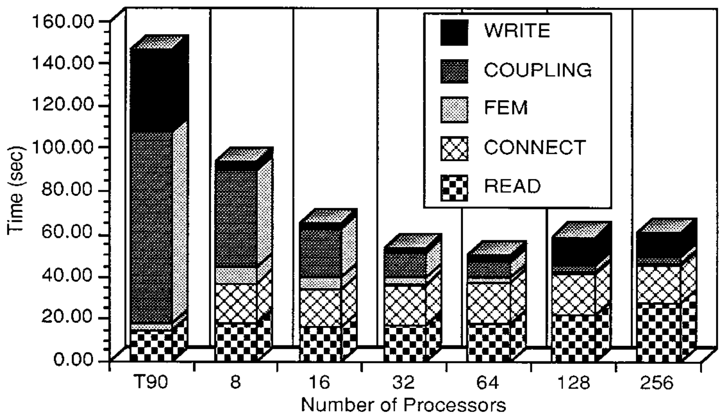

Figure 3. Computation time and scaling for a relatively small simulation (dielectric cylinder with 43 791 edges, radius = 1 cm, height = 10 cm, permittivity = 4·0 at 2·5 GHz). First column shows time for single processor T90. Times on T90 for CONNECT and FEM have been combined

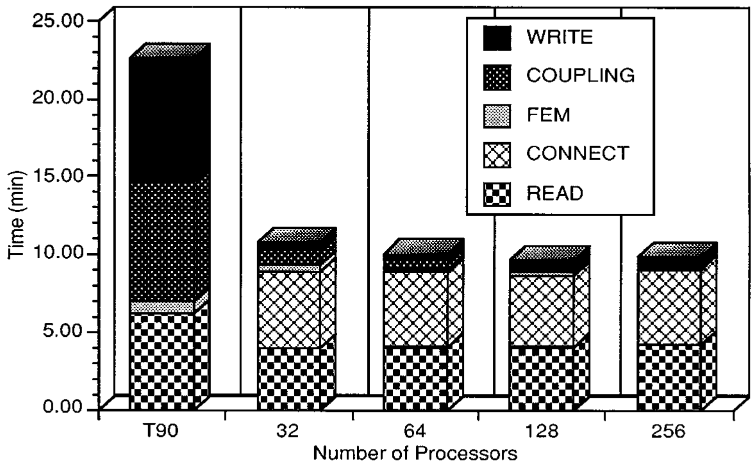

Figure 4. Computation time and scaling for a relatively large simulation (dielectric cylinder with 579 993 edges, radius = 1 cm, height = 10 cm, permittivity = 4·0 at 2·5 GHz). First column shows time for single processor T90. Times on T90 for CONNECT and FEM have been combined

number of processors reduces the amount of time by a factor of approximately two. For routines that are sequential, where only one processor is doing the work using the other processors' data, the time goes up very slightly as the number of processors for the overall code are increased. This is due strictly to communication latency. As the number of processors increases, the percentage of array elements which are not local increases, and the time to load or store these elements is longer than the time to load or store local elements. The I/O time should have roughly the same behaviour, but for practical tests the I/O time is more dependent on the I/O load of the other T3D processors and the load on the front-end YMP that is between the T3D and the disks than the number of T3D processors being used in P_SLICE. It is clear that the routines that benefit most from the parallel implementation on the T3D are COUPLING and WRITE.

4. PARALLEL SOLUTION OF PARTITIONED SYSTEM

As outlined above, the partitioned system of equations is solved in two steps, namely P_SOLVE and P_FIELD. Initially, the quasi-minimum residual algorithm[16] is used to solve the sparse system of equations $\mathbf{KX} = \mathbf{C}$, resulting in the reduced sub-matrix $\mathbf{Z}_K$. The parallel quasi-minimum residual solver developed for this application operates on matrix data decomposed by row slabs in P_SLICE after reordering (Figure 5 shows matrix structure before and after reordering). The machine is logically considered to be a linear array of processors, with each slab of data residing in one of the processors. $\mathbf{C}$ and $\mathbf{X}$ are also decomposed by row slabs, corresponding to the row partition of the matrix. Central components of the quasi-minimum residual algorithm that are affected by the use of a distributed memory machine are the parallel sparse matrix-dense vector multiply, and dot products and norm calculations that need vector data distributed over the machine. The dominant component is the matrix-vector multiplication, accounting for approximately 80 per cent of the time required to run P_SOLVE.

