Parallel Computation of Three-Dimensional Nonlinear Magnetostatic Problems

David Levine* **William Gropp*** **Kimmo Forsman†** **Lauri Kettunen†**

Abstract

We describe a general-purpose parallel electromagnetic code for computing accurate solutions to large computationally demanding, 3D, nonlinear magnetostatic problems. The code, CORAL, is based on a volume integral equation formulation. Using an IBM SP parallel computer and iterative solution methods, we successfully solved the *dense* linear systems inherent in such formulations. A key component of our work was the use of the `PETSc` library, which provides parallel portability and access to the latest linear algebra solution technology.

1 Introduction

Electromagnetic field computation is an important design tool that has been used in many applications since its emergence in the early 1960s. The first applications of electromagnetic field computation were driven mainly by two broad subject areas—accelerator physics and rotating electrical machinery (motors and generators). When the use of electromagnetic field design and analysis techniques became widespread in these two areas, many other applications were quick to follow. Among the most important of these in industry are magnetic recording, magnetic resonance imaging (MRI), nondestructive testing, position sensing, modeling electronic components and integrated circuits, and studies involving high-speed magnetically levitated (Maglev) ground transportation.

The ever-increasing use of electromagnetic field computation makes improvements in solution accuracy and problem size vital. For example, currently there is a great demand to examine electromagnetic compatibility problems at the circuit board and even at the integrated circuit level. So far, only simplified calculations in two and three dimensions have been performed. For practical applications, however, problems with millions of degrees of freedom will need to be solved.

Furthermore, because of increased competitiveness in industry, the turnaround time in the design phase must also be reduced. Essentially this means being able to solve more problems faster to arrive at an optimal design in shorter time. As an example, in the magnetic recording

*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4801. This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

†Tampere University of Technology, FIN-33610, Tampere, Finland.

industry, users solve two-dimensional problems in minutes and three-dimensional problems in hours. A typical mode of operation is to do several variations on a design study in a single day. Currently, users cannot do enough three-dimensional studies because of the lengthy solution time and their desire to study many design variations quickly.

For the past twenty years most electromagnetic field computations have been done by using programs based on finite-element methods and run on sequential computers. However, the limitations mentioned above (accuracy, problem size, and solution time) make further advances with this approach more and more difficult. We believe that our approach—combining integral equation methods, iterative linear system solvers, and massively parallel computers—can address many of the current limitations and provide timely and accurate solutions to large, computationally demanding problems.

This paper discusses the CORAL code we have developed to implement our approach. CORAL is an integral equation code that runs on massively parallel processors (MPPs) and solves three-dimensional, nonlinear magnetostatic problems. By combining iterative linear solvers and by exploiting the large amounts of main memory and disk space available on MPPs, we can efficiently solve the dense linear systems inherent in an integral equation formulation. The use of the `PETSc` library provides both parallel portability and access to the latest linear algebra solution technology.

2 Integral Equation Methods

In this section we compare integral equation methods with the more traditional finite-element approach used in electromagnetic programs. We then briefly discuss the integral equation formulation we use. More complete details of the integral equation approach may be found in [8].

2.1 Comparison of Finite Elements and Integral Methods in Electromagnetic Programs

The first widely used three-dimensional nonlinear magnetostatics program was the integral code GFUN [11]. However, beginning in the mid-1970s, finite-element programs came more into favor. A comparison of the advantages of the two approaches can help illustrate how this situation evolved and why we believe integral methods deserve reexamination at this time.

Three advantages of finite-element programs stand out. First, the theory, including error estimation, is more developed for the finite-element method (FEM). Essentially, it was taken over directly from structural finite-element programs. Second, on a sequential computer, solving a very large problem is faster with finite elements. This speed results because the finite-element method leads to sparse matrices whose time of solution scales as $O(m \log m)$, where m is the degrees of freedom in the finite-element formulation. Integral equation methods (IEMs), on the other hand, lead to dense matrices for which the solution time with direct methods scales as $O(n^3)$, where n is the number of degrees of freedom in the integral equation formulation. In addition, IEMs have a matrix definition time that scales as $O(n^2)$ and can be significant (note, however, that the m for FEMs may be much larger than the n for IEM, as described below.) Historically, these characteristics have meant that in practice IEMs have either taken too much computation time compared with finite-element programs to achieve the same level of accuracy, or have not used enough elements to obtain accurate solutions. Third, much less physical memory is required for the storage of the sparse matrix arising from the finite-element method than for the dense matrix arising from integral methods.

