

A COMPARISON OF PRECONDITIONING TECHNIQUES FOR PARALLELIZED PCG SOLVERS FOR THE CELL-CENTERED FINITE-DIFFERENCE PROBLEM

Richard L. Naff¹, John D. Wilson¹

¹*U.S. Geological Survey, MS 413, Box 25046, Denver, CO 80225 USA*

ABSTRACT

This paper reports on a parallelization of the preconditioned conjugate gradient algorithm for sparse, symmetric matrices. Parallelization is based in domain partitioning into non-overlapping subdomains; the resulting parallelized algorithm is briefly described. Comparisons are made between three block preconditioners commonly used in the parallelization of the preconditioned conjugate gradient methods: Jacobi, incomplete Choleski, and Gauss Seidel. Basic timing and iteration results for these preconditioners are presented; these results tentatively indicate that the simpler block Jacobi algorithm is as efficient as the more complex block incomplete Cholesky and block Gauss Seidel.

1. INTRODUCTION

As the cost of high-speed, silicon-based processors has decreased, a trend toward parallel computing in which processors are linked in either a shared-memory or distributed-memory configuration has evolved (*Foster, 1996; Saad and van der Vorst, 2000*). To take advantage of either of these configurations, one must enlist specialized software which enables a program to distribute tasks to processors. In this study, standard Message Passing Interface (MPI) (*Gropp et al., 1999*) functions are used to assign tasks to processors in a distributed-memory environment; this environment consists of a Linux-based cluster of dual CPU computers ranging in speed from 700 MHz to 3400 MHz. The computers are linked by means of a private Local Area Net (LAN) using full duplex transmission and Gigabit Ethernet switching, allowing transmission at 1000 Mbps.

The objective of this study is to produce a parallel solver for a symmetric sparse matrix similar to that produced by the cell-centered finite-difference (CCFD) algorithm. The linear system resulting from a CCFD problem can be denoted as

$$\Lambda\xi = \beta \tag{1}$$

where Λ is the stiffness matrix, β is the right-hand-side (RHS) vector and ξ is the unknown vector. Because a cluster of computers is envisioned as the primary computing tool, parallelization is coarse grain (*Foster, 1996*). The basis of parallelization is found in partitioning the matrix such that a subset of the linear equations contained in (1) is assigned to each processor. In the case of this study, this partitioning is equivalent to subdividing the computational domain of the CCFD problem into non-overlapping subdomains. The solution algorithm is based in the preconditioned conjugate gradient (PCG) algorithm in which block preconditioning is used. Parallelization of the PCG algorithm

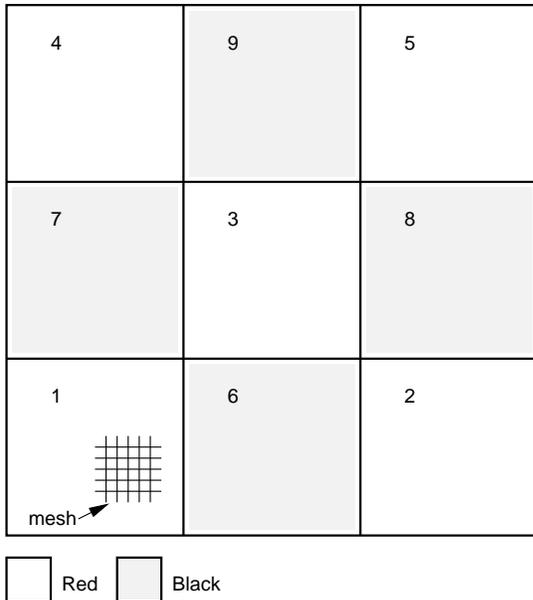


FIGURE 1. 3×3 red/black subdomain partitioning.