A parallel library of the needed level-one BLAS routines was developed using CRAY T3D *shmem_put* and *shmem_get* message passing. The routines required by the quasi-minimum residual algorithm are CDOTU and SCNRM2, and the parallel implementation of these was trivial, consisting of a local BLAS call to calculate each processor's contribution to the result, and a call to a global sum routine to calculate the final result.

4.1. *Parallel sparse matrix-dense vector multiplication*

The parallel sparse matrix-dense vector multiply involves multiplying the $\mathbf{K}$ matrix that is distributed across the processors in row slabs, each containing a roughly equal number of

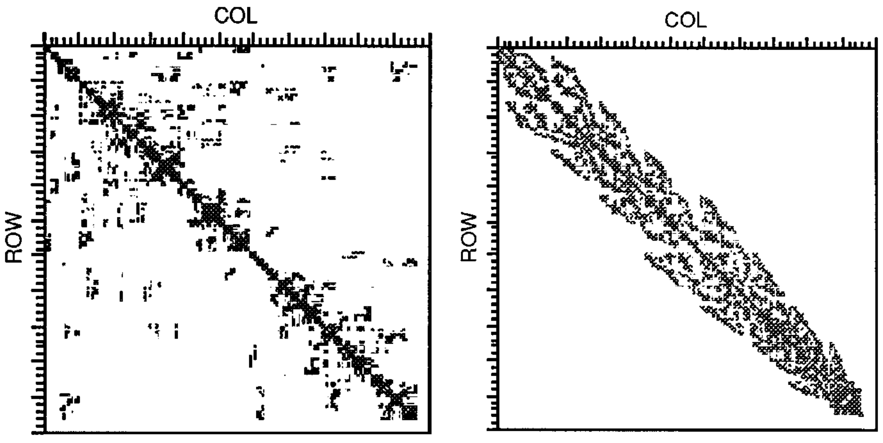

Figure 5. Original matrix structure (left) and after reordering (right). Filled spots indicate non-zero entries of matrix

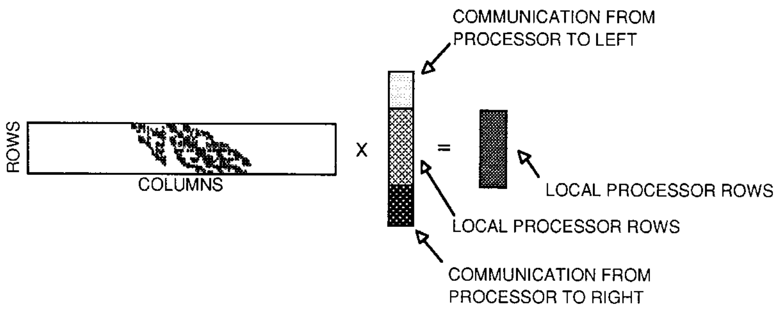
COMMUNICATION FROM PROCESSOR TO LEFT
ROWS
COLUMNS
X
=
LOCAL PROCESSOR ROWS
LOCAL PROCESSOR ROWS
COMMUNICATION FROM PROCESSOR TO RIGHT

Figure 6. Local sparse matrix-dense vector multiplication graphically displayed

non-zero elements, and a dense vector **x**, that is also distributed over the processors, to form a product vector **y**, distributed as is **x** (Figure 6). Since the **K** matrix has been reordered for minimum bandwidth, the minimum and maximum column indices of the slab are known. If the piece of the dense vector **x** local to this processor has indices within this extent of column indices, the multiplication may be done locally and the resultant vector **y** will be purely local. In general, the local row indices of the dense vector **x** do not contain the range of column indices; therefore, a communication step is required to obtain the portions of the multiplication vector **x** required by the column indices of the **K** matrix. This communication step only requires data from a few processors to the left and right. The exact number of processors communicating data is dependent on the row bandwidth of the local piece of **K**, and the number of processors being used. In the simulations considered, the number of processors communicating data is typically one or two in each direction on scaled problems.

