Preconditioning techniques for the conjugate gradient solver on a parallel distributed memory computer

Christian Vollaire, Laurent Nicolas

Preconditioning Techniques for the Conjugate Gradient Solver on a Parallel Distributed Memory Computer

C. Vollaire, L. Nicolas
CEGELY - UPRESA CNRS 5005 - Ecole Centrale de Lyon
BP 163 - 69131 Ecully Cedex - France

Abstract–**This paper describes the parallelization of the conjugate gradient algorithm fitted with three types of preconditioning in order to compute large finite element complex sparse system of equations on a distributed memory parallel computer. Parallel performances are analyzed and compared using a problem of 60000 degrees of freedom. The electromagnetic scattering of a plane wave by a perfect electric conducting airplane is finally given as a large example.**

Index terms–**Finite element methods, parallel algorithms, distributed memory systems, sparse matrices.**

I. Introduction

Massively parallel distributed memory computers provide the increase in computing performances required to solve large problems. Indeed, only parallel computation actually enables to modelize real devices because it reduces the computation time and mainly arranges enough memory.

In this paper, we are dealing with the implementation of a Finite Element (FE) formulation for high frequency (HF) electromagnetic scattering problems [1] on a parallel distributed memory computer [2]. Most of the parallel FE codes use domain decomposition techniques to assemble the FE matrix [3-5]: the solver operates first on the different sub-domains and then on the global FE matrix. This method is efficient but requires a pre-processing step. Moreover, the number of processors which can be used is often limited by the decomposition method. For these reasons we have preferred to distribute the rows of the FE matrix to the processors and to solve the global matrix. The assembling is performed by degrees of freedom (dof), so it has very good parallel performances [6]. On the other hand, the solving is CPU-expensive with bad parallel performances [2].

The objective of this paper is to describe how the solver has been modified in order to obtain better performances on a cluster of 10 DEC-ALPHA workstations linked by a FDDI ring and fitted with Parallel Virtual Machine software. The parallelization and the efficiency of the Conjugate Gradient (CG) algorithm strongly depend on the type of preconditioning. The Diagonal Preconditioning (DP) is first presented. The implementation of the Incomplete Cholesky Preconditioning (ICP) is then described and a new technique, named Block Incomplete Cholesky Preconditioning (BICP), is proposed. Parallel performances are analyzed on a matrix of 60000 dof. This is the largest problem which can be solved on only one processor with 64 MB RAM. Finally performances are analyzed from the modeling of a realistic device.

Manuscript received November 3, 1997.

C. Vollaire, vollair@trotek.ec-lyon.fr; L. Nicolas, laurent@trotek.ec-lyon.fr, http://cegely.ec-lyon.fr/.

II. Diagonal Preconditioning

Small-scale parallelism is used. No message passing is required for the preconditioning. However, the convergence rate is low. A matrix-vector multiplication is required to compute the residual vector at each iteration of the CG [2]. Only the lower part of the FE matrix is stored because it is symmetric. Each processor works of N/P lines, where N is the number of lines of the matrix and P is the number of processors. The vector to multiply is duplicated on all the processors. This multiplication is performed in parallel: each processor computes a partial residual vector (Fig. 1). The load balancing is nearly perfect because of the constant bandwidth of the matrix (Fig. 2).

Once the multiplication is performed, each processor *broadcasts* its partial residual vector to all the others. These partial results are added to obtain the final residual vector in Single Program Multi Data (SPMD) mode. This operation requires (P^2-P) messages passing per iteration. Another way

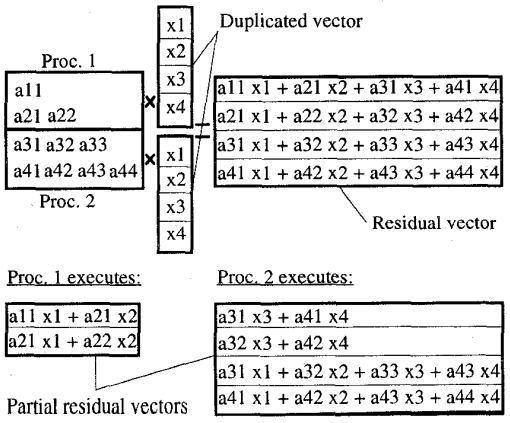

Fig. 1. Parallel multiplication matrix-vector - example of a 4 × 4 matrix (N=4) stored on 2 processors (P=2).