Nevertheless, we emphasize that IEMs have a number of natural advantages over FEMs. First, only the active regions need to be discretized. Therefore, in a problem with motion, there is no need to adjust the mesh in the region connecting the moving and stationary components, since that region is not meshed. This is a major advantage when the domain is not connected. Also, not meshing the air region saves a lot of time in setting up the problem and eliminates many—often most—elements.

Second, far-field boundary conditions are automatically taken into account. There is no danger of error from taking the boundaries too close; nor is there a need to find how far away to place them by trial and error. Even a crude integral model will give a good estimate of the fringing field.

Third, fields in the air region show smooth and realistic variation. Variation is not determined by the order or geometry of the mesh, as it is for finite-element codes. This is a big advantage when the variation must be known very accurately, as in MRI or accelerator magnets.

Fourth, for eddy current problems there is no need to keep track of which elements are in motion and which are not, since the mesh is fixed to the region of interest. Conversely, with FEMs it is necessary to keep track of which elements connect moving and stationary elements—a task that can be quite difficult.

Finally, integral methods readily lend themselves to parallel processing. Both the evaluation of the different matrix elements and the determination of the field at different points after the problem is solved can be done completely in parallel. Indeed, several methods have already been developed to efficiently solve dense systems of linear equations on parallel computers.

2.2 Numerical Formulation

Motivated by these advantages of IEMs we developed the CORAL code. CORAL [7, 9], solves three-dimensional, nonlinear magnetostatics problems. The formulation we use in CORAL is based on the idea of superposition of fields from current and magnetization sources. We denote magnetic flux density with B, magnetic field strength with H, magnetization with M, permeability with μ, and susceptibility with χ. Let H^s stand for the magnetic field strength from current sources in the absence of magnetic materials. The field component from magnetization of materials is denoted with H^m. The field from source currents can be integrated from Biot-Savart's law and therefore is known a priori. Arranging the unknown terms on the left and the source terms on the right, we get

$$H - H^m = H^s \ . \tag{1}$$

The H^m-field at point r, resulting from a distribution of M in domain V, can be integrated such that

$$H^m(r) = grad\left[\frac{-1}{4\pi} \int_V \frac{M(r') \cdot (r - r')}{|r - r'|^3} dv' \right] . \qquad (2)$$

Since M can be given in terms of H, namely,

$$M = \chi(|H|)H, \qquad (3)$$

we get

$$H - H^m(\chi, H) = H^s \ , \qquad (4)$$

and hence the fields on the left-hand side can be written in terms of H, if χ is known.

The formulation employed in CORAL is now developed by multiplying the divergence condition of B,

$$div\, B \ = \ div\, \mu H \ = \ 0 \ , \qquad (5)$$

with an appropriate test function, summing over V, substituting Eq. (4) into (5), and applying integral relationships (i.e., theorems analogous to Green's first identity) to get

$$\int_V \mu H \cdot h' - \int_V \mu H^m(\chi, H) \cdot h' = \int_V \mu H^s \cdot h', \quad \forall h', \qquad (6)$$

where h' is a test function that is a gradient field.

In order to establish a discrete problem, the magnetic field H is approximated in the space W^1 spanned by "edge elements," that is, Whitney elements (see below) of degree $p = 1$. The system of equations is developed from Eq. (6) by choosing h' the same as the basis functions of the kernel of curl W^1. The numerical problem implies the tangential continuity of H at all points, whereas the normal continuity of B is satisfied only in the weak sense.

In CORAL, the physical interpretation of the unknown variable is the line integral of the magnetic field H along an edge. In magnetostatics, this integral is equal to the difference in scalar potential between the end nodes of the edge; hence for a (simply) connected region, the number of unknowns is one less than the number of nodes. In practice the line integrals along a spanning tree of the mesh are the independent variables chosen; from them, all other line integrals can be determined.

