Highly Parallel Implementation of the 3D Integral Equation Asymptotic Phase Method for Electromagnetic Scattering[1]

Xianneng Shen, *RS/6000 System Division, IBM Corporation*
Aaron W. Davis, *Computer Science Department, University of Washington*
Keith R. Aberegg, *ERIM International, Ann Arbor, MI*
Andrew F. Peterson, *School of Electrical & Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250, peterson@ee.gatech.edu*

ABSTRACT: In this paper, we discuss the implementation of the 3D Integral Equation–Asymptotic Phase (IE–AP) method using the parallel architecture IBM RS/6000 SP. The IE-AP method is a hybrid numerical/asymptotic approach for electromagnetic scattering that attempts to reduce the number of unknowns required to accurately model electrically large structures. The IE-AP method will be described, and results will be reported for the parallel matrix fill implementation, and the relative performance of the PESSL and PETSc toolkits for parallel matrix solution.

1. Introduction

The application of interest is the radar cross-section (RCS) computation for a 3D arbitrarily shaped electrically large target. Practical RCS prediction using numerical methods has long been thought of as unrealistic because of the excessive computation and memory requirements. The RCS prediction of a fighter-sized aircraft, for example, requires the solution of a matrix equation whose dimension can easily exceed 10^6. The cost of such computations also discouraged efforts to improve other aspects of computational electromagnetics. The successful development of massively parallel processing (MPP) technologies improves the opportunity to solve these problems [1-4].

In this paper, we discuss the development and implementation of the 3D Integral Equation – Asymptotic Phase (IE-AP) code using a Massively Parallel Processing (MPP) architecture. The IE-AP method is a hybrid numerical/asymptotic approach for electromagnetic scattering that incorporates an asymptotic phase function in an attempt to reduce the number of required unknowns by as much as an order of magnitude. The formulation was originally implemented as a traditional serial process [5-6]. In order to fully test the IE-AP approach, the method was implemented on a parallel-architecture IBM RS/6000 SP. An overview of the IE-AP formulation is presented in the following section, while the remainder of this article is focused on the parallel implementation using the IBM SP and some of the observations we made during this process.

2. The IE-AP Formulation

The Integral Equation – Asymptotic Phase (IE–AP) method is a hybrid numerical/asymptotic procedure for determining the current density and far fields associated with electromagnetic scatterers. The method attempts to eliminate the traditional dependence on a 10-per-wavelength unknown density, without limiting the shape of the target under consideration or the accuracy. The method has been developed and tested on 2D [5] and 3D [6] perfectly conducting scatterers, and the initial results suggest an order of magnitude reduction in the necessary unknown density. In the present implementation of the method, the current density is taken to be the product of an asymptotic result (the phase of the physical optics current) and a residual vector function that is solved for using the method of moments. The motivation for the approach is the idea that the traditional 10-per-wavelength unknown density is primarily dictated by the phase progression of the electromagnetic wave or current (360 degrees per wavelength), and therefore over large portions

[1] This work was supported by the Cornell Theory Center, by a Supercomputing Program for Undergraduate Research Grant from the National Science Foundation, and by NSF ECS-9257927.

of targets an asymptotic result can provide the necessary phase progression. By treating the unknown currents as a product of the asymptotic result and an unknown residual function, it is thought that the residual function will be slowly varying and therefore can be represented by far fewer unknowns per wavelength.

The two-dimensional implementation of the IE–AP method reported in [5] involved the combined-field equation (CFIE), curved (parabolic) cells, subsectional quadratic polynomial basis functions, Dirac delta (TM) or pulse (TE) testing functions, and the incorporation of the appropriate TM edge singularity at corners. Test geometries considered in [5] consisted of circular cylinders, elliptic cylinders, wedge-cylinder structures, and square cylinders, all of which had perimeter dimensions in the range of 44 to 71 wavelengths. In most cases, the IE–AP result obtained with an average unknown density of one per wavelength or less differed by no more than 3 percent from a reference numerical solution obtained with the traditional 10-unknowns-per-wavelength density. In the 2D case, the method appears to produce an order-of-magnitude reduction in unknown density without significant additional error in the results. Based on the apparent success of the 2D approach, the idea was extended to three dimensions [6].

To eliminate interior resonance difficulties that arise with the analysis of electrically large scatterers with the electric-field or magnetic-field equations [9-10], the 3D CFIE was employed. The CFIE for scattering from perfectly conducting targets can be written in the form

$$\alpha \bar{E}^{inc}_{tan} + (1-\alpha)\eta\, \hat{n} \times \bar{H}^{inc} = -\alpha \left(\frac{\nabla\nabla\cdot\bar{A} + k^2\bar{A}}{j\omega\varepsilon} \right)_{tan} + (1-\alpha)\eta\, (\bar{J}_s - \hat{n} \times \nabla \times \bar{A}) \quad (1)$$

where α is an arbitrary parameter in the range $0 < \alpha < 1$, η is the medium impedance, ε is the permittivity, ω is the radian frequency,

$$\bar{A}(u,v) = \iint \bar{J}_s(u',v') \frac{e^{-jkR}}{4\pi R}\, du'dv' \quad (2)$$

and R is the distance from the source point (u′,v′) to the observation point (u,v), which in (1) is located an infinitesimal distance outside the scatterer.