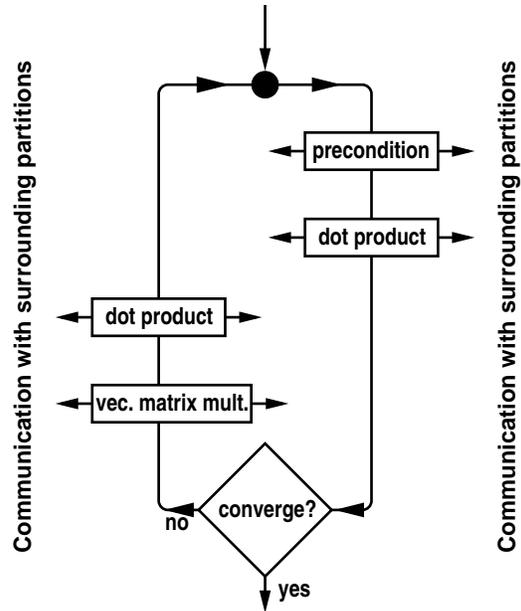


FIGURE 2. Schematic flow-chart for PCG algorithm.

requires that information be passed between processors at various steps; communication costs associated with the passing of this information adds to the work load of the algorithm and diminishes the efficiency obtained from parallelizing.

2. BASIC CONCEPTS

Basic to any parallelization of a matrix solver is a scheme by which work is proportioned to processors (*Foster, 1996*). In this paper, it is assumed that the matrix has correspondence to an underlying regular mesh that can be partitioned into non-overlapping subdomains. This partitioning becomes the basis by which the work of the matrix solver is proportioned among processes. The resulting mesh partitions, sharing corners and edges, are numbered using a simple red/black coloring scheme. Under these conditions, indirect addressing can be avoided in the reordered matrix.

As an example of these concepts, consider a two-dimensional domain which has been discretized for a CCFD problem; a possible partitioning of such a domain into nine subdomains is illustrated in Figure 1. The red/black numbering of the partitioning ensures that any given black partition adjoins only red partitions and *vice versa*. In general, the reordered linear system can be denoted as

$$\Omega x = b \tag{2}$$

where $\Omega = P\Lambda P^T$, $x = P\xi$, $b = P\beta$ and P is a permutation matrix accounting for the reordering of Λ in response to red/black partitioning. For the two-dimensional red/black

The vector-matrix multiply requires actual MPI sends and receives of the vector entries corresponding to entries in C_{ij} . However, as vector-matrix multiplication is not order dependent, this operation is also relatively efficient but does incur a communication cost. Depending on the preconditioner, communications costs associated with preconditioning the matrix can range from negligible to significant; these costs will be discussed in the following section.

3. BLOCK PRECONDITIONING

Three block preconditioners are applied to the red/black matrix partitioning used in this study: Jacobi, incomplete Choleski and Gauss Seidel. Of these, block Jacobi (BJ) (*Axelsson*, 1996) is conceptually the simplest; in matrix (3), the connector matrices C_{ij} are ignored and only the diagonal blocks A_{ii} are approximately inverted against b_i . That is, let $D_A = \text{diag}(A_{11}, \dots, A_{nn})$; then $x \approx D_A^{-1}b$ where D_A^{-1} is approximated with a smoothing operation. Incomplete Cholesky with zero fill (IC(0)) and with one fill (IC(1)) are used as smoothers for the diagonal blocks A_{ii} . With reference to (5), the BJ algorithm can be expressed simply as

- (1) SOLVE approximately: $Rx_u = b_u$
- (2) SOLVE approximately: $Bx_l = b_l$

Because the connector matrices C_{ij} are ignored, communications costs for this preconditioner are negligible.

The block incomplete Choleski (BIC) is based on a parallel IC(0) algorithm given by *Saad* (2003). This algorithm is an extension of the point-wise algorithm to the case of matrix partitioning based in non-overlapping subdomains. In reference to matrix (4), let $R \approx L_R U_R$ and $B \approx L_B U_B$ where $U_R = D_R L_R^T$, $U_B = D_B L_B^T$ and L_R , D_R , L_B , D_B are the IC(0) factorizations of R and B , respectively. The BIC algorithm can be expressed in the following form:

- (1) SOLVE by forward elimination: $L_R y_u = b_u$
- (2) FORM: $r_l = b_l - D_R^{-1} C^T y_u$
- (3) SOLVE by forward elimination: $L_B y_l = r_l$
- (4) SOLVE by back substitution: $U_B x_l = y_l$
- (5) FORM: $r_u = y_u - C x_l$
- (6) SOLVE by back substitution: $U_R x_u = r_u$