In magnetostatics the problems are typically nonlinear, because the magnetic properties of the materials depend on the field strength. These kinds of problem are solved iteratively; an initial guess for χ is inserted, and the nonlinearity is taken into account by solving successive problems updating the material data at each cycle. After a few cycles the Newton-Raphson method is applied to accelerate the convergence.

3 The CORAL Program

CORAL uses a three-dimensional tetrahedral mesh generated by the PROBE [4] mesh generator. The mesh uses Whitney elements, a class of finite elements introduced by Bossavit in connection with computational electromagnetics [1]. Whitney elements differ from traditional finite-elements in that the degrees of freedom are related to all the simplices in a simplicial mesh (i.e., the nodes, edges, facets, and volumes).

CORAL is written primarily in Fortran 77 with a few routines written in C. The main parts of CORAL are routines that find a spanning tree, form the paths corresponding to the co-tree edges, generate the integral equation and Jacobian matrices, compute the terms on the right-hand side, and solve the system of equations. Below, we describe the software tools used to parallelize CORAL and the structure of the parallel code.

3.1 Parallel Software Tools

Our goal was to develop a general-purpose code that would run on both MPPs and workstation clusters. For this reason we focused on the message-passing programming model. To satisfy the portability requirement, not only the message-passing primitives, but also the linear algebra routines and I/O facilities must be portable. Our solution to this requirement was to use `PETSc` (Portable and Extensible Tools for Scientific Computing) [6], a large toolkit of software for portable, parallel (and serial) scientific computation.[1]

The two components of `PETSc` that we used were the `Chameleon` and `PSLES` (Parallel Simplified Linear Equation Solvers) libraries. `Chameleon` is a second-generation message-passing system that provided a uniform way to access third-party and vendor-specific message-passing libraries.[2] `Chameleon`'s message-passing calls needed no changes to run on either a cluster of workstations or MPP computers. `PSLES` is an easy-to-use, efficient, portable parallel library for solving systems of linear equations. `PSLES` supports a variety of iterative methods and preconditioners, and a dense LU solver. `PSLES` allows the specification of options such as the solver algorithm, choice of preconditioning matrix, and setting of the tolerance. It accepts many matrix formats, including dense and sparse. Because it is built on top of `Chameleon`, `PSLES` runs on most distributed-memory architectures.

In `PSLES`, a simple interface hides the algorithm and parameter choices and allows a user to easily experiment with different methods. For example, Figure 1 shows the code fragment from CORAL that solves linear systems. This fragment is contained within an outer loop that solves the nonlinear problem. The functions `SpDnCreateFromData` and `PSPCreate` create a matrix in `PSLES` format from the user-specified data structure `matrix`, the submatrix dynamically allocated by each processor (see Section 3.3). The `If` block chooses the algorithm to solve the linear systems. The choice is specified to the `PSVCreate` call via the variable `method`, along with `pmat`, a pointer to the processor's submatrix. `PSVCreate` returns the pointer `ctx` used in the succeeding `PSLES` function calls. The `PSVSetPBDDDomainsNumber` function specifies the number of blocks to use if block diagonal preconditioning is used. The `PSVSetUp` call allocates scratch memory and sets to default values parameters and options not otherwise set. Finally, `PSVSolve` solves the system of equations with right-hand side `b`. The answer is returned in the array `x`.

[1] We used PETSc 1.0 in this work. PETSc 2.0 is now available [5].

[2] At the time of this work, MPI, the Message Passing Interface, had not yet been defined.

```
smat = SpDnCreateFromData(m, m, n, matrix)
pmat = PSPCreate( smat, m )

If      ( precnd .eq. PSVJacobi ) then
    method = PSVJacobi
Else If ( precnd .eq. PSVPBDD   ) then
    method = PSVPBDD
Else If ( precnd .eq. PSVNOPRE  ) then
    method = PSVNOPRE
Else
    method = PSVLU
Endif

ctx  = PSVCreate(pmat, method)
Call   PSVSetPBDDDomainsNumber( ctx, numbdd );
Call   PSVSetUp(ctx)
its  = PSVSolve(ctx, b, x )
```

Figure 1: Code fragment from CORAL for solving linear systems using PSLES

3.2 Input

CORAL reads input from several files: a binary file containing data describing the material from which the magnet is made, as well as the geometry and density of the electromagnetic coils; a description of the tetrahedral mesh elements (spatial coordinates, neighbor lists, and which nodes define an element); and a file containing parameters of the solution method. CORAL is structured so that only one processor reads the input files and broadcasts the data to other processors.