Assuming that the incident field is a uniform plane wave, the current in Equation (1) is replaced by the expansion

$$\bar{J}_s(u,v) \cong e^{jk(x \sin\theta \cos\phi + y \sin\theta \sin\phi + z \cos\theta)} \sum_n j_n \bar{B}_n(u,v) \quad (3)$$

where (x,y,z) denotes a point on the surface of the scatterer with a local parametric representation (u,v), and (θ,ϕ) denotes the spherical angle from which the incident wave propagates. The exponential function represents the asymptotic phase associated with the uniform plane wave excitation; equivalently this is the phase arising from the physical optics approximation of the surface current. In contrast to the classical physical optics approximation, however, we use (3) throughout the lit and shadow portions of the target.

Our implementation uses triangular patches with parabolic curvature to represent the scatterer, and a type of razor-blade testing function to enforce the equation [8]. Based on our experience with the 2D results, it was decided to implement higher-order basis functions for the 3D case instead of the traditional Rao-Wilton-Glisson (RWG) functions, which provide at most a linear representation. The vector basis functions we implemented have one higher polynomial order in each variable than the RWG functions, and provide a linear representation of the normal vector component and a quadratic representation of the tangential vector component (LN/QT) along cell edges prior to mapping [7, 10]. The mapping process used to incorporate patch curvature is described in [8]. Reference [7] provided data showing the improved accuracy of the LN/QT functions on curved patches compared with the RWG functions on flat facets, and demonstrated a reduction of 2-3 in the required number of unknowns. [Note that this improvement does not take into account the incorporation of the asymptotic phase function in Equation (3).]

A few preliminary results were generated for the 3D IE–AP method, incorporating the asymptotic phase function, using spheres and cone-spheres as test cases [6]. Spheres permit a comparison with the exact solution, while cone-spheres permit comparison with alternative formulations that incorporate axial symmetry (so-called "body of revolution" codes). For example, Figures 1a, 1b, and 1c show the surface currents and scattering cross section for a sphere of radius 3λ. These results were obtained using 720 unknowns, for an average unknown density of about 6.5 unknowns per square wavelength of surface area. Note that there are discontinuities in the current, more evident in the ϕ component, which are typical of any expansion in divergence-conforming vector basis functions (such as the RWG functions and the LN/QT functions used here). For the particular mesh used to produce these data, the scattering cross section exhibits some error in the 60-90 degree sector.

As a second example, consider a cone-sphere with a total length of 10 wavelengths, an endcap radius of 1.48 λ, and a cone half angle of 7.5 degrees. Figure 2 shows the scattering cross section obtained with the IE-AP formulation using 1600 unknowns, for a total average density of 29 unknowns per square wavelength. A reference solution obtained using a "body of revolution" code with 40 unknowns per wavelength along the generating arc is shown for comparison. Although the agreement is acceptable over most of the range, the data show some apparent error in the 0-80 degree portion of the plot.

The preceding results suggest that a substantial reduction in unknowns might be possible with the IE-AP approach, as compared with a traditional subsectional integral equation formulation that almost always requires more than 100 unknowns per square λ. However, the specific results in Figures 1 and 2 contain error, and better scatterer

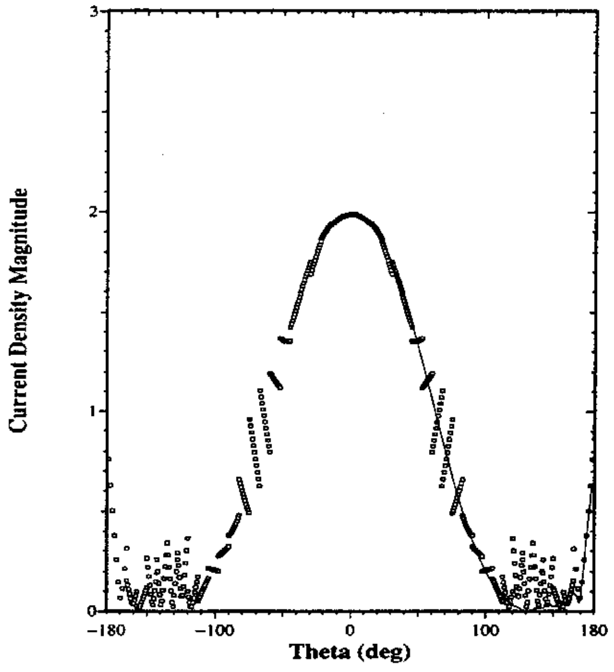

Figure 1b. The ϕ-component of the surface current density induced on a sphere of radius 3 λ. The CFIE result (markers) is compared with the exact solution (solid curve).

