Solution of Dense Systems of Linear Equations Arising from Integral Equation Formulations

Kimmo Forsman,[1] William Gropp,[2] Lauri Kettunen,[1] David Levine,[2] and Jukka Salonen[1]

[1]Tampere University of Technology, Laboratory of Electricity and Magnetism, P.O. Box. 692, FIN-33101 Tampere, Finland

[2]Argonne National Laboratory, Mathematics and Computer Science Division, 9700 South Cass Ave., Argonne, IL 60439, U.S.A.

***Abstract*—This paper discusses efficient solution of dense systems of linear equations arising from integral equation formulations. Several preconditioners in connection with Krylov iterative solvers are examined and compared with LU factorization. Results are shown demonstrating practical aspects and issues we have encountered in implementing iterative solvers on both parallel and sequential computers.**

I. Introduction

In numerical solutions of Maxwell's equations, integral formulations are attractive because they enable researchers to address problems without discretizing air regions. This property is especially useful for problems with moving objects. However, integral equation formulations inherently lead to dense systems of equations, and therefore they are often thought to be computationally expensive.

The main difficulties with integral formulations are the amount of memory needed to store the system matrix and the solution time of the dense system of linear equations. In practice, the storage requirements may easily exceed the computer's memory, and the solution time may also be unacceptable. To make integral methods more appealing, effort must be devoted to solving large dense linear systems efficiently. In addition, since parallel computing is an obvious way to increase the computer's memory capacity and computation speed, it is important to develop efficient linear equation solvers for both sequential and parallel computers.

Several factors must be taken into account when considering a solver: (1) the number of operations it needs, including its O-complexity, (2) its storage requirements, and (3) its computer implementation. In a parallel machine one should also minimize the amount of data broadcast between the processors to gain efficiency.

Our goal in this paper is to discuss various issues we have encountered in trying to find and implement efficient sequential and parallel solvers for a magnetostatic volume integral formulation. The corresponding code is called GFUNET [1], and the system matrices it generates are asymmetric. We have tested both iterative Krylov methods and LU factorization.

The rest of this paper is organized as follows. The Basic Linear Algebra Subprograms are discussed in Section II. The Krylov class of iterative methods are discussed in Section III. The preconditioners we implemented are described in Section IV. Our numerical experiments are discussed in Section V. Finally, our conclusions are given in Section VI.

II. BLAS

The execution time of a computer program depends not only on the number of operations it must execute, but also on the location of the data in the memory hierarchy of the computer. The time the processor needs to access data varies within the memory hierarchy. For optimal performance data movement within the memory hierarchy should be minimized.

Standard programming languages such as Fortran or C, do not have tools to explicitly control the data movement within the memory hierarchy. However, many computers provide machine-optimized versions of the Basic Linear Algebra Subprograms (BLAS) [2], low-level linear algebra routines that optimize the use of the memory hierarchy. In the case of dense system of linear equations, use of the BLAS can significantly decrease the total solution time. Efficiency is achieved by splitting a dense matrix into blocks that are operated on in a manner that minimizes the number and cost of the memory accesses.

The advantage of using the BLAS is shown in Table I. Systems of equations generated by GFUNET are solved on a DEC 3000-700 AXP workstation using the LU solver from Numerical Recipes [3] and the LU solver from LAPACK [4], where machine-optimized BLAS routines are used. The LAPACK solver is an order of magnitude faster than the Numerical Recipes LU solver.

From a practical standpoint this result means that one should strive to use BLAS operations (matrix-vector and matrix-matrix) for large sets of data (i.e., for "long" vec-

TABLE I
Solution Times Obtained by LU Factorization

Number of Equations	250	500	1000	2500
Numerical Recipes LU	1.02	9.32	89.4	1536
LAPACK LU	0.14	0.79	6.0	108

tors) to code an efficient solver. For example, in many cases it is reasonable to insert additional zeroes into the numerical data in order to create large blocks of data for the BLAS routines. This strategy often turns out to be more efficient than executing several successive (Fortran `DO` or C-language `for`) loops for small bits of data. For this reason it is difficult without empirical testing to determine which solver is the most efficient.