3.3 Memory Allocation

For parallel computation, the integral equation matrix and related arrays were decomposed. The other data structures were replicated on each processor. Fortran's static memory allocation, however, is inappropriate for distributed arrays, since we wished to allocate only as much memory as necessary to hold the processor's portion of the arrays, but these sizes were not known until run time. Therefore, we wrote several C language routines.

3.4 Matrix Generation

Each nonlinear iteration requires the generation of a new integral equation or Jacobian matrix. The full matrices are of order $n_{edges} \times n_{edges}$ (n_{edges} is the number of edges in the spanning tree). These matrices were decomposed rowwise as shown in Figure 2. Each processor's submatrix is of order $n_{edges}/n_{procs} \times n_{edges}$ (n_{procs} is the number of processors being used).

Figure 2: Rowwise decomposition of matrix in CORAL

Since the equations are related to edges, and a node may belong to more than one edge, some processors may compute the same data for a node that has a contribution to more than one equation. An advantage of this is that no data broadcasting between the processors is needed. In addition, only a small amount of overlapping data is computed on two or more processors.

Each nonlinear iteration, the matrix elements are computed from the solution to the linear system from the previous nonlinear iteration and some geometry-dependent integral terms that arise from Eq. (2). Computing these integral terms is a time-consuming part of the integral equation matrix generation. However, since these terms depend only on the problem geometry, they may be computed once and read each nonlinear iteration when a new matrix is generated. This approach saves a considerable amount of processing time and is a critical part of the integral formulation.

The total amount of data storage required for the integral terms is $n_{elements} \times n_{edges} \times 3$, and for large problems can easily exceed main memory ($n_{elements}$ is the number of tetrahedral mesh elements). This is true even after the matrix decomposition when each processor needs access only to its $n_{elements} \times n_{edges}/n_{procs} \times 3$ part of the data. Therefore, the computed results were stored to a disk file and read each successive nonlinear iteration.

The row-decomposition of the matrix used here is not the optimal one for direct numeric factorization; in that case, a variant of a 2-d block-cyclic decomposition is appropriate. However, such a decomposition is far less natural for the matrix assembly part of the code. In addition, the decomposition used here is efficient for the iterative solvers.

3.5 Linear System Solution

Several special features of the linear systems arise in CORAL. First, the matrix resulting from Eq. (6) is asymmetric. Second, each system of linear equations arises from an outer nonlinear problem and so may need only a relatively low accuracy solution. Third, the actual matrix, while dense, has many "small" elements. Finally, the size of the matrices to be solved varies significantly according to the mesh refinement and desired solution accuracy.

Traditionally, dense systems of linear equations are solved directly by Gaussian elimination. However, the solution time for Gaussian elimination scales as $O(n^3)$, where n is the order of the matrix, and can be prohibitive for large values of n. An alternative approach is to use iterative solvers. In general, the solution time for iterative methods involving dense matrices scales as $O(n^2)$ per iteration. The number of iterations, I, is heavily dependent on the initial guess and the choice of matrix preconditioner. While I may grow as a function of n, in practice this growth is quite slow with appropriate preconditioners.

In CORAL we believed there were several advantages to using iterative methods. First, to solve a nonlinear problem, one must solve a related sequence of linear systems. With iterative methods, we would be able to use the solution to one of the linear systems in the sequence as the starting solution to the next linear system in the sequence, and hopefully reduce the solution time. Second, for some nonlinear problems it may not be necessary to solve the early linear systems in the sequence to high accuracy. Unlike direct methods, iterative methods allow an early exit from the solution procedure with an approximate solution. Finally, for large problems, we believed iterative methods would be faster than direct methods. This will be true asymptotically if $O(n^3) > I\,O(n^2)$ and we expected slow growth in I as a function of n.