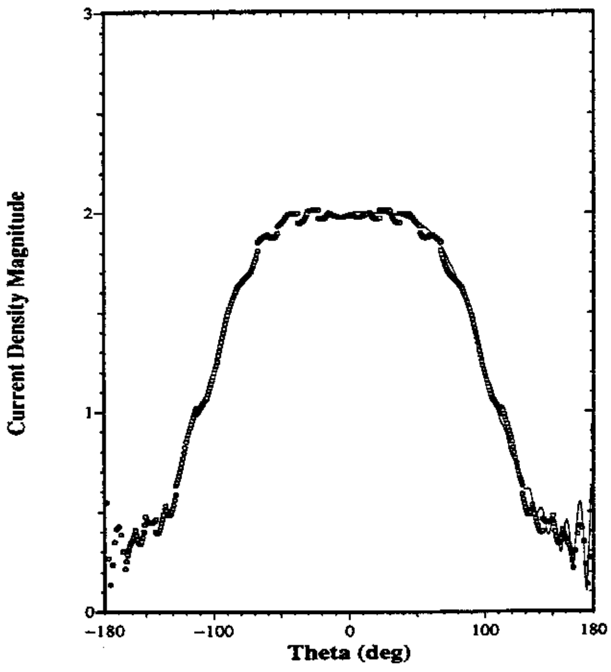

Figure 1a. The θ-component of the surface current density induced on a sphere of radius 3 λ. The CFIE result (markers) obtained using the IE-AP expansion with only 720 unknowns is compared with the exact solution (solid curve).

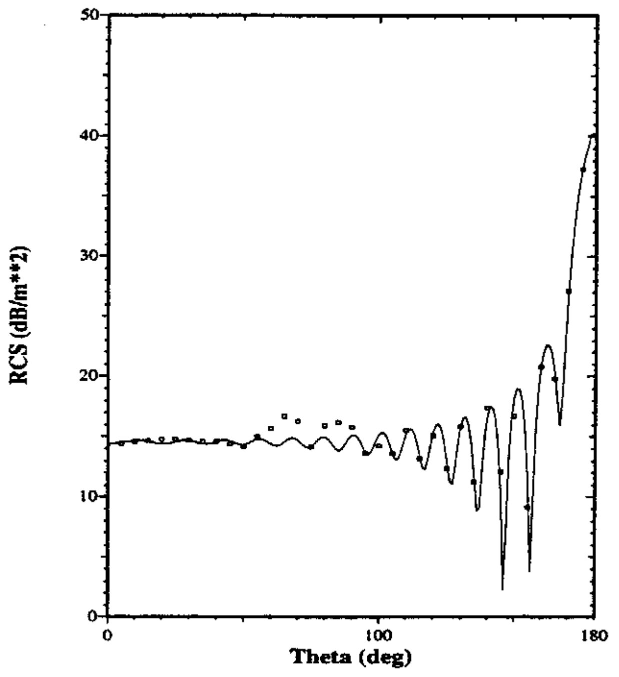

Figure 1c. The scattering cross section for a sphere of radius 3 λ. The CFIE result obtained using the IE-AP expansion with only 720 unknowns (markers) is compared with the exact solution (solid curve).

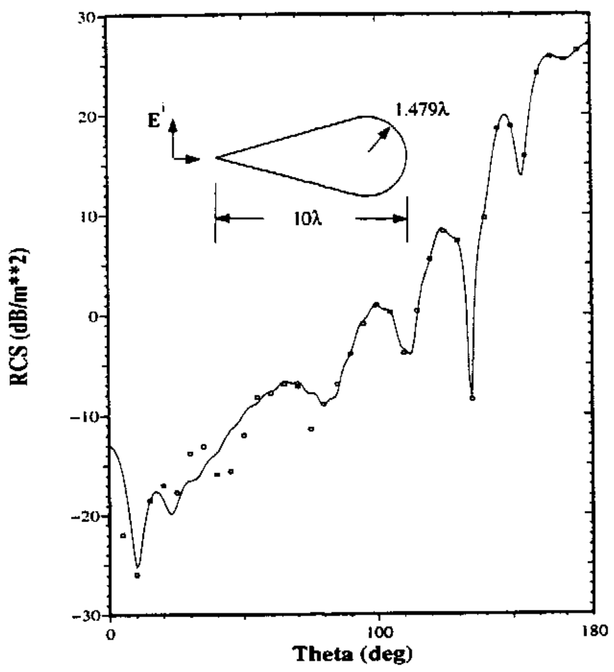

Figure 2. The scattering cross section of a cone-sphere of length 10 λ, an endcap radius of 1.48 λ, and a cone half angle of 7.5 degrees. Results obtained using the IE-AP formulation with 1600 unknowns (markers), a density of 29 unknowns per λ^2, are compared with a reference solution obtained using a "body of revolution" code with 40 unknowns per wavelength along the generating arc (solid curve).