Note that operationally the SOLVE steps in this algorithm, without the FORM steps, are equivalent to the the BJ algorithm when IC(0) is applied. This procedure is possible because of the simplicity of the IC(0) preconditioner. As in the point-wise algorithm, it is necessary that the order of operation be maintained. In the FORM steps, sections of the arrays y_u and x_l must be transferred from surrounding partitions to form the $C^T y_u$ and $C x_l$ products, thus incurring communication costs. Note that, as the SOLVE steps contain block operations that are order independent, the elimination process for one matrix A_{ii} can occur concurrently with another. Thus, as soon as the transfer of information to a block to form an appropriate right-hand side has occurred, the elimination process can start.

The block Gauss-Seidel (BGS) algorithm is based in the Schur complement of (4):

$$\begin{bmatrix} R & C \\ C^T & B \end{bmatrix} = \begin{bmatrix} R & 0 \\ C^T & S \end{bmatrix} \begin{bmatrix} R^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} R & C \\ 0 & S \end{bmatrix} \quad (6)$$

where $S = B - C^T R^{-1} C$ is the Schur complement matrix. Neglecting the $C^T R^{-1} C$ term in S leads to the Gauss-Seidel approximation (*Golub and Van Loan, 1983*):

$$\begin{aligned} \begin{bmatrix} R & C \\ C^T & B \end{bmatrix} &\approx \begin{bmatrix} R & 0 \\ C^T & B \end{bmatrix} \begin{bmatrix} R^{-1} & 0 \\ 0 & B^{-1} \end{bmatrix} \begin{bmatrix} R & C \\ 0 & B \end{bmatrix} \\ &= \begin{bmatrix} R & 0 \\ C^T & B \end{bmatrix} \begin{bmatrix} I & R^{-1} C \\ 0 & I \end{bmatrix} \end{aligned} \quad (7)$$

When approximation (7) is applied to (5), the BGS algorithm results:

- (1) SOLVE approximately: $Ry_u = b_u$
- (2) FORM: $r_l = b_l - C^T y_u$
- (3) SOLVE approximately: $Bx_l = r_l$
- (4) FORM: $r_u = b_u - Cx_l$
- (5) SOLVE approximately: $Rx_u = r_u$

The approximate solves are again carried out with either IC(0) or IC(1) on the A_{ii} block matrices contained in R and B . Communication costs are associated with the algorithm FORM steps in which it is necessary to transfer y_u and x_l information between partitions; preconditioner communication costs should be approximately equivalent to those of the BIC algorithm. As compared to the BIC algorithm, it is apparent that one additional approximate inversion is required.

One measure of preconditioner superiority is the improvement in conditioning that it imparts to a matrix. That is, in the PCG solution of (2), the function of the preconditioner is to improve the condition number of Ω so as to decrease then number of iterations to convergence. One's impression of these block preconditioners is that, given IC(0) smoothing in all cases, BGS will provide better preconditioning than either BIC or BJ. That is, assuming that inclusion of the connector matrices C_{ij} increases the effectiveness of the preconditioning, then BGS and BIC should be superior to BJ in preconditioning performance. And as BGS contains one additional approximate matrix inversion than BIC, one would assume that BGS is superior to BIC in this regard. Thus, the BGS algorithm should take the fewest iterations to converge for any given problem, and BJ should take the most. On the other hand (and again assuming IC(0) smoothing in all cases), the BGS algorithm requires the most operations per iteration and also will be encumbered with communication costs associated with the connector matrices. The BIC algorithm also is encumbered by communication cost approximately equivalent to the BGS algorithm, but requires fewer operations per iteration. As the number of elements in the connector matrices increases with increased partitioning, communication costs for both the BIC and BGS algorithms also should increase. The BJ algorithm contains no communication costs relative to the connector matrices and requires slightly fewer operations per iteration than the BIC algorithm. If superior preconditioning assumed for the BGS and BIC algorithms do not sufficiently speed up convergence, then the BGS and BIC algorithms may be inferior to the simpler BJ algorithm as they may require greater execution times to achieve convergence.