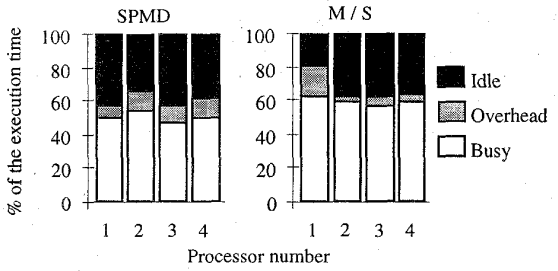

Fig. 2. Average of state of every processor for the CG with the DP (solving on 4 processors - 60000 degrees of freedom).

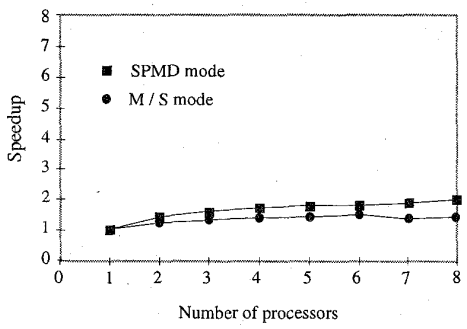

Fig. 3. Speedups of the CG with the DP for both methods of concatenation

to calculate the residual vector is the Master Slave (MS) mode: each partial residual vector is *sent* to one master processor (processor number 1 in Fig. 2). This one computes the final residue and *broadcasts* it to all the others.

The MS method minimizes the number of communications ((P-1)×2 messages / iterations) but introduces a simultaneous idle time when the slaves are waiting for the entire residual vector. Furthermore, the messages broadcasted by the master are larger (equal to the total number of lines of the matrix) than SPMD mode (equal to the number of lines stored on the considered processor). Therefore the SPMD mode is more efficient. On the other hand, because the convergence rate is low, the cost of communications is penalizing for both methods.

III. Incomplete Cholesky Preconditioning

Compared to the DP, the building of the incomplete Cholesky matrix and forward-backward substitutions are also required. The FE matrix A is factorized in two matrices: A = L×L^t. The incomplete Cholesky matrix L is built by column [7]. This algorithm is implicitly parallel because the Lij terms can be computed independently once the diagonal term Ljj has been computed. The knowledge of both lines i and j is required to compute the ij term. If both lines are not stored on the same processor, a message passing is necessary (Fig. 7). The memory space needed to achieve the ICP is three times larger than DP because L and L^t have the same structure with

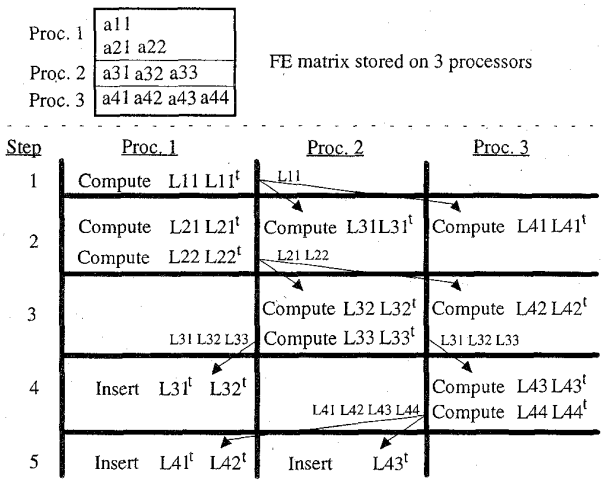

Fig. 4. Building of L and L^t in 5 parallel steps.