meshes are necessary to improve the accuracy of the results. These test cases were meshed empirically, with many of the cells electrically large (less than 10 unknowns per wavelength over most of the cone-sphere, but with a much higher density of cells near the tip). Since the cell sizes arising within the IE-AP implementation can range from the usual size (0.1 λ) to cells that can be quite large (perhaps 10 λ in some situations) it is difficult to arrive at a nearly optimal scatterer model without feedback from the numerical solution. We feel that some form of adaptive gridding is necessary to optimize the cell dimensions based on local error estimates obtained from a coarse-cell model. In addition, one would expect that the savings from the IE-AP procedure would become greater as the electrical size of the scatterer increased. Unfortunately, available computer resources limited the range of scatterers easily analyzed in [6] to those of about 10 wavelengths in linear dimension. In order to treat larger problems and study the IE–AP method in detail, the present authors teamed with the support of the Cornell Theory Center to adapt the method to the parallel-architecture IBM SP machine.

Because of the reduced number of unknowns, and the additional cost associated with the entries of the matrix equation, the IE–AP method represents a shift of the traditional computational burden away from the matrix solve part of the process and toward the matrix fill part. Consequently, the procedure might be easier to parallelize with a high efficiency than a traditional integral equation formulation. In the following sections, we consider the parallelization of the 3D IE-AP algorithm for the IBM SP architecture.

3. Parallel Implementation

There are several distinct parallel-architecture computer systems available [1-2, 11]. They can be cataloged as shared memory, distributed shared memory, and distributed memory systems. We implemented the parallel IE–AP code on the Cornell Theory Center's 512-node IBM RS6000 SP system using the Message-Passing Interface (MPI). The IBM RS6000 SP employs a distributed memory architecture with relatively powerful nodes and a fast interconnect network, the SP switch network. The SP supports three classes of parallel programming models: message-passing, data parallel, and shared memory. An explicit message-passing program can run very efficiently since it matches the underlying SP system architecture. The message-passing library can be accessed from a FORTRAN program.

The explicit message-passing approach using MPI has a number of advantages. MPI is a standard message-passing library, and most parallel computer vendors have implemented MPI to facilitate code portability despite underlying hardware differences.

Because the IE–AP matrix entries involve a traditional Green's function multiplied with an asymptotic phase function, and integral domains that may span several wavelengths, the matrix entry computation is somewhat more costly than with a traditional 10-unknown-per-wavelength integral equation code. For N unknowns, the computational complexity of the matrix fill procedure is N^2 while the matrix factorization involves complexity of order N^3. The coefficient of the N^2 term in the complexity calculation for the IE–AP approach is somewhat larger than in traditional formulations, while the order N is smaller (by perhaps an order of magnitude). Since the matrix fill task represents the majority of the computation for a moderate-sized problem, the parallelization study initially focused on that part of the procedure. Subsequently, a parallel LU factorization was implemented using two available libraries.

The simple implementation of the matrix fill is discussed in the following section. The remainder of the paper is devoted to a discussion of the parallel LU factorization procedure.

4. Parallel Matrix Fill

The matrix entries arising from integral equation formulations are usually completely independent expressions, and therefore the matrix fill task is easy to parallelize with high efficiency [2-4, 12-14]. In our initial implementation, every processor or node performs the identical initialization tasks prior to the matrix fill. Assuming P+1 available processors (or nodes) with one used as a dedicated processor, the task of computing each matrix entry is partitioned into P + 1 subtasks and distributed among P processors and dedicated processor. Each subtask involves computing a block of columns (or rows) consisting of either int(N/P) or int(N/P)+1 columns (or rows) of the matrix. When each node finishes computing it sends its block to the dedicated processor, which assembles the global matrix. Thus, each processor does only its share of the computation. In the initial implementation, the dedicated node performed the matrix factorization and solved the matrix equation. There is no communication between nodes during matrix fill since all the computation is carried out locally. Each node only needs to communicate one block of data to the dedicated processor.

If we define speedup as S=(sequential execution time)/(parallel execution time), we are able to achieve almost linear speedup in the matrix fill portion of the program. It is obvious that the limit of the speedup for a P-processor system is P, a ideal speedup. In this case, there is very little overhead in partitioning the data and no message passing is needed during the matrix fill. This is a very simple implementation but it achieves excellent speedup for system with a few thousand unknowns.

