Parallel iterative linear solvers for oil reservoir models

by Ilan Efrat
Miron Tismenetsky

This paper suggests a new algorithm for solving sparse linear systems which is readily parallelized and is very efficient for matrices arising in such domains as reservoir modeling. The algorithm is, in fact, a variant of the incomplete block factorization technique, accelerated by Biconjugate Gradient iterations or by another acceleration method. Implementation on a vector computer such as the IBM 3090 Vector Facility is described. We also suggest some refinements of known preconditioning methods enabling their parallel computation. Numerical experiments are presented to display the performance of the suggested methods. Special attention is given to fully implicit, multiphase models which yield asymmetrical systems.

1. Introduction

This paper presents an algorithm for an efficient, easily vectorized, parallel iterative solution of sparse linear systems. Because it is a block method, it involves operations between long subvectors and thus may be performed efficiently on a vector computer such as the IBM 3090 Vector Facility. The algorithm is in fact a variant of incomplete block factorization accelerated by the Biconjugate Gradients method [1] (or some other acceleration method). Typical linear systems occurring, for example, in oil reservoir simulations may consist of 10^5 or more unknowns. Because the solution of these unknowns consumes vast computational resources, efficient linear solvers which are readily parallelized are of great importance. Although modern direct solvers are fast and robust, there are many large systems occurring in oil reservoir simulations and elsewhere, especially in three-dimensional models, where iterative methods are substantially superior.

We present numerical experiments which show the attractive performance of our algorithm as compared with popular iterative solvers. We show how to perform the method on parallel or vector computers and give complexity estimates.

The computational bottleneck in parallel implementation of iterative methods is matrix preconditioning. Popular conditioning techniques are based on a pointwise LU decomposition and thus are sequential in their nature. We give several techniques to aid in parallelizing.

Some methods are suggested for the implementation of preconditioned iterative methods on computers with low-level parallelism (4–8 processors). This type of computing is possible using the recently announced FORTRAN Multitasking Facility on IBM multiprocessor mainframes.

2. Reservoir model equations

This section gives a brief survey of the governing oil reservoir model equations, which are detailed in [2].

• *Single-phase flow*

Flow of a single fluid within a porous medium is described by a second-order nonlinear differential equation

$$\nabla \cdot (T\rho \nabla P) = \frac{\partial(\phi\rho)}{\partial t}, \tag{1}$$

where P = fluid pressure, ρ = density (function of P), T = transmissibility (function of rock type and fluid viscosity), and ϕ = rock porosity. The reservoir is assumed to be surrounded by a nonpermeable rock. Hence boundary conditions are of the no-flow type (Neumann boundary conditions). After time discretization of the right-hand side of the equation, the left-hand side is linearized, usually by using Newton iteration. The resulting linear partial differential equations are then discretized. In the incompressible case this may be done in a way that yields a symmetric linear system.

• *Multiphase flow*

Most reservoir models include at least two major fluids, normally oil and water. In a multiphase model Eq. (1) is replicated:

$$\nabla \cdot (T_i\rho_i \nabla P_i) = \frac{\partial(\phi\rho_i S_i)}{\partial t}. \tag{2}$$

The new variable S_i is the relative mass concentration of phase i ($\sum S_i = 1$). The variable T_i is factored as $T \cdot t_i$, where T is a property of the rock (a function of position) and t_i is a nondecreasing function of S_i. The pressure difference $C_{ij} = P_i - P_j$ is called capillary pressure and is usually considered a function of S_i and S_j.

For a two-phase flow the following constraints result:

$$T_i = T \cdot t_i(S_i),$$

$$S_1 + S_2 = 1,$$

$$P_2 = P_1 + C(S_2).$$

Combining these constraints with Eq. (2) to eliminate S_2 and P_2, we obtain

$$\nabla \cdot (Tt_1\rho_1 \nabla P_1) = \frac{\partial(\phi\rho_1 S_1)}{\partial t},$$

$$\nabla \cdot \{Tt_2\rho_2 \nabla[P_1 + C(S_1)]\} = -\frac{\partial(\phi\rho_2 S_1)}{\partial t}. \tag{3}$$

The first equation is second-order self-adjoint for a fixed S_1, and first-order for a fixed P_1. The second equation is first-order for a fixed S_1. Its nature for a fixed P_1 is clarified when we rewrite it as follows (assuming, for simplicity, constant density):

$$\left[\left(t_2\nabla T + T\frac{\partial t_2}{\partial S_1}\nabla S_1\right) \cdot \left(\nabla P_1 + \frac{\partial C}{\partial S_1}\nabla S_1\right)\right] + Tt_2\left[\nabla^2 P + \frac{\partial C}{\partial S_1}\nabla^2 S_1 + \frac{\partial^2 C}{\partial S_1^2}(\nabla S_1)^2\right] = -\frac{\partial(\phi S_1)}{\partial t}. \tag{4}$$

This is a convection-diffusion equation. The diffusion coefficient is proportional to $\partial C/\partial S_1$.