This communication could be performed using either *shmem_get* or *shmem_put*. These are one-way communication calls where the processor from whose memory the data is being gathered or to whose memory the data is being stored, respectively, is not interrupted by the communication. The *shmem_get* formulation is more intuitive and simpler to program, but the communication bandwidth of the *shmem_put* routine on the T3D is substantially higher than the communication bandwidth of the *shmem_get* routine. For this reason, the *shmem_put* formulation is used. This formulation requires the cache to be flushed to maintain cache coherency, but the resulting performance of the matrix–vector multiplication is still 15 per cent higher than the performance obtained using the *shmem_get* formulation.

As described previously, the **K** matrix is stored in row slabs using row-compressed storage. As **K** is symmetric, this is equivalent to a column slab decomposition using column-compressed storage. **K** may be used in either way in the matrix-vector multiplication. In this step, a non-zero in column i requires $\mathbf{x}(i)$ to be obtained, and a non-zero in row j will produce a partial result for $\mathbf{y}(j)$. This implies that **K** stored in column slabs will require only communication of portions of **y** non-local to the processor after the local portion of the multiply, and similarly, **K** stored in row slabs will require communication only to gather **x** before the local portion of the multiply. Since similar amounts of communication are required using either storage scheme, the scheme that minimizes the time spent in local work has been chosen for implementation. This is the row slab decomposition of **K**, because the row-compressed storage scheme better reuses the T3D processor's local cache, and therefore has better overall performance.

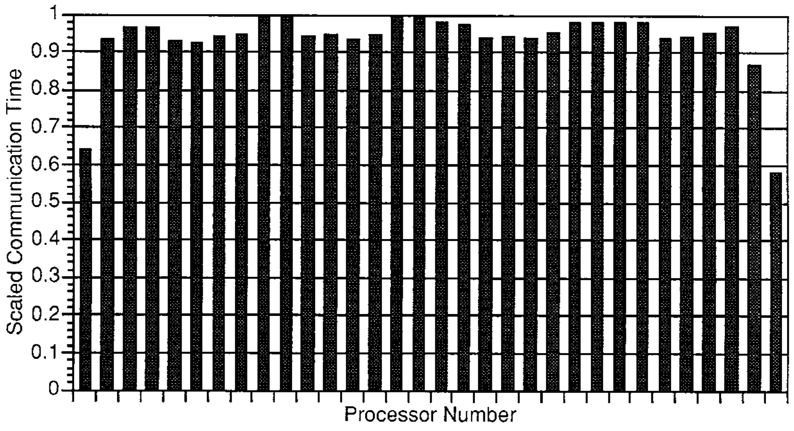

Figure 7. Graph of communication load balance for parallel matrix vector multiply, 271 158 edge dielectric cylinder, 32 processors

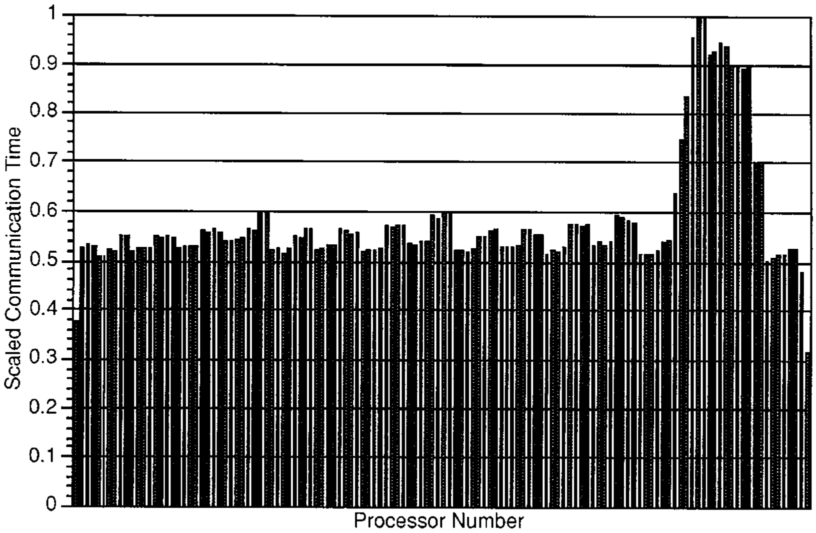

Figure 8. Graph of communication load balance for parallel matrix vector multiply, 579 993 edge cylinder, 128 processors