A. L^t is built to achieve a quick access by columns to the terms. Fig. 4 illustrates this strategy on a 4×4 matrix stored on 3 processors.

The knowledge of the storage by columns of L^t allows to reduce the CPU time needed for the back substitution. This algorithm is implicitly sequential: the first processor computes first his part of y and broadcasts it to all the others in SPMD mode. The second processor can then begin his computation and so on. This step is very penalizing in term of parallel performances, because it is performed at each iteration (Fig. 5).

From Table I it appears clearly that the ICP allows to reduce the number of iterations to solve the system of equations. However, because of the large amount of message passing required, it cannot be applied to large problems.

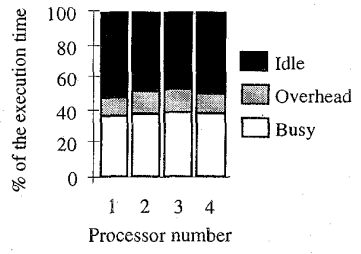

Fig. 5. Average of state of every processor for the CG with the ICP (solving on 4 processors - 60000 degrees of freedom).

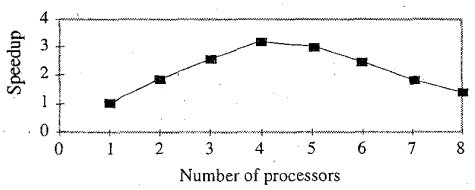

Fig. 6. Speedup for the CG with the ICP.

TABLE I
CPU TIME PER PROCESSOR AND NUMBER OF ITERATIONS FOR THE DP, THE ICP AND THE BICP – 60000 DEGREES OF FREEDOM

Number of processors	1		4		8	
	time (s)	iterations	time (s)	iterations	time (s)	iterations
DP	615	204	367	204	316	204
ICP	3193	59	901	59	1773	59
BICP	3193	59	353	61	234	79

IV. BLOCK INCOMPLETE CHOLESKY PRECONDITIONING

To avoid messages passing during the building of the incomplete Cholesky matrix, this one is assembled per block: only the terms stored on one processor are used to build the corresponding Cholesky submatrix. There is no passage of terms between the processors contrary to the classical ICP. Furthermore, on the contrary of the method proposed in [8], it does not require that every processor owns all rows and columns. So some parts of the matrix are not built at all (Fig. 7), and the terms effectively assembled are approximated because of the back dependency. With one processor, this method corresponds to the usual ICP.

As seen by the number of iterations (Table I), this scheme leads to a weakening of the preconditioning compared to the ICP. Furthermore, the number of iterations increases with the number of processors. The memory space is three times larger than the DP because of the matrices L and L^t. On the other hand, the preconditioning matrix is built without message passing. This leads to a better utilization of the processors (Fig. 8). The preconditioning matrix is then constituted by independent submatrices (Fig. 7). Each processor can compute independently his part of the result vector, so this step is entirely parallel. The concatenation of these partial results is performed by messages passing in SPMD mode.

The number of operations required to compute both preconditioning matrix and result vector depends on the

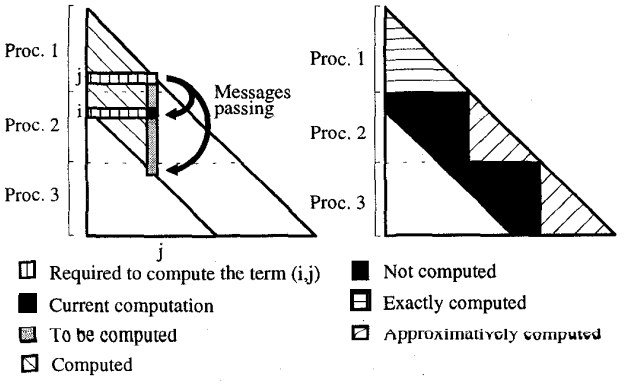

Fig.7. Cholesky matrix constructed per columns (right); block incomplete Cholesky matrix (left) - example with 3 processors.