Numerical solution for the two-phase model is harder than for the single-phase model because

1. The resulting linear system is twice as big, and since the complexity of its solution is greater, its cost increases considerably.
2. The system is no longer self-adjoint, and the matrix produced by a Finite Difference discretization is usually nonsymmetric, a serious drawback for most linear system solvers.

In many simulations where convection effects dominate the diffusion effect, it is possible to solve implicitly for P, and then explicitly (without matrix inversion) for S. This procedure is called IMPES (IMplicit Pressure, Explicit Saturation). IMPES imposes a stability limit on the time-integration step and hence may require more iterations than the fully implicit approach.

3. Preconditioned Biconjugate Gradients

Iterative methods are most effective when the matrix is symmetric positive definite (SPD). Unfortunately, many of the linear systems in multiphase reservoir models are asymmetric and have some complex eigenvalues. The matrix asymmetry grows when convection effects are strong relative to diffusion effects. For our numerical experiments with parallel preconditioning algorithms, we chose the Preconditioned Biconjugate Gradients (PBCG) method [1]. The choice was motivated by the following:

1. Unlike some other nonsymmetric generalizations of the Conjugate Gradients (CG) method, BCG reduces to CG if the matrix is SPD.
2. No estimation of the spectrum of the matrix A is needed for BCG.
3. Our experience is that for linear systems appearing in two-phase oil reservoir models, PBCG converges almost as fast as PCG for the corresponding single-phase problem, even when convection effects are dominant.

To present the PBCG algorithm, let x_0 be an initial guess for the solution of $Ax = b$, and K an easily invertible approximation to A. Define $r_0 = \bar{r}_0 = b - Ax_0$; $p_0 = K^{-1}r_0$; $\bar{p}_0 = (K^{\mathrm{T}})^{-1}\bar{r}_0$. PBCG then computes iteratively for $k = 1, 2, \cdots$,

$$\alpha_k = (K^{-1}r_{k-1}, \bar{r}_{k-1})/(\bar{p}_{k-1}, Ap_{k-1}),$$

$$x_k = x_{k-1} + \alpha_k p_{k-1},$$

$$r_k = r_{k-1} - \alpha_k A p_{k-1},$$

$$\bar{r}_k = \bar{r}_{k-1} - \alpha_k A^{\mathrm{T}}\bar{p}_{k-1},$$

$$\beta_k = (K^{-1}r_k, \bar{r}_k)/(K^{-1}r_{k-1}, \bar{r}_{k-1}),$$

$$p_k = K^{-1}r_k + \beta_k p_{k-1},$$

$$\bar{p}_k = (K^{\mathrm{T}})^{-1}\bar{r}_k + \beta_k \bar{p}_{k-1}.$$

Premultiplying A by K^{-1} is sometimes called preconditioning. Recent efficient preconditioning techniques are responsible for dramatic improvements in the efficiency of iterative methods. It is well known that iterative solution techniques are sensitive to the matrix spectral condition number $\kappa(A)$. For a given reservoir model, $\kappa(A)$ grows as h^{-2} when the mesh spacing h is refined. Note that drastic variations in rock transmissibility may increase $\kappa(A)$ by several orders of magnitude. The role of the conditioning operation is to reduce the spectral condition number of the resulting matrix so that $\kappa(K^{-1}A) \ll \kappa(A)$.

In the following sections we outline several preconditioning techniques and ways to parallelize them. Although presented in the context of PBCG, these methods are suitable for other preconditioned iterative methods as well.

• *Modified Block Incomplete Decomposition (MBID) in two dimensions*

In this section we consider a numerical solution of large sparse algebraic systems

$$Ax = b, \qquad A \in R^{N\times N}, \qquad b \in R^N, \tag{5}$$

with the matrix A partitioned into block form. These blocks correspond to partitioning of the state vector along lines of the Finite Difference grid. For instance, in the case of a five-point discretization scheme over an $n \times n$ square, all the blocks are $n \times n$, and they form a block tridiagonal matrix.

In the L-phase model the variables are usually grouped into nodes with L phase parameters per node. This grouping results in a matrix $A = \{A_{ij}\}$, where each A_{ij} is itself a block matrix consisting of $L \times L$ blocks.

For a numerical solution of (5) which is simultaneously efficient and vectorized, we suggest a new variant of an incomplete block LU decomposition of A followed by application of the BCG method. Note that the matrix A is assumed to be real but not necessarily symmetric.

To explain our approach, let A be first a block tridiagonal matrix with block entries A_{ij} (A_{ij} are sparse matrices and $A_{ij} = 0$ for $|i-j| > 1$). If all leading principal block submatrices of A are nonsingular, then [3] there exists the decomposition

$$A = LU,$$

where the upper and lower block matrices $U = [U_{ij}]$ and $L = [L_{ij}]$ are defined as follows:

$$U_1 = A_{11}; \qquad U_j = A_{jj} - A_{j,j-1}U_{j-1}^{-1}A_{j-1,j}$$

(where, for brevity, U_{jj} is denoted by U_j),

$$L_{jj} = I_j, \qquad L_{j+1,j} = A_{j+1,j}U_j^{-1},$$

$$U_{j-1,j} = A_{j-1,j},$$

and I_j stands for the appropriate identity matrix. This leads to a well-known algorithm for solving (5) which is equivalent to the block Gaussian method. In presenting the algorithm, we assume that A_{jj} are square matrices of order n_j, $\sum_{j=1}^{m} n_j = N$, and that the vectors x and b are partitioned conformably with the partition of A:

$$x = [x_1^{\mathrm{T}}, x_2^{\mathrm{T}}, \cdots, x_m^{\mathrm{T}}]^{\mathrm{T}}, \qquad b = [b_1^{\mathrm{T}}, b_2^{\mathrm{T}}, \cdots, b_m^{\mathrm{T}}]^{\mathrm{T}}.$$