4 Computational Experiments

In this section we report on our computational experiments. We focus on the computational performance. Details of the accuracy of the calculated electromagnetic field are given in [8].

4.1 Test Problems

We present results for two test problems. TEAM 13 is one of the international TEAM (Testing Electromagnetic) benchmark problems [10]. It consists of thin steel plates that are exited below the saturation level. One of the difficulties when modeling TEAM 13 is the narrow air gap between the steel plates, which requires a large number of elements to achieve accurate results below the saturation level. Figure 3 shows the geometry of TEAM 13. TEAM 13 is a problem for which an integral equation approach works well since only the magnetic regions have to be discretized. Moreover, because the steel plates are thin, a relatively high number of elements can be concentrated close to the air gap and to the bend of the plates. Finite-element methods require many elements in the air close to the gap and the bend to compute accurate solutions. TEAM 13 was run using a mesh of 5,087 tetrahedral mesh elements, resulting in a matrix of order 1,446.

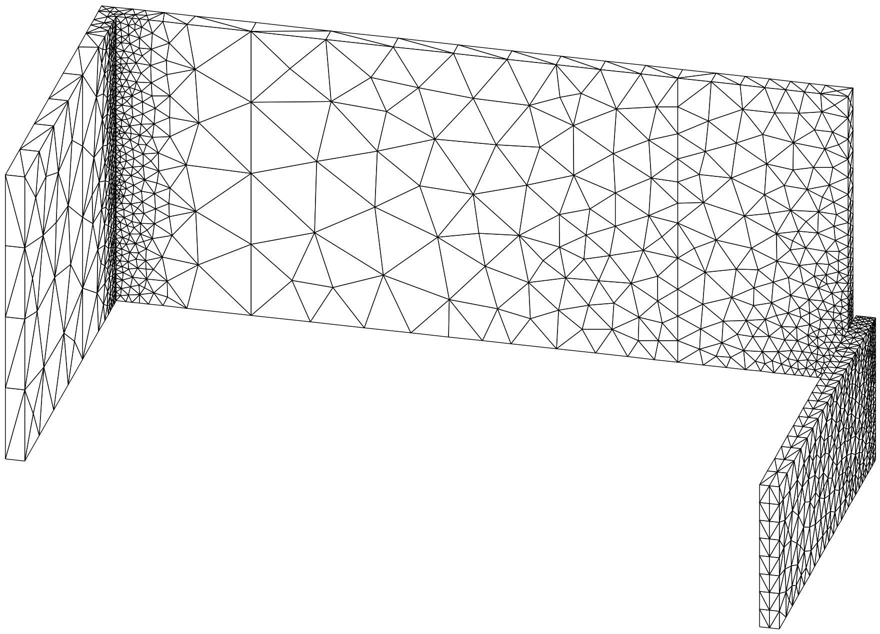

Figure 3: Geometry of TEAM 13

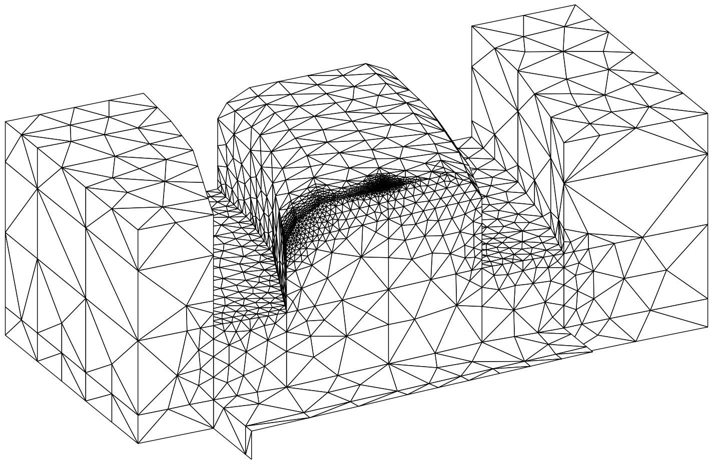

Figure 4: Geometry of APS Dipole Magnet.