Figure 3 shows the speedup achieved for a 1000 unknown system by this approach verses the ideal speedup. There are three speedup curves in Figure 3, labeled *matrix fill*, *optimal*, and *wall clock*. The matrix fill speedup is the ratio of the execution time of the matrix fill in one node to that obtained multi-node. The wall clock time speedup is the ratio of wall clock time used for the entire program execution (including matrix solve) by one node to that required multi-node. The optimal speedup is the ideal case

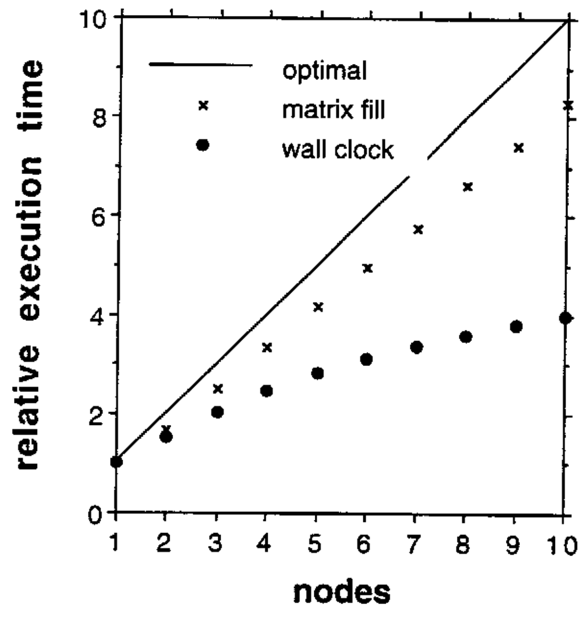

Figure 3. Speedup obtained for a system of order 1000 when the matrix fill task is distributed among various numbers of processors and the matrix solve task is performed serially. The speedup for the matrix fill task and the overall execution (wall clock) are compared with perfect parallelism (optimal).

which achieves 100% parallelism. One can see that the ideal speedup is almost achieved for the matrix fill but poor speedup is observed overall (wall clock speedup). The poor overall performance appears to be the result of two major factors. Let T_{wp} denote the wall clock time of the parallel code execution, and T_{ws} denote the wall clock time of the sequential code execution. The wall clock time speedup S_w is given by $S_w=T_{ws}/T_{wp}$. There are three factors contributed to T_{wp}, the parallel fill execution time, the sequential execution time of the rest of the program (in this case, primarily the matrix factorization), and the communication time between the dedicated node and the other nodes in the application. To improve the wall clock speedup, the matrix solution procedure and the data communication must be optimized.

5. Parallel Matrix Solve

The parallel implementation of the IE-AP matrix fill process using MPI yielded substantial improvement over the serial program. However, it is obviously not efficient to communicate the entire matrix to a single processor for factorization. Although the time to fill the matrix dominates matrix solve time for small problems, this is not the case for systems of order greater than 5000. An additional limiting factor is incurred by communicating the entire matrix to a single processor, which limits the matrix order to a system small enough to fit within the memory of a single processor. The Cornell Theory Center SP thin nodes with 128MB of RAM limit the matrix order to 2400. An out-of-core solution could alleviate this problem, but since we are already computing the matrix elements on many different processors, an easier solution is to eliminate the communication requirement by performing the matrix factorization in parallel.

A large number of articles have proposed and evaluated parallel matrix factorization algorithms [1-2, 11-17]. Since this is a relatively mature area for certain architectures, and libraries for specific machines are in widespread use [17], we chose to investigate parallel numerical libraries specifically recommended for the SP-2. The available libraries were the Portable Extensible Toolkit for Scientific computation (PETSc) and IBM's Parallel Engineering and Scientific Subroutine Library (PESSL). Below, we report on our experience in using these libraries to solve the system of equations in parallel.

5.1 Parallelization Using PETSc

The first attempt to add a parallel matrix solve was made using the PETSc Version 2 Beta 13. PETSc is a parallel numerical library developed at Argonne National Laboratory that attempts to provide a high-level object-oriented interface that automatically manages such details as message passing.

In order to integrate PETSc into our application, we compute the values to be inserted into the PETSc data objects as before. We then call routines telling PETSc where in the global structure to insert the data. Since the programmer is generally unable to directly index individual objects, PETSc maintains a certain level of abstraction, allowing one to change the underlying representation of a data object without actually modifying the program.