III. Krylov Subspace Methods

The iterative solution of the dense nonsymmetric linear system

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

is an $mO(n^2)$ process, where m is the number of iterations. If m is much less than n, iterative solvers can be significantly faster than direct methods.

Krylov methods are one class of iterative methods. At iteration s a Krylov method produces an approximation x_s for (1) of the form $x_s \in x_0 + K_s(r_0, B)$. Here, x_0 is any initial guess for (1), $r_0 = b - Ax_0$, and $K_s(r_0, B) = span\{r_0, Br_0, \ldots, B^{s-1}r_0\}$ is the sth Krylov subspace generated by r_0 and B. The idea is to find an approximation x_s such that $(b - Ax_s)$ is perpendicular to L_s, where L_s is another subspace of dimension s. Different Krylov methods arise from different choices of the subspaces K_s and L_s and from the ways in which the system is preconditioned.

We have written Fortran code based on the TEMPLATES book [5] for the following Krylov methods: generalized minimal residual (GMRES) [6], conjugate gradient squared (CGS) [7], and bi-conjugate gradient stabilized (Bi-CGSTAB) [8]. In our implementation, we have strived to use BLAS routines whenever possible.

A. GMRES

GMRES uses a Gram-Schmidt process to compute an l_2-orthonormal basis $V_s = \{v_1, v_2, \ldots, v_s\}$ of the Krylov subspace $K_s(r_0, A)$. The approximate solution at iteration s is given by $x_s = x_0 + V_s y_s$, where y_s is chosen so that the residual $b - Ax_s$ is minimized. GMRES requires that all vectors in V_s be stored. For this reason it is often considered to be too expensive in both computation time and storage requirements. To alleviate the storage problem, *restarting* has been suggested: the user chooses N_r such that every N_r iterations the set of stored vectors is emptied. In our tests, however, GMRES generally converged quite rapidly so that restarting was not necessary.

B. BiCG-like Methods

The conjugate gradient method (CG) minimizes $f(x) = \frac{1}{2}x^T Ax - b^T x + \frac{1}{2}b^T A^{-1} b$. The x that minimizes f is the solution of $Ax = b$. In each iteration a new search direction d_s is selected such that $d_s^T A d_j = 0,\ 0 \le j \le s-1$, and t is found such that $f(x_s + td_s)$ is minimized. The new approximation is $x_{s+1} = x_s + td_s$. CG is an effective method, but unfortunately it is suitable only for symmetric and positive definite systems. The Biconjugate gradient (BiCG) method is a modification of CG. Instead of forcing search directions to be mutually conjugate, BiCG constructs two mutually A-orthogonal search directions d_i and $\tilde{d}_i$ such that $\tilde{d}_i^T A d_j = 0,\ i \ne j$. The selection of $\tilde{d}_i$ is based on an auxiliary system $A^T \tilde{x} = \tilde{b}$. This means that $x_s \in x_0 + K_s(r_0, A)$ and is orthogonal to $K_s(\tilde{r}_0, A^T)$. Here $\tilde{b}$ and $\tilde{r}_0$ are arbitrary vectors. CGS and Bi-CGSTAB are variants of BiCG that do not require the computation of $A^T x_s$. It has been shown [7] that CGS is faster than BiCG but often has quite irregular convergence behavior. Bi-CGSTAB was developed to have the same convergence rate as CGS at its best, without having the same difficulties. The advantage of BiCG-like methods over GMRES is that they have limited computation and storage requirements in each iteration step.

IV. Preconditioning

The efficiency of an iterative solver depends strongly on the preconditioner. Finding a "good" preconditioner is often difficult. It should be easy to form, it should be a good approximation of A, and the solution u of

$$Mu = v, \tag{2}$$

should be easy to compute. (Vector v in (2) depends on the iterative solver in use.)