4. SOLVER RESULTS

For testing purposes, a large sparse matrix was generated with characteristics similar to those produced by discretization of the ground-water flow equation. CCFD-like coefficients were generated for a regular mesh with cell dimensions of $160 \times 80 \times 40$; assembly of these coefficients produced the test matrix. The matrix contains both random and anisotropic characteristics as the equivalent of the cell conductivities were generated randomly and the resulting coefficients were multiplied by a constant anisotropic factor for the x , y and z directions. These factors vary by multiples of ten from x largest to z smallest. This sparse matrix was multiplied by a randomly generated unknown vector x to produce a RHS vector b . Domain partitioning was used to create subdomains; allowing the set (p, q, r) to represent the number of partitions perpendicular to the x , y and z directions, then the partitioning sequences used were $(2, 1, 1)$, $(2, 2, 1)$, $(2, 2, 2)$, $(4, 2, 2)$ and $(4, 4, 2)$. These partition sequences resulted in 2, 4, 8, 16 and 32 subdomains, with each subdomain corresponding to a separate processor.

The computer cluster used in this study is largely heterogeneous, which presents problems for timing tests. That is, the processors run at clock speeds that vary from 700 MHz to 3400 MHz, with the slowest twelve processors having clock speeds that vary from 700 MHz to 800 MHz. However, eight of the processors run at 1977 MHz; this homogeneous subset was used to run a series of solves for the three block preconditioners, but allowed only for a maximum of eight subdomains. The entire heterogeneous cluster was used when 32 processors were needed; this entailed using the slowest as well as the fastest processors. To ensure that the solutions be comparable, the slowest processors were included first in every partitioning sequence. Thus, for the heterogeneous runs, if the partitioning sequences required less than twelve processors, then no processor with a clock speed greater than 800 MHz was used. For partitioning sequences requiring more than twelve processors, then the faster processors were accessed as well.

The timing results presented are the elapsed time in the PCG solver and do not include any setup costs (partitioning of domain, etc.). As more subdomains are used, the communication costs for all preconditioner types increased. With increasing inter-process communication, the variability in timing results also increased. This increased variability in execution times is likely the result of how the processors interact within the cluster, which is somewhat random as the processors on different machines are not synchronized. To compensate for this variability, multiple runs for each test case were made; the reported execution time is the average from these runs. Thus, the timing results presented for two subdomains is the average of two runs, for four subdomains is the average of three runs, for eight subdomains is the average of four runs, for 16 subdomains is the average of five runs, and for 32 subdomains is the average of six runs. For the single partition $(1, 1, 1)$, a single run was used.

Figure 3 is a presentation of the number of iterations required for the various combinations of block preconditioner and smoother. The BGS preconditioners present the best performance in that the number of iterations for convergence increase only slightly with increased partitioning. The BJ preconditioners, on the other hand, exhibited a rather stark decrease in preconditioning performance with increased partitioning. In the case of the IC(1) smoother, the number of iterations required increases by more than 50% as the

number of subdomains increases from 1 to 32; similar results have been reported by *Lo and Saad* (1996) for block Jacobi preconditioning with overlapping subdomains. Because the C_{ij} connectors are ignored with BJ, and as the number of elements in these connectors increases with increased partitioning, it was expected that BJ preconditioning would degrade with increased partitioning. The BIC algorithm is a bit of a disappointment in this regard. While the two-subdomain result is similar to a single partition with regard to iterations required, there is a 25% increase in iterations required in going to four subdomains. This increase likely indicates a sensitivity of the incomplete Cholesky algorithm to the test matrix anisotropy. That is, while the unpartitioned test matrix requires 106 iterations to converge, the renumbered, four-subdomain partitioned matrix requires 131 iterations to converge. If the test were isotropic, the number of iterations would still likely increase, but the renumbering effect would probably not be so dramatic as depicted here. As compared to BIC, BJ may also exhibit sensitivity to anisotropy, but principally when partitioning is increased perpendicular to the x direction. That is, for both the (2, 1, 1) and (4, 2, 2) partitioning sequences, BJ for both IC(0) and IC(1) shows a marked increase in iterations required. As the matrix coefficients are larger in the x direction than either the y or z directions, it is not surprising that losing this information results in decreased preconditioning.