After all the data objects have been set up, the application invokes a PETSc solver, which for linear systems is the SLES linear equation solver. In PETSc it is necessary to create a solver context, which may be thought of as an object representing the system. After matrices and vectors have been associated with the system, one calls a generic solve routine to actually solve the system. In its most basic form this process would be done in FORTRAN as follows:

```
call SLESCreate(MPI_COMM_WORLD, sles, ierr)
call SLESSetOperators(sles, A, A,
        DIFFERENT_NONZERO_PATTERN, ierr)
call SLESSetFromOptions(sles, ierr)
call SLESSolve(sles, b, x, iterations,
        ierr)
call SLESDestroy(sles, ierr)
```

This approach has the advantage of being general. By only altering command-line options, it is possible to change the method used to solve the system. The developer can trivially test the utility of different techniques for solving a given system.

After adapting to the peculiarities of the library, we were able to create a working version of our application using PETSc. PETSc allowed us to utilize the routines from the MPI version with little modification, since PETSc uses a slab data partitioning scheme compatible with our original parallel fill routines. This made it possible to call the original fill routines, then easily insert the entries into a PETSc matrix. We then used the generic PETSc solve functions, changing command-line options to experiment with various methods.

We soon discovered that PETSc had severe limitations for the IE–AP type of application, which produces a dense matrix. The PETSc library is targeted at applications with sparse matrices requiring iterative solution. There is no parallel dense LU solver, which forced us to choose between using dense matrices and an iterative solver or using sparse matrices and a block Jacobi solver with LU factorization on individual processor blocks. After some experimentation we chose the latter, although even it did not provide acceptable timing results.

5.2 Results Using PETSc

The PETSc toolkit is not well-suited for our application, as we soon discovered upon viewing timing results. Figure 4 shows the speedup achieved by this approach, which is poor in terms of overall wall clock performance and for the matrix fill part of the computation. The application was executed with the command-line options

```
-mat_mpidense
-pc_type bjacobi
-sub_pc_type lu
```

Despite exhibiting substantial parallel speedup, the

PETSc implementation was much slower than a hand-coded LU routine executing on a single processor.

Upon examining these timings and the output given by running the application with the PETSc option

```
-log_summary
```

we determined that the slow matrix fill time is due almost totally to time spent in the matrix assembly routines. Initially, we believed the problem was the result of our attempting to insert values on the wrong processor and, consequently, being forced to wait for PETSc to communicate the data to the proper node. Unfortunately, we tested a column block partitioning scheme and found even worse results, leading us to believe that PETSc was indeed partitioning as we first suspected but was forced to do some communication during assembly. The execution time could most likely be improved by interleaving some communication between the assembly routines.

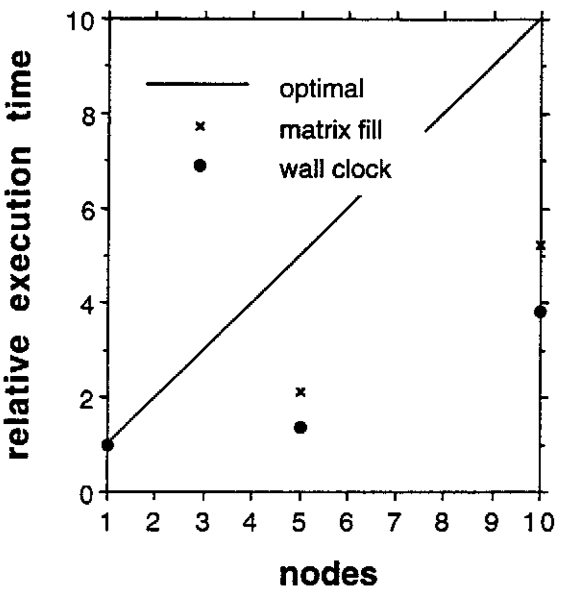

Figure 4. Speedup obtained for a system of order 1000 when the PETSc toolkit is employed. The speedup for the matrix fill task and the overall execution (wall clock) are compared with perfect parallelism (optimal).

The poor times for the solver are easier to explain since it was intended for sparse systems. It is possible that these timing results are not entirely the fault of PETSc, as the library gives the user much control over the algorithms and data types used through the use of command-line options. We may have chosen a poor combination of options or done something else in a manner not conducive to optimal performance. Nonetheless, PETSc does not appear to be the best choice for this application.

5.3 Parallelization Using PESSL

Because of the poor results obtained from the PETSc library, a second attempt was made to parallelize the matrix solver portion of the code using IBM's Parallel Engineering and Scientific Subroutine Library (PESSL). PESSL is a library of numerical subroutines that are optimized for the IBM RS/6000 architecture, leading us to expect superior performance on the Cornell Theory Center SP2. Furthermore, it is compatible with the widely available ScaLAPACK library [17], thus facilitating ports to additional architectures.