Since integral equation formulations typically lead to diagonally dominant matrices, the first choices for the preconditioner are the diagonal of A, diagonal blocks, and a band. However, in our case there are rather large off-diagonal elements in the system matrix due to the spanning tree extraction technique [1], [9] we use. These large elements, far from the diagonal, make it more difficult to find an appropriate preconditioner. Therefore, we have also tested the standard incomplete LU factorization (SILU), and developed a "sparse preconditioner" based on picking the elements of the system matrix that have the largest absolute value compared with the corresponding diagonal elements. The list below contains a brief description of these five preconditioners.

1. *Diagonal*: $M = \text{diag}(A)$.
2. *Block Diagonal*: M is formed by taking q equal-sized diagonal blocks from A. We solve (2) by using complete LU factorization for each block, $M_i u_i = v_i$, $(i = 1, 2, \ldots, q)$.
3. *Band*: M is formed by taking directly from A a band whose bandwidth is bw.
4. *Standard Incomplete LU*: The SILU preconditioner [10] is $M = (L_s + \tilde{D})\tilde{D}^{-1}(\tilde{D} + U_s)$, where L_s and U_s

TABLE II

ADDITIONAL STORAGE NEEDED FOR PRECONDITIONERS

Preconditioner	Additional Storage	$q = 4,\ bw = n/4$ and $nz = 0.05n^2$
Diagonal	0	0
Block Diagonal	n^2/q	$0.25n^2$
Band	$bw \cdot n$	$0.25n^2$
SILU	n	n
Sparse	$7nz$	$0.35n^2$

are the strictly lower and upper triangular parts of A. $\tilde{D}$ is found by requiring $\mathrm{diag}(M) = \mathrm{diag}(A)$.

5. *Sparse*: A sparse preconditioner is formed by letting the nonzeros of M be those elements of A satisfying $|a_{ij}| \geq \tau \min(|a_{ii}|, |a_{jj}|)$, where τ is the dropping coefficient. The number of nonzeros in M is nz. Equation (2) is solved using UMFPACK [11] developed for unsymmetric sparse matrices.

Using the block diagonal, band, and sparse preconditioners, one must decide how large a portion of the system matrix A to include in the preconditioner M. In our case the choice was based on numerical experiments.

Since the system matrix is dense and requires significant memory to store, the additional storage needed for the preconditioner must be considered when comparing efficiency. The additional memory requirements are given in Table II.

V. NUMERICAL EXPERIMENTS

In this section we discuss sequential and parallel experiments we have carried out using the different linear system solvers. The linear systems arise during GFUNET's solution of three-dimensional nonlinear magnetostatics problem, where a related sequence of linear systems is solved. Details of the nonlinear solution method are given in [1].

The first test problem is the international electromagnetic force benchmark TEAM (Testing Electromagnetic Analysis Methods) problem 20 [12]. Because there are small air gaps in the problem, and because of the spanning tree extraction technique we employ, there are large

TABLE III

TIMING OF SOLVERS

Equations	153	1011	2867
$\mathrm{LU}_{\mathrm{Num.Rec.}}$	0.233	95.4	2488
$\mathrm{LU}_{\mathrm{LAPACK}}$	0.037	6.2	133
GMRES	0.044 (16)	2.5 (14)	26 (14)
Bi-CGSTAB	0.065 (14)	3.9 (12)	38 (13)
CGS	0.063 (15)	4.1 (13)	38 (13)

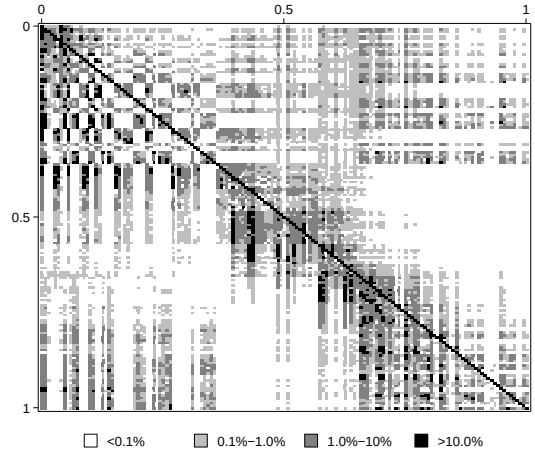

Fig. 1. Absolute values of the system matrix A of TEAM Problem 20 in percents of the maximum absolute value, first nonlinear cycle, $n = 153$.

elements far from the diagonal, as shown in Figure 1. This uneven distribution of large elements makes the problem challenging for iterative solvers.