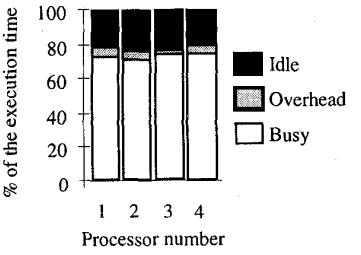

Fig.8. Average of state of every processor for the CG with the BICP (solving on 4 processors - 60000 degrees of freedom)

number of processors available. When this number increases, the building of the matrix L and the forward-backward substitutions are more efficient in CPU time because preconditioning submatrices become smaller. Then the number of operations per processor decreases. Actually, the weakening of the preconditioning is widely compensated by the decrease of the number of computations. On the other hand, the number of dof per processor has to be sufficiently large: if too low, the overhead becomes predominant (see the breakpoint at the 4 processors level in Fig. 9).

The parallel performances can be expressed in terms of super-linear speedup (Fig. 9): when more than 1 processor are used, the total CPU time for the solving becomes smaller than that of 1 processor. Two other problems (different geometries) in the 60000 dof range show the repeatability of these results (Fig. 9). Rigorously the speedup should be defined as the CPU ratio between the best sequential algorithm and the considered parallel algorithm. For this example, the DP is the best sequential algorithm, leading to a speedup on 8 processors equal to: 615÷234=2.6 (Table I). However, with complex geometries, the DP does not converge at all and this definition of speedup cannot be used.

V. COMPARISON BETWEEN THE METHODS ON LARGE PROBLEMS

The electromagnetic scattering of a plane wave by a perfect electric conducting airplane is presented as a large

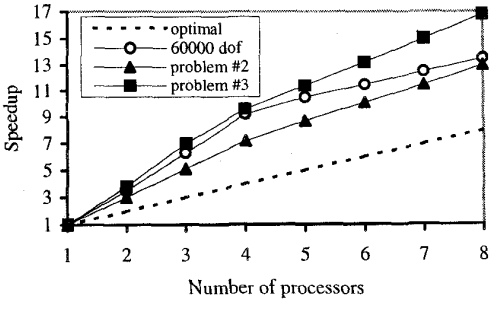

Fig. 9. Speedup for the CG with the BICP. Problems #2 and #3 are also in the 60000 dof range.

problem. The FE formulation is coupled to an Engquist-Majda absorbing boundary condition. It is written in terms of the vector field (**E** or **H**). The numerical discretization is performed with nodal elements, leading to 3 complex unknowns (6 dof) per node. The entire formulation is discussed in detail in [1]. The Cuthill-McKee renumbering algorithm reduces the band of the matrix. For such problems, the mesh size is related to the frequency of the incident plane wave (10 nodes per wavelength have to be used).

Only DP and BICP are compared (Table III and Table IV). Due to the large amount of message passing required, it was not possible to use ICP for such problems. Due to the number of dof, it was also not possible to compute the larger problem on 4 processors. For both examples, whatever the number of processors, the BICP is the most efficient. It allows to reduce the number of iterations by generating an good preconditioning. Furthermore, it does not introduce a very penalizing overhead such as the ICP method.

VI. Conclusions

We have presented in this paper several preconditioning methods for the Conjugate Gradient to solve large sparse matrices on a distributed memory computer fitted with a FDDI ring. Because the convergence rate is low, the cost of communications is penalizing when using the diagonal preconditioning. Small-scale parallelism is not adequate to distributed memory computer with this type of network. The incomplete Cholesky preconditioning allows to reduce the number of iterations but cannot be used on large problems because of the large amount of messages passing required. The new preconditioning method, named block incomplete Cholesky, appears to be a good compromise in term of generated preconditioning and CPU time.