The use of PESSL is relatively straightforward. It is able to operate on the data structures created in the original MPI version of our code without modification. It is necessary, however, to set up descriptors giving the global dimensions, block structure, and other partition information for each data object. We found that the method used to partition the matrix had a great impact on performance and, therefore, experimented with several methods. The task of finding an optimal partitioning scheme and block size was hindered, however, by the sheer number of modifications to the code required to implement different partitioning schemes such as slab, cyclic, and block cyclic.

5.3.1 Slab Partitioning

Initially, the data was partitioned exactly as in the MPI version of the code; that is, with each processor holding one contiguous block of rows of the moment matrix. Such a partitioning scheme was very easy to implement, since the MPI matrix fill routines could be used without modification. It was only necessary to set up the PESSL data descriptors for the matrix and vectors and then subsequently to call the LU factorization and solve routines PZGETRF and PZGETRS.

Unfortunately, this method exhibits the drawback of being slow, although it is far better than PETSc. Figure 5 shows the speedup achieved in terms of wall clock time performance of the solve routines for a system of 1000 unknowns. It shows that the PESSL implementation scales better than both the simple implementation and the PETSc implementation. However, the slab partitioning approach imposes a load imbalance on the LU factorization process. To improve the performance of the parallel LU factorization, an alternative partitioning scheme needs to be employed.

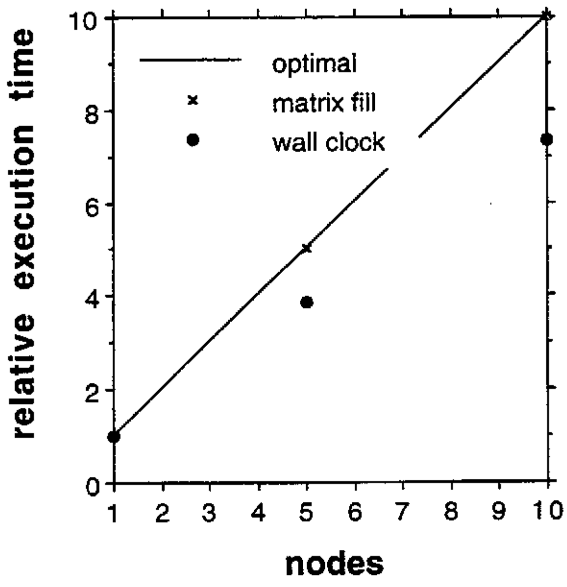

Figure 5. Speedup obtained for a system of order 1000 when the PESSL library was employed with slab partitioning.

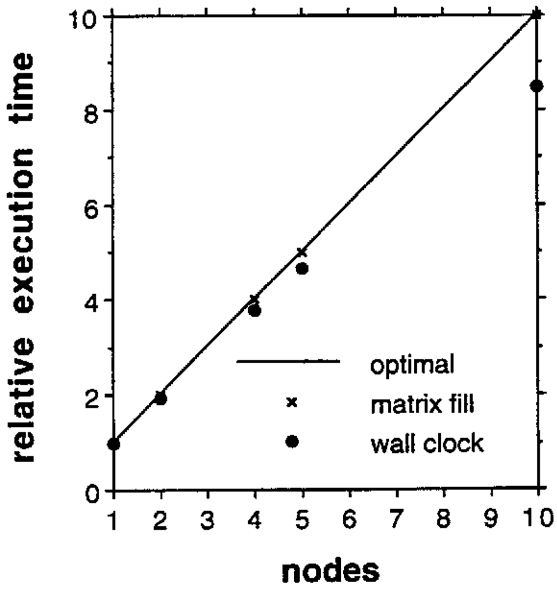

Figure 6. Speedup obtained for a system of order 1000 when the PESSL library was employed with cyclic partitioning.

5.3.2 Cyclic Partitioning

In an attempt to compensate for the deficiencies of the slab partitioning method, a second approach was taken. This time, rows were distributed across processors cyclically with a block size of two rows. For example, if there are two processors, blocks 1, 3, 5, 7, etc. reside on the first processor, while blocks 2, 4, 6, 8, etc. reside on the second processor. Such a scheme helped to balance the processor load and, as a consequence, produced much better results (Figure 6).

The performance obtained using cyclic partitioning could likely be improved by using a larger block size, which would reduce the communication overhead and probably provide greater parallel speedup. The block size of two rows was chosen for convenience, since in the version of the code in which only the matrix fill is done in parallel, row elements are computed in groups of two.

5.3.3 Results Obtained Using Cyclic Partitioning

In an attempt to determine the ability of the modified application to scale to large problems, we executed it on systems consisting of 3060, 5520, and 10240 unknowns. The cyclic PESSL version of the program was altered so that the coefficient matrix is dynamically allocated to only use the space needed on a given node, making the solution of these systems possible. A 40-node SP machine was used to solve a system with 10240 unknowns, requiring about 1.9 hours wall clock time to execute. To demonstrate scalability, we ran the program for a 5520-unknown system using both 10 SP nodes and 20 SP nodes. The wall clock time of the execution on 20 SP nodes is almost one half of the wall clock time of the 10 SP node case.