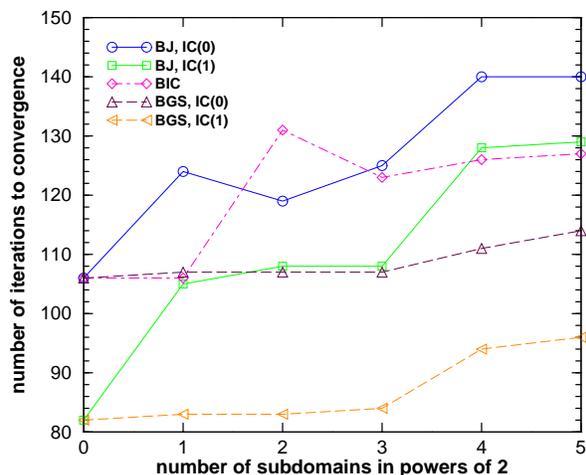


FIGURE 3. Iteration results for subdomain partitioning

follow from the iteration results of Figure 3. In general, these results suggest that the simpler BJ algorithm is superior to the BGS algorithm; the superior preconditioning provided by the BGS algorithm does not compensate for the work required to produce this level of preconditioning. For eight-processor partitioning, BIC requires almost as many iterations to converge as BJ with IC(0). Thus, considering the increased communication cost associated with BIC, it is not surprising that simple BJ requires less execution time than BIC for this case. That BIC is superior to BGS in execution time for two-processor partitioning follows from Figure 3 where both BIC and BGS with IC(0) use nearly the same number of iterations to converge. BGS with IC(0) requires more work per iteration than BIC and thus a greater execution time, in this case, is expected. For partitioning

The homogeneous timing series consists of the partitioning sequences (1, 1, 1), (2, 1, 1), (2, 2, 1) and (2, 2, 2) producing 1, 2, 4 and 8 subdomains respectively; results for this series are presented in Figure 4. First, one anomaly is noted: for a single partition, all the IC(0) results should cluster; they do not. For reasons yet to be determined, the BIC results consistently produced timing results more equivalent to IC(1) preconditioning rather than IC(0). Generally speaking, the block preconditioners using IC(1) as a smoother produce slightly superior execution times to the IC(0) smoother. The IC(1) smoother does require more work per iteration than the IC(0), so this result does not immediately

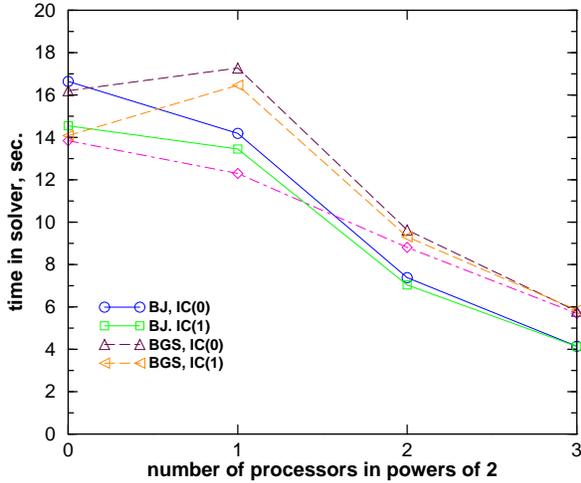


FIGURE 4. Timing results for homogeneous processors.