Table III summarizes the results for TEAM problem 20 using three different computational meshes with corresponding systems of 153, 1,011, and 2,867 linear equations, respectively. The problems were run on a DEC Alpha 3000-600 AXP workstation. Two LU and three iterative solvers are compared. For the iterative methods the band preconditioner with $bw = n/4$ was used. The times shown are the average time in CPU-seconds for the first four nonlinear iterations. The average number of iterative solver iterations each nonlinear iteration is given in brackets. On each nonlinear cycle the solution of $Mx_0 = b$ was used as an initial guess for the iterative solvers. The stopping criterion for the iterative solvers was $\|b - Ax\|_2/\|b\|_2 < \alpha$, where α is the convergence tolerance (we used $\alpha = 10^{-8}$). The results show that all of the iterative solvers are more efficient than the LAPACK LU solver.

Table IV compares the LAPACK LU solver with GMRES using five different preconditioners. Here, three different test problems, TEAM problem 13 [13], TEAM problem 20, and a positron accumulator ring dipole magnet [1], were used. The timings, parameters, and starting guess are the same as those used in Table III. The block diagonal and sparse preconditioners both provide consistently good results which are better than LAPACK's LU solver.

Because sequential computers are limited in their storage capacity and computation speed, parallel computers are an attractive platform for the solution of larger problems. However, programming an efficient solver for a parallel computer is more complex than for a workstation. Algorithms that work well on a sequential machine may not be suitable for parallel implementation. For example,

TABLE IV

TIMING OF GMRES WITH VARIOUS PRECONDITIONERS

Problem	TEAM 13	TEAM 20	PAR-dipole
Equations	1444	1011	1556
LU_{LAPACK}	18	6.2	22
Diagonal	42.6 (195)	3.5 (35)	18 (77)
Block $Diag_{q=4}$	7.2 (23)	2.7 (16)	15 (47)
$Band_{bw=n/4}$	11.5 (34)	2.5 (14)	27 (72)
SILU	66.4 (158)	3.5 (16)	34 (70)
$Sparse_{nz=0.05*n^2}$	11.1 (35)	2.1 (11)	13 (35)

on a parallel computer the implementation of a sparse preconditioner can be a complex procedure. A block diagonal preconditioner based on a parallel LU solver, however, is easier to develop. It is also easier to make efficient use of the BLAS routines on a parallel computer if a diagonal, band, or block diagonal preconditioner is employed.

The parallel 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. The parallel solvers are from `PETSc` (Portable and Extensible Tools for Scientific Computing) [14], a large toolkit of software for portable, parallel scientific computation. The `PETSc` solvers are written to make efficient use of the BLAS.

Table V shows the performance of the parallel LU solver in `PETSc` as a function of the number of processors on a version of the PAR-dipole magnet problem with 7,536 equations. As expected, the results show good speedup can be achieved with LU factorization with an increasing number of processors.

In Table VI we compare the solution times of the parallel LU solver with a parallel implementation of GMRES. The number of blocks used in the block diagonal preconditioner is fixed at four, independent of the number of processors. (With four blocks, all the problems we have tested have converged.) The test problem is the PAR-dipole magnet with 3,241 linear equations. As can be seen, the GMRES solutions are more than twice as fast as the LU solutions.

It should be emphasized that the competitive edge of an iterative solver can be lost, especially on a parallel computer, if the solver does not convergence quickly enough. A good preconditioner and initial guess are vital for the iterative solver. For instance, on the test problems we have tried, the block diagonal preconditioner with $q \geq 5$ was not a good approximate of the system matrix. As a result, GMRES became slower than LU factorization.