4.2. Performance and scalability of parallel sparse matrix-dense vector multiplication

The goal of the combination reordering-partitioning strategy discussed above is to minimize as well as equalize communication in P_SOLVE, while retaining memory load balance. The partitioning chosen clearly succeeds in evenly dividing the data among the processors; Figures 7 and 8 show the relative communication time of the processors.

Figure 7 shows results representative of the majority of the cases that have been run. All processors, excepting those on the ends of the linear processor array, have a relatively similar amount of communication, and since the communication is synchronized, all processors will require as much time as the one that uses the most time. Only the two end processors will be idle very long at the barrier. For this case, all processors except the first and last have to communicate with two other processors, one to the left and one to the right.

Figure 8 shows the other possible class of results, shared by a minority of cases that have been run. Again, the two end processors are using less time for communication than the majority of processors. However, in this example, a small subset of the processors are using more time in communication than the average processor. All the processors except those in this subset have to wait a substantial amount of time at the barrier, and the speed per processor of this run is lower than that of the first example. Again in this example, all processors but the first and last have to communicate with at least two other processors, one to the left and one to the right, but here, the processors in the subset that are spending more communication time are communicating with possibly two processors in either direction. The issue in these few cases is that the decomposition of the **K** matrix was performed entirely based on storage load balance, with the assumption that the reordering would equalize the row bandwidth and create communication load balance. This assumption is generally valid, as shown in Figure 7, though not always, as shown in Figure 8.

Another factor in the performance of the parallel matrix-vector multiplication is the percentage of communication. This is mainly related to the number of processors to the left and right that each processor must communicate, and as discussed above, the maximum number that any processor must communicate with. It is clear that running a fixed size problem on an increasing number of processors will generate a growing amount of communication. The amount of communication is a function of how finely the **K** matrix is decomposed, since its maximum row bandwidth after reordering is not a function of the number of processors used in the decomposition. If the maximum row bandwidth is m and each processor in a given decomposition has approximately m rows of **K**, then most processors will require one processor in each direction for communication. If the number of processors used for the distribution of **K** is doubled, each processor will have approximately $m/2$ rows of **K**. Since the row bandwidth does not change, each processor will now require two processors in each direction for communication. But since the number of floating point operations required has not changed, the communication percentage should roughly double. This can be seen in Figure 9, which shows communication percentage versus number of processors, for four problem sizes.

Figure 10 shows the local rate of operations/second for the parallel matrix vector multiplication. It is measured after communication has been completed. It can be seen that the performance of this operation is roughly constant, and is not easily identifiable as a function of problem size or number of processors. To a limited extent, a problem which involved more data on each processor will run slightly faster than would a problem with less data on each processor, but as Figure 10 demonstrates, this is not necessarily true. The storage of the data and how it fits in the T3D's cache is more important than the amount of data, and this forces the local performance rate not to be a simple function of problem size per processor.

Shown in Figure 11 are plots of time to convergence on different numbers of processors for five different problems. The number of unknowns in the finite element mesh and the number of columns of **C** are indicated on the plots. The quasi-minimum residual algorithm was stopped when the normalized residual was reduced three orders of magnitude for each column of **C**. With an initial guess being the zero vector, this results in a normalized residual of 0·1 per cent, a value that is sufficient for this scattering problem. Given a fixed communication percentage and a fixed rate for local work, doubling the number of processors for a given problem would halve the total