Table 1: TEAM 13 Timings (sec.), One Block Per Processor

	4 Proc.	8 Proc.	16 Proc.
Nonlin Iter.	11	> 15	>25
Time	252	>1800	>3600

The second test problem was a dipole magnet from the Advanced Photon Source at Argonne National Laboratory. Figure 4 shows the geometry of this magnet. The poles are curvy, and the magnet has shielding plates in front of the coils. In addition, the ends of the poles are beveled. Since the geometry is nontrivial, this problem is a challenging test for integral formulations. Because of symmetry, only one-fourth of the APS dipole magnet had to be discretized. The APS dipole magnet was run using a mesh of 14,149 tetrahedral mesh elements, resulting in a matrix of order 3,242.

4.2 Results

The results we present were computed on an IBM SP parallel computer with 128 RS/6000 model 370 processors, each with 128 Mbytes of memory and a one Gbyte local disk. Compilation was done by using the IBM `xlf` Fortran compiler with level O3 optimizations. `Chameleon` generated EUI-H message-passing calls.

In the work reported in [3] we compared LAPACK's LU solver [2] with several different iterative solvers and preconditioners on a DEC Alpha workstation. In one set of tests, generalized minimal residual (GMRES), bi-conjugate gradient stabilized (Bi-CGSTAB), and conjugate gradient stabilized (CGS) using a band preconditioner were all more efficient than LU, with GMRES being the most efficient. In another set of experiments, LU was compared with GMRES using five different preconditioners. Of these, block diagonal and a sparse preconditioner provided consistently better results than the LU solver.

To successfully use GMRES on a parallel computer, a key issue was to develop an appropriate *parallel* preconditioner. We first implemented a block diagonal preconditioner (BDD) that used one block per processor. Using the same number of blocks as the number of processors has favorable implementation features (i.e., no interprocessor communication is necessary). Another advantage of a BDD preconditioner is that a sequential LU solver can be applied to the solution of the individual blocks.

Table 1 shows the performance of GMRES with the parallel BDD preconditioner on TEAM 13, as a function of the number of processors. The "$>$" in front of an entry indicates the solution did not converge. The results show that while easily implemented, this version of BDD does not scale to larger numbers of processors in the sense that the number of iterations required grows with the number of blocks. Testing on the APS Dipole magnet and other problems showed similar results.

The results in Table 1 led us to develop a block diagonal preconditioner that supports multiple processors per block, or multiple blocks per processor. With this we performed some informal tests to identify an "appropriate" number of blocks that would yield good (but not necessarily optimal) results. Typically, this was one block for each 400–800 equations.

Table 2: Timing Results (sec.) as a Function of the Number of Processors

Problem	4	8	16	32	64	128
TEAM13	301	159	119	73	146	247
APD Dipole	X	X	5951	3419	2506	6168

Table 3: Timing (sec.) of Parallel LU and GMRES, 3241 Equations

Processors	4	8	16	32	64
LU	275.7	143.1	76.7	40.2	24.8
GMRES (147 iter.)	109.9	57.8	29.7	17.0	10.9

Table 2 contains the total solution time to solve both test problems as a function of the number of processors. The number of nonlinear iterations required to solve TEAM 13 was 11, and for the APS dipole magnet it was 45. LU was used to solve the first nonlinear iteration and GMRES with the new BDD preconditioner successive nonlinear iterations. The number of BDD blocks was fixed at four and the solution from the previous nonlinear iteration was used as a starting point.

Two points are notable. First, compared with the results in Table 1, GMRES converged in each case with an "appropriate" block size. Second, a good speedup was achieved as long as the granularity (number of equations) per processor was sufficient. As the granularity decreased, so did the speedup. In fact, when the granularity got too small, the parallel computing overheads degraded performance overall.

To show that GMRES compares favorably with LU in a parallel computing environment, we reproduce Table VI from [3] in Table 3. The times given are for solving *only* the second nonlinear iteration of the APS dipole magnet. GMRES was run with the BDD preconditioner with the number of blocks fixed at four, and the solution to the first nonlinear iteration used as a starting point.