VI. CONCLUSIONS

We have shown that iterative solvers are efficient in solving dense and asymmetric systems of linear equations

TABLE V

TIMING of PARALLEL LU-SOLVER, 7536 EQUATIONS

Processors	16	32	64
Solution time	948.2	533.0	308.1

TABLE VI

TIMING OF PARALLEL LU and GMRES, 3241 EQUATIONS

Processors	4	8	16	32	64
LU	275.7	143.1	76.7	40.2	24.82
GMRES (147 iter.)	109.9	57.8	29.7	17.0	10.9

arising from integral equation formulations. The use of machine-optimized versions of the BLAS is essential for optimal performance.

Iterative methods for $Ax = b$ are attractive in particular when one does not demand accuracy from the solution. This is precisely our situation. The solution of $Ax = b$ is needed as part of the iterative solution of a nonlinear set of equations $f(x) = 0$. An accurate solution of $Ax = b$ is needed only when we approach the solution of $f(x) = 0$.

For large problems we have found GMRES to be more efficient than direct methods if a good preconditioner is used. It is faster and more reliable than Bi-CGSTAB and CGS methods. However, as expected, iterative solvers are strongly dependent on the choice of the preconditioner. The efficiency of a preconditioner depends on the problem at hand. In a sequential computing environment, the sparse preconditioner with $nz = 0.05n^2$ worked well on all test problems. The number of iterations depends very little on the problem dimension (provided that the "element density distribution" remains about the same).

On a parallel machine the development of an iterative solver is a more demanding process. However, the results show that GMRES is still superior to LU factorization, if the parallel preconditioner is carefully designed.

ACKNOWLEDGMENT

The use of the Argonne High-Performance Computing Research Facility is gratefully acknowledged. The HPCRF is funded principally by the U.S. Department of Energy Office of Scientific Computing. The work of the second and fourth authors 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.

REFERENCES

[1] L. Kettunen, K. Forsman, D. Levine, and W. Gropp, "Integral equations and nonlinear 3D magnetostatics," *Int. J. Numer. Methods Eng.*, vol. 38, pp. 2655–2675, 1995.

[2] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, "Basic linear algebra subprograms for fortran usage," *ACM Transactions on Mathematical Software*, vol. 5, pp. 308–323, 1979.

[3] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes, The Art of Scientific Computing.* Cambridge University Press, 1992.

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

[5] R. Barret et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, 1994.

[6] Y. Saad and M. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 6, pp. 865–881, 1985.

[7] P. Sonneveld, "CGS: a fast Lanczos-type solver for nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 10, pp. 36–52, 1989.

[8] H. A. van der Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 13, pp. 631–644, 1992.

[9] K. Forsman and L. Kettunen, "Tetrahedral mesh generation in convex primitives by maximizing solid angles," *IEEE Trans. Magn.*, vol. 30, pp. 3535–3538, September 1994.

[10] H. A. van der Vorst, "High performance preconditioning," *SIAM J. Sci. Statist. Comput.*, vol. 10, pp. 1174–1185, 1989.

[11] T. A. Davis, "Users' guide for the unsymmetric-pattern multifrontal package (UMFPACK)," Tech. Rep. TR-93-020, CIS Dept., Univ. of Florida, Gainesville, FL, 1993.

[12] N. Takahashi, T. Nakata, and H. Morishige, "Summary of results for problem 20 (3-d static force problem)," in *Proc. of the Fourth Int. TEAM Workshop*, (Florida International University, Miami, U.S.A.), pp. 85–91, 1994.

[13] T. Nakata, N. Takahashi, and K. Fujiwara, "Summary of results for team workshop problem 13 (3-d nonlinear magnetostatic model)," in *Proc. of the Fourth Int. TEAM Workshop*, (Florida International University, Miami, U.S.A.), pp. 33–39, 1994.

[14] W. D. Gropp and B. F. Smith, "Scalable, extensible, and portable numerical libraries," in *Proceedings of Scalable Parallel Libraries Conference*, pp. 87–93, IEEE, 1994.