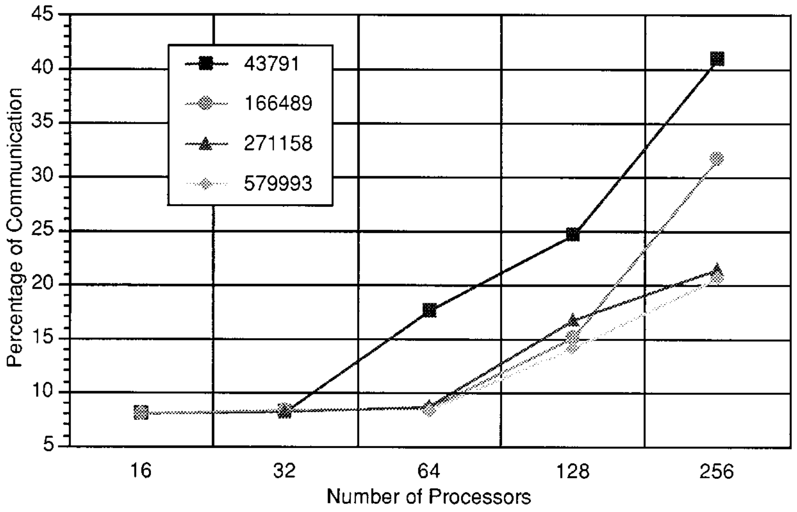

Figure 9. Percentage of communication versus number of processors for parallel matrix–vector multiplication, for four different size (number of edges) meshes of dielectric cylinder

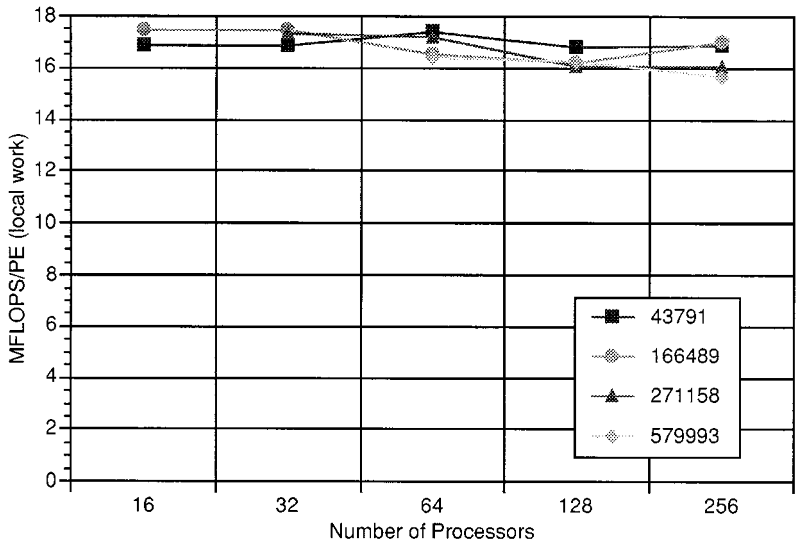

Figure 10. Local operation rate versus number of processors for parallel matrix–vector multiplication, for four different size (number of edges) meshes of the dielectric cylinder

solution time. The curves in Figure 11 do not drop linearly at this rate because these assumptions are not met, as shown by Figures 9 and 10. The decreased amount of work per processor causes the curves to level off as the number of processors increases.

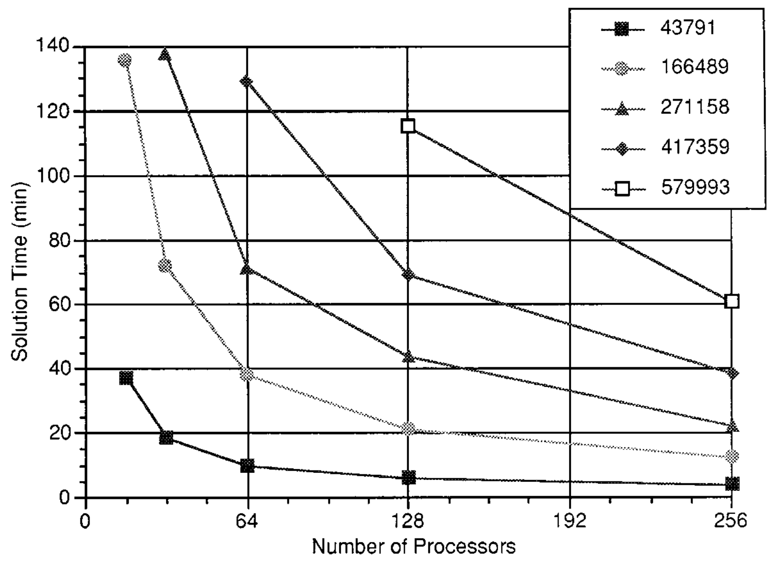

Figure 11. Time of convergence for five different problems. The time shown is the total execution time for the solver on different numbers of processors. The **C** matrix had 116 columns in each case