5 Conclusions

Prior to our work, it was thought that integral equation methods were computationally too demanding to be a viable approach. For that reason scientists and engineers have not taken advantage of IEMs when modeling electromagnetic calculations. In our work with CORAL, however, we have shown that the combination of large-scale parallel computing and iterative linear solvers make integral equation methods a practical approach for solving three-dimensional, nonlinear magnetostatic problems.

The IBM SP was a key factor in our success. The large main memory on each processor allowed us to store in-core very large, dense matrices. Also, the large amount of disk space on each processor allowed us to "precompute" several large, computationally demanding results, store them to disk, and read them each nonlinear iteration, saving considerable computation time.

The PSLES component of the PETSc library allowed us to easily test different solvers and preconditioners. In fact, we often run CORAL using LU to solve the first nonlinear iteration, and GMRES for the successive nonlinear iterations. Both solvers are integrated seamlessly together in the code. We found that in order to outperform LU factorization, sophisticated implementations of the preconditioning matrix that scale well on large numbers of processors are required.

The Chameleon component of PETSc was also an important part of our work. Using Chameleon, we were able to do the initial development and debugging of CORAL on a workstation network and, with no source code changes, port to the IBM SP. CORAL was parallelized by using just the broadcast and reduction primitives of Chameleon in combination with the PSLES solvers—*no* explicit message-passing was required! The success of Chameleon and PSLES in the CORAL code suggests that libraries based on MPI will go a long way toward solving the problem of porting codes to parallel computers.

Our next goal is time-dependent problems with moving objects in which eddy currents arise. With integral methods there is no need to keep track of which elements are in motion and which are not, since the mesh is fixed to the region of interest. The fact that one need not discretize air and that exterior boundary conditions are automatically incorporated offers significant advantages compared with finite-element methods.

Acknowledgments

We thank Sean Pratt, Jennifer Rovegno, Jukka Salonen, Diana Tabor, Hania Yassin, and Vector Fields Inc. for their assistance. The computations were performed on the IBM SP in Argonne's High-Performance Computing Research Facility.

References

[1] A. Bossavit. Whitney forms: A class of finite elements for three-dimensional computations in electromagnetism. In *IEE Proceedings*, volume 135, Pt. A, pages 493–499, 1988.

[2] E. Anderson et al. *LAPACK Users's Guide*. SIAM, Philadelphia, 1992.

[3] K. Forsman, W. Gropp, L. Kettunen, D. Levine, and J. Salonen. Solution of dense systems of linear equations arising from integral formulations. *IEEE Antennas and Propagation* (in press).

[4] K. Forsman and L. Kettunen. Tetrahedral mesh generation in convex primitives by maximizing solid angles. *IEEE Transaction on Magnetism*, 30:3535–3538, 1994.

[5] W. Gropp, L. Curfman McInnes, and B. Smith. PETSc World Wide Web home page. http://www.mcs.anl.gov/petsc/petsc.html, 1995.

[6] W. Gropp and B. Smith. Scalable, extensible, and portable numerical libraries. In *Proceedings of Scalable Parallel Libraries Conference*, pages 87–93. IEEE, 1994.

[7] L. Kettunen. *Volume Integral Formulations for Three Dimensional Electromagnetic Field Computation.* PhD thesis, Tampere University of Technology, 1992. Publication 86, Tampere, Finland.

[8] L. Kettunen, K. Forsman, D. Levine, and W. Gropp. Volume integral equations in nonlinear 3-d magnetostatics. *International Journal of Numerical Methods in Engineering*, 38:2655–2675, 1995.

[9] L. Kettunen and L. Turner. A Volume integral formulation for nonlinear magnetostatics and eddy currents using edge elements. *IEEE Transactions on Magnetics*, 28(2):1639–1642, 1992.

[10] T. Nakata, N. Takahashi, K. Fujiwara, K. Muramatsu, T. Imai, and Y. Shiraki. Numerical analysis and experiments of 3-D non-linear magnetostatic model. In *Proceedings of TEAM Workshop on Computation of Applied Electromagnetics in Materials*, pages 308–310, Okayama, Japan, 1990.

[11] M. J. Newman, L. Turner, and C. W. Trowbridge. GFUN: An interactive program as an aid to magnet design. In *Proceedings 4th Magnet Technology Conference*, Brookhaven, 1972.