Furthermore, such a developed code is immediately implementable on a CRAY T3E. This parallel distributed memory computer is a MIMD type too. Moreover, this machine is equipped with very high performances network and message passing library (SHMEM). This should allow the solver to have better performances. On this type of massively parallel architecture, no significant speed up can be obtained for a large number of processors because the number of dof per processor becomes too small when the number of processors increases.

Note that the algorithms presented in this paper are not specific to a HF electromagnetic formulation. They could be applied to any physical problem discretized with a FE method, with complex or non complex terms dispatched amount the memories of a distributed memory computer.

TABLE II
DESCRIPTIONS OF LARGE PROBLEMS

Frequency (GHz)	Number of tetrahedra	Number of nodes	Degrees of freedom
0.1	201556	33431	200586
0.3	308922	51183	307048

TABLE III
CPU TIME PER PROCESSOR AND NUMBER OF ITERATIONS FOR THE DP, AND THE BICP (200586 DEGREES OF FREEDOM)

Number of processors	4	
	time (s)	iterations
DP	25331	5009
BICP	19434	3231

TABLE IV
CPU TIME PER PROCESSOR AND NUMBER OF ITERATIONS FOR THE DP AND THE BICP (307048 DEGREES OF FREEDOM)

Number of processors	8	
	time (s)	iterations
DP	80280	7476
BICP	58950	4902

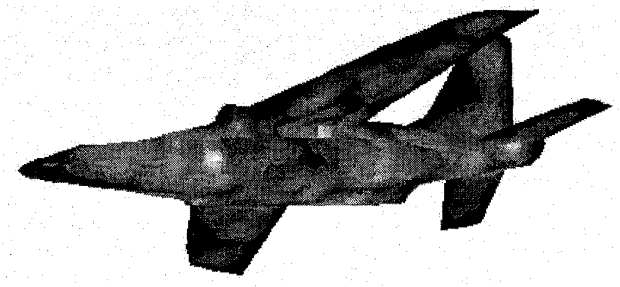

Fig. 10. Perfect electric conducting airplane illuminated by a plane wave (0.3 GHz) - magnitude of the magnetic field.

References

[1] L. Nicolas, K. A. Connor, S. J. Salon, B. G. Ruth, and L. F. Libelo, "Three dimensional FE analysis of high power microwave devices," *IEEE Trans. Mag.*, vol. 29, no 2, pp. 1642-1645, March 1993.

[2] C. Vollaire, L. Nicolas, and A. Nicolas, "Finite elements coupled with absorbing boundary conditions on parallel distributed memory computer," *IEEE Trans. on Mag*, vol. 33, no 2, pp. 1448-1451, March 1997.

[3] Y. Saad, "Krylov subspace method on supercomputers," *SIAM J. Sci. Stat. Comput.*, vol. 10, no 6, pp. 1200-1232, November 1989.

[4] K. Iwano, V. Cungoski, K. Keneda, H. Yamashita, "A parallel processing method in FE analysis using domain division," *IEEE Trans. on Mag.*, vol. 30, no 5, pp. 3598-3601, September 1994.

[5] R. Lee, V. Chupongstimm, "A portioning technique for Finite Element solution of electromagnetic scattering from electrically large dielectric cylinders," *IEEE Trans. on Ant. and Prop.*, vol. 42, no 3, pp. 737-741, May 1994.

[6] D. Zois, "Parallel processing techniques for FE analysis: stiffnesses, loads and stresses evaluation," *Comp. & St.*, vol. 34, no 32, pp. 353-374, 1990.

[7] H. Magnin and J.L. Coulomb, "A parallel and vectorial implementation of basic linear algebra subroutines in iterative solving of large sparse linear systems of equations," *IEEE Trans. on Mag.*, vol. 25, no 4, pp. 2895-2897, July 1989.

[8] M. L. Barton, "Three-dimensional magnetic field computation on a distributed memory parallel processor," *IEEE Trans. on Mag.*, vol. 26, no 2, pp. 834-836, March 1990.