4.3. Additional work in P_SOLVE

After each column of $\mathbf{K}^{-1}\mathbf{C}$ is computed using the quasi-minimum residual algorithm, it must be multiplied by $\mathbf{C}^{\dagger}$ to obtain the equivalent column of $\mathbf{Z}_k$. Each of these multiplication requires a global communication, since **C** is distributed over the T3D by row slabs. To reduce the number of global communications, after a number of columns of $\mathbf{K}^{-1}\mathbf{C}$ are computed, these are multiplied by $\mathbf{C}^{\dagger}$, and the columns of $\mathbf{Z}_k$ obtained are written out sequentially to disk. The original quasi-minimum residual algorithm solved a single solution vector at a time. A pseudo-block (multiple right-hand-side) quasi-minimum residual variant was written, which performs each quasi-minimum residual iteration on some number of columns of **C** simultaneously. As the residual of each column of $\mathbf{K}^{-1}\mathbf{C}$ converges below the threshold, that column is no longer used in the quasi-minimum residual algorithm. This variant performs the same number of floating point operations as the single right-hand-side quasi-minimum residual algorithm, but the **K** matrix is required to be loaded from memory much less often. This leads to a time savings of 10–15 per cent in P_SOLVE.

5. CALCULATION OF OBSERVABLES

The final code of the simulation, P_FIELD, completes the matrix calculation shown in equation (8) and computes observable quantities (radar cross-section, near fields, etc.) After the $\mathbf{Z}_M$, $\mathbf{Z}_J$ and $\mathbf{Z}_0$ sub-matrices and $\mathbf{V}_{\mathrm{i}}$ vector (s) are computed, and the sub-matrix $\mathbf{Z}_K$ (formed by P_SOLVE) is read in from disk, a parallel dense matrix LU decomposition algorithm is used to solve the reduced system.[19] Since this system is much smaller than the larger sparse system solved above, the **Z** matrices may be distributed on a smaller set of processors, chosen to optimize the solve

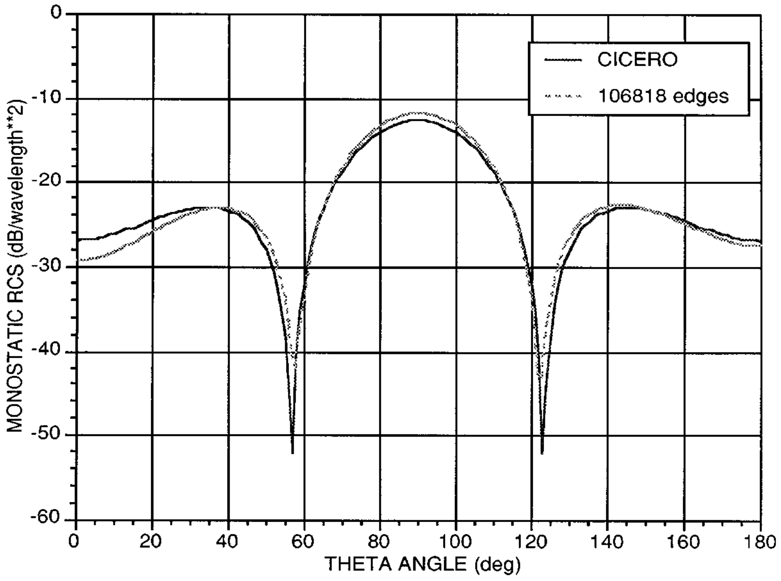

Figure 12. Monostatic radar cross section for dielectric cylinder with radius = 1·0 cm, height = 10·0 cm, relative permittivity = 4·0 at 2·5 GHz.

time. The time needed to solve this system compared to the sparse system is a small fraction, typically less than 1 per cent.