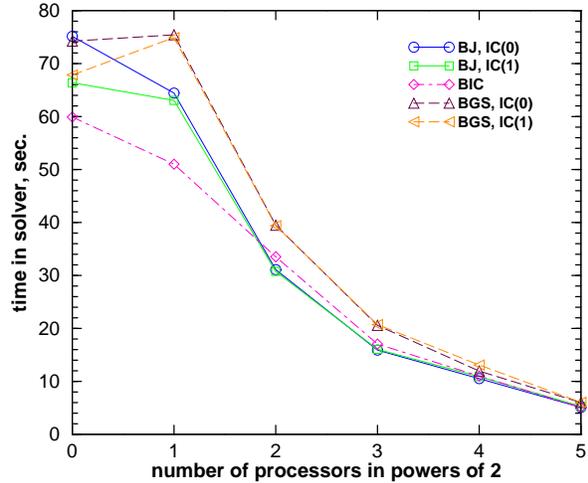


FIGURE 5. Timing results for heterogeneous processors.

requiring more than two processors, BIC uses significantly more iterations to achieve convergence than BGS with IC(0), and the speed advantage associated with the simpler BIC algorithm is lost.

Timing results for the heterogeneous cluster solutions, using up to 32 processors, are shown in Figure 5. The heterogeneous timing series consists of the partitioning sequences (1, 1, 1), (2, 1, 1), (2, 2, 1), (2, 2, 2), (4, 2, 2) and (4, 4, 2) producing 1, 2, 4, 8, 16 and 32 subdomains respectively. These results generally show a trend, similar to that for the homogeneous-cluster results, of decreasing execution times with increased processor usage. Again, the single-partition result for BIC tends to behave more as IC(1) results rather than IC(0) results. From this figure, it is apparent that the IC(0) and IC(1) smoothers generally exhibited only very small timing differences for BGS and BJ preconditioners. Also, the BIC timing results here behave rather more like the BJ results, as opposed to the homogeneous cluster results where they tend toward the BGS results. The heterogeneous cluster results also suggest that preconditioning with BGS is inferior to either BIC or BJ.

Because of the initial cost of setting up the parallel problem, the first doubling from one to two processors is never as effective as subsequent doublings in reducing execution time of the solver. For BJ with IC(0), if the first doubling is neglected, then doubling of the number of processors reduces the solver time by a factor ranging from 0.48 to 0.66, with an average value of 0.54. Again neglecting the first doubling, for BGS with IC(0) the solver time with each doubling is reduced by a factor ranging from 0.50 to 0.60, with an average value of 0.55. Similarly, BIC reduces the solver time by a factor ranging from 0.46 to 0.72, with an average value of 0.61. These scalings generally suggest that BJ, for coarse-grain parallelization, is superior BIC but about equivalent, in this respect, to BGS. Reduction factors less than or equal to 0.50 are frequently associated with the 32-processor heterogeneous-cluster results. For the heterogeneous-cluster runs, faster processors were accessed when 16 and 32 processor were called for. Thus, these timing reductions may be biased by the heterogeneous nature of the cluster.

5. CONCLUDING REMARKS

The initial conclusion of this study is that, with regard to obtaining superior execution times, the superior preconditioning obtained with either BGS or BIC is not particularly advantageous. This conclusion is preliminary in that only one matrix type is examined; other matrix types (nonrandom, homogeneous) should also be examined. A secondary conclusion is that the matrix partitioning, in conjunction with the matrix type, can influence execution times. With certain partitioning, increased number of processors lead to increased iterations to convergence under BIC and BJ, demonstrating the influence of anisotropy on these schemes.

REFERENCES

- Axelsson, O. (1996), *Iterative Solution Methods*, Cambridge University Press, Cambridge, UK.
- Foster, I. (1996), *Designing and Building Parallel Programs*, Addison-Wesley, Reading, Massachusetts.
- Golub, G., and C. Van Loan (1983), *Matrix Computations*, John Hopkins University Press, Baltimore, USA.
- Gropp, W., E. Lusk, and A. Skjellum (1999), *Using MPI: portable parallel programming with message-passing interface*, second ed., MIT Press, Cambridge, Massachusetts.
- Lo, G. C., and Y. Saad (1996), Iterative solution of general sparse linear systems on clusters of workstations, *Tech. report UMSI 96/117*, Minnesota Supercomputing Institute, Minneapolis, MN.
- Saad, Y. (2003), *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics, Philadelphia, USA.
- Saad, Y., and H. van der Vorst (2000), Iterative solution of linear systems in the 20th century, *Journal of Computational and Applied Mathematics*, 123, 1–33.