6. Conclusion

This paper described the parallelization of the 3D IE–AP code developed by Aberegg [6]. The IE-AP procedure tends to shift the computational burden away from the matrix solve part of the process and toward the matrix fill part, suggesting that it might be well-suited for parallel-architecture implementation. The IBM RS6000 SP2 was used as a parallel-architecture platform for the implementation.

The matrix fill task was easily parallelized using available MPI instructions. Two implementations of the matrix solve process were explored. While the PETSc implementation did not result in overall speedup, the PESSL implementation did exhibit an improvement over a serial implementation. Timing results are presented to demonstrate the performance.

The parallel-architecture implementation of the IE–AP approach will facilitate systematic testing and evaluation of the formulation on a wide range of scattering targets. Future work will address the overall efficiency of the IE–AP approach.

7. References

[1] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors*, Englewood Cliffs, New Jersey: Prentice-Hall, 1988.

[2] T. Cwik and J. Patterson, *Computational Electromagnetics and Supercomputer Architecture*, PIER 7, Cambridge, Massachusetts: EMW Publishing, 1993.

[3] X. Shen, *Massive Parallel Processing Applied to Computational Electromagnetics*, PhD Dissertation, Syracuse University, 1994.

[4] X. Shen, G. E. Mortensen, C. C. Cha, G. Cheng, and G. C. Fox, "Parallelization of the Parametric Patch Moment Method Code," *Digest of the 1995 ACES Symposium*, Monterey, CA, March 20-24, 1995.

[5] K. R. Aberegg and A. F. Peterson, "Application of the integral equation–asymptotic phase method to two-dimensional scattering," *IEEE Trans. Antennas Propagat.*, vol. 43, pp. 534-537, May 1995.

[6] K. R. Aberegg, *Electromagnetic scattering using the integral equation–asymptotic phase method.* PhD Dissertation, Georgia Institute of Technology, Atlanta, 1995.

[7] K. R. Aberegg, A. Taguchi, and A. F. Peterson, "Application of higher-order vector basis functions to surface integral equation formulations," *Radio Science*, vol. 31, pp. 1207-1213, September-October 1996.

[8] A. F. Peterson and K. R. Aberegg, "Parametric mapping of vector basis functions for surface integral equation formulations," *ACES J.*, vol. 10, pp. 107-115, November 1995.

[9] A. F. Peterson, "The interior resonance problem associated with surface integral equations of electromagnetics: Numerical consequences and a survey of remedies," *Electromagnetics*, vol. 10, pp. 293-312, 1990.

[10] A. F. Peterson, S. L. Ray, and R. Mittra, *Computational Methods for Electromagnetics*, New York: IEEE Press, 1998.

[11] D. I. Kaklamani and A. Marsh, "Benchmarking high-performance computing platforms in analyzing electrically large planar conducting structures via a parallel computed method of moments technique," *Radio Science*, vol. 31, pp. 1281-1290, Sep.-Oct. 1996.

[12] T. Cwik, "Parallel Decomposition Methods for the Solution of Electromagnetic Scattering Problems", *Electromagnetics*, Vol. 12, pp. 343-357, 1992.

[13] J. P. Brooks, K. K. Ghosh, E. Harrigan, D. S. Katz, and A. Taflove, "Progress in CRAY-based algorithms for computational electromagnetics," in T. Cwik and J. Patterson, *Computational Electromagnetics and Supercomputer Architecture*, PIER 7, Cambridge, Massachusetts: EMW Publishing, 1993.

[14] S. D. Gedney, A. F. Peterson, and R. Mittra, "The moment method solution of electromagnetic scattering problems on MIMD and SIMD hypercube supercomputers," in T. Cwik and J. Patterson, *Computational Electromagnetics and Supercomputer Architecture*, PIER 7, Cambridge, Massachusetts: EMW Publishing, 1993.

[15] D. S. Scott, "Solving large out-of-core systems of linear equations using the Intel iPSC/860," in T. Cwik and J. Patterson, *Computational Electromagnetics and Supercomputer Architecture*, PIER 7, Cambridge, Massachusetts: EMW Publishing, 1993.

[16] T. Cwik, R. van de Geijn, and J. Patterson, "Application of massively parallel computation to integral equation models of electromagnetic scattering", J. Opt. Soc. Am. A, Vol. 11, No. 4, pp. 1538-1545, April 1994.

[17] ScaLAPACK User's Guide may be found online at: http://www.netlib.org/scalapack/slug/scalapack_slug.html