The radar cross-section is found from the mesh-surface-equivalent currents $\bar{M}$ and $\bar{J}$. This calculation—an integral over the surface—is easily parallelized on the processors executing P_FIELD. If the radar cross-section for more than one excitation vector is needed (monostatic), a block of solution vectors are found, and a block of radar cross-sections calculated. For completeness Figure 12 shows the radar cross-section for the dielectric cylinder used in the previous results; comparison is made to the CICERO code.[20] Further results of calculated observables may be found in Reference 10.

6. DISCUSSION

Shown in Figure 13 is the comparison of time requirements of the three stages of the simulation, for four different problem sizes. The problem simulated corresponds to the dielectric cylinder outlined in previous results. As is clearly shown, the dominant component of the simulation is P_SOLVE—the iterative solution of the sparse system. The matrix decomposition stage (P_SLICE) is relatively small, while the observable calculation stage (P_FIELD) is a minor fraction of the total time. This last stage can grow if a large number of field calculations are required, but it will typically remain a small fraction of the matrix solution time.

Using matrix decomposition by row-slab partitioning following reordering produced data structures that generally allowed a balanced matrix–vector multiplication in the iterative solver. The data load balance was almost exactly uniform, while the communication overhead was moderately small and similarly uniformly balanced over the machine for the majority of problems

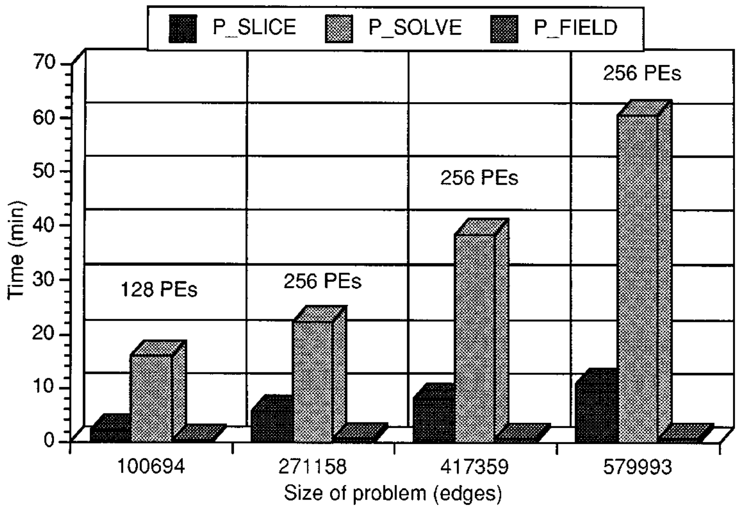

Figure 13. Comparison of time requirements for three stages of simulation for four different problem sizes. The problems correspond to the dielectric cylinder shown in Figure 12

considered. For scaled-sized problems, the communication time was roughly 15 per cent of the total matrix–vector multiplication time. Even bringing this expense down to zero time would not lead to a major improvement in the overall performance of the code. However, major improvements are possible in two areas: the local multiplication and the number of quasi-minimum residual iterations.

First, the performance on the local portion of the sparse matrix-dense vector multiplication could be improved. This is dependent on the sparse data-storage structure of the matrix and how it is loaded into the local cache. The relative sparsity of the reordered row slab of the matrix causes the multiplication to jump around in the cache as it loads the elements of the **X** vector. If the these local row slabs were reordered in such a way as to obtain a more dense matrix, the local performance would increase dramatically.

Second, an efficient parallel preconditioner, or block iterative solver could decrease the number of iterations needed in the matrix equation solution. Naturally, the preconditioner must not increase either the overhead in setting up the problem or obtaining the final solution more than it saves by lowering the iteration count. The block solver also must not increase the time per iteration more than the amount it saves by lowering the iteration count. These last two approaches are currently being examined.

ACKNOWLEDGEMENT

The authors wish to gratefully acknowledge the support of Jean Patterson, manager of the task *Research in Parallel Computational Electromagnetics*, and Mike Heroux of Cray Research, who assisted in developing an understanding of various sparse matrix-dense vector multiplication formulations. The JPL/Caltech Supercomputer used in this investigation was provided by the NASA Offices of Mission to Planet Earth, Aeronautics, and Space Science. The Cray T90 used in this investigation was provided by the Information Services Department of Cray Research. The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

1. L. Freitag, M. Jones and P. Plassmann, 'Parallel algorithms for unstructured mesh computation', *Comput. Systems Engng*., **5,** 297–309 (1994).
2. J. Oden and A. Patra, 'A parallel adaptive strategy for *hp* finite element computations', *Comput. Methods Appl. Mech. Engng*., **121,** 449–470 (1995).
3. T. Tezduyar, S. Aliabadi, M. Mehr and S. Mittal, 'Massively parallel finite element simulation of compressible and incompressible flows', *Comput. Methods Appl. Mech. Engng*., **119,** 157–177 (1994).
4. S. Hutchinson, E. Hensel, S. Castillo and K. Dalton, 'The finite element solution of elliptical systems on a data parallel computer', *Int. J. Numer. Meth. Engng*., **32,** 347–362 (1991).
5. A. Chatterjee, J. Volakis and D. Windheider, 'Parallel computation of 3-D electromagnetic scattering using finite elements', *Int. J. Numer. Model.*: *Elect. Networks*, *Devices Fields*, **7,** 329–342 (1994).
6. S. Gedney and U. Navsariwala, 'A comparison of the performance of the finite difference time-domain, finite element time domain, and planar generalized Yee algorithms on high-performance parallel computers', *Int. J. Numer. Model.: Elect. Networks Devices Fields*, **8,** 265–275 (1995).
7. N. Madsen, 'Divergence preserving discrete surface integral methods for Maxwell's curl equations using non-orthogonal unstructured grids', *J. Comput. Phys*., **119,** 34–45 (1995).
8. J.-J. Jin, *The Finite Element Method in Electromagnetics*, Wiley, New York, 1993.
9. T. Cwik, C. Zuffada and V. Jamnejad, 'Efficient coupling of finite element and integral equation representations for three-dimensional modeling', in T. Itoh, G. Pelosi and P. Silvester (eds.), *Finite Element Software for Microwave Engineering*, Wiley, New York, to be published, 1996.
10. T. Cwik, C. Zuffada and V. Jamnejad, 'Modeling three-dimensional-scatterers using a coupled finite element-integral equation representation', *IEEE Trans. Antennas Propag*., **44,** 453–459 (1996).
11. C. Zuffada, T. Cwik and V. Jamnejad, 'Modeling radiation with an efficient hybrid finite element-integral equation-waveguide mode modeling technique', *IEEE Trans. Antennas Propag*., **45,** 34–39 (1997).
12. B. Nour-Omid, A. Raefsky and G. Lyzenga, 'Solving finite element equations on concurrent computers', A. Noor (ed.), *American Soc. Mech. Eng*., 1986, pp. 291–307.
13. A. Pothen, H. Simon and K. Liou, 'Partitioning sparse matrices with eigenvectors of graphs', *SIAM J. Matrix Anal. Appl*., **11,** 430–452 (1990).
14. B. Hendrickson and R. Leland, 'An improved spectral graph partitioning algorithm for mapping parallel computations', *SIAM J. Sci. Comput*., **16,** 452–469 (1995).
15. G. Karypis and V. Kumar, 'A fast and high quality multilevel scheme for partitioning irregular graphs', *Technical Report TR 95-035*, Department of Computer Science, University of Minnesota, 1995.
16. R. Freund, 'Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices', *SIAM J. Statist. Comput*., **13,** 425–448 (1992).
17. A. George and J. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
18. J. Lewis, 'Implementation of the Gibbs–Poole–Stockmeyer and Gibbs–King algorithms', *ACM Trans. Math. Software*, **8,** 180–189 (1982).
19. T. Cwik, R. van de Geijn and J. Patterson, 'Application of massively parallel computation to integral equation models of electromagnetic scattering (Invited Paper)', *J. Opt. Soc. Am. A*, **11,** 1538–1545 (1994).
20. L. N. Medgeysi-Mitschang and J. M. Putnam, 'Electromagnetic scattering from axially inhomogeneous bodies of revolution', *IEEE Trans. Antennas Propagation*, **AP-32,** 797–806 (1984).