Set $U_1 = A_{11}$, $y_1 = b_1$, and compute for $j = 2, 3, \cdots, m$

$$U_j = A_{jj} - A_{j,j-1}U_{j-1}^{-1}A_{j-1,j}, \tag{6}$$

$$y_j = b_j - A_{j,j-1}U_{j-1}^{-1}y_{j-1}. \tag{7}$$

Then set $x_m = U_m^{-1}y_m$, and compute for $j = m-1, m-2, \cdots, 1$

$$x_j = U_j^{-1}(y_j - A_{j,j+1}x_{j+1}). \tag{8}$$

In the applications under consideration the matrices A_{ij} are sparse and banded. Nevertheless the matrices U_j gradually become full. The resulting computational complexity in the case of $n_j = n$ for all j is $O(mn^2)$.

To reduce the complexity caused primarily by the necessity of inverting U_{j-1} in (6), we combine several ideas which modify the algorithm into a preconditioner.

1. The matrices U_{j-1}^{-1} in (6) are replaced by sparse approximate inverses. This can be done in several ways [4–6]. Here we use an approximate inverse which is simple, applicable to most problems arising in reservoir modeling, and, moreover, trivially vectorized.

   Let $U = D(I - C)$, where D denotes the diagonal of U. If $\|C\| < 1$, it is reasonable to approximate U^{-1} by $(I + C + C^2 + \cdots + C^k)D^{-1}$. Numerical tests have verified that the approximation with $k = 1$ is quite satisfactory in many reservoir modeling problems. Thus

   $$U^{-1} \simeq (I + C)D^{-1} = (2I - D^{-1}U)D^{-1}. \tag{9}$$

   Johnson et al. [7] suggests a refinement of the idea of Dubois et al. [8] to exploit the truncated Neumann series as an approximate inverse of a matrix. Specifically, they select scalars $z_0, z_1, \cdots, z_k$ to minimize the condition number of the matrix

   $$(z_0 I + z_1 C + \cdots + z_k C^k)(I - C),$$

   provided that the lower and upper limits of the spectrum of C are known. Our experience shows that the matrix C arising in reservoir simulations is usually strongly contracting, and therefore no substantial gain results from this attempted optimization.

2. Let the nonzero submatrices A_{jj} be banded. If $A_{j,j-1}$ and $A_{j-1,j}$ fail to be diagonal, the bandwidth of the matrices U_j in (6) increases to n_j, and despite (9) the complexity of the algorithm remains high. To avoid the fill-in of U_j it is

hence suggested that the bandwidth of U_j be kept constant, say, p, where p is the maximal bandwidth of the diagonal blocks A_{jj}. Thus we replace (6) with the approximation

$$U_j \simeq \{A_{jj} - A_{j,j-1}U_{j-1}^{-1}A_{j-1,j}\}^{(p)} \tag{6'}$$

where $\{\cdot\}^{(p)}$ denotes the p-banded matrix obtained from (6) by replacing all elements outside the p-band with zeros. This deletion of diagonals is motivated by the fast decay of their entries away from the main diagonal, as occurs in most problems of oil reservoir simulation (see [9]). Clearly, the bandwidth of U_j should be kept larger if the matrices $A_{j,j-1}$ and $A_{j-1,j}$ differ significantly from diagonal matrices.

3. At this stage some compensation for the error which occurs in replacing (6) with (6′) is required. Specifically, we require that the approximation of A have the same row sums as A (following Gustafsson's "row-sum agreement" [10]). To this end, it suffices to make the row sums of the (block diagonal!) defect matrix $E = A - LU$ equal zero.

Combining all three ideas above (approximate inverse, "cutting" the additional diagonals, and the row-sum agreement), we obtain the following preconditioning algorithm MBID (A_{ij} are assumed, for simplicity, to be of order $n = \sqrt{N}$):

Set $\hat{U}_1 = A_{11}$. Compute for $j = 2, 3, \cdots, m$

$$U_j = A_{jj} - A_{j,j-1}(2I - D_{j-1}^{-1}\hat{U}_{j-1})D_{j-1}^{-1}A_{j-1,j}, \qquad I = I_n,$$

$$\hat{U}_j = U_j^{(p)},$$

$$[d_1\ d_2\ \cdots\ d_p] = (A_{jj} - A_{j,j-1}\hat{U}_{j-1}^{-1}A_{j-1,j} - \hat{U}_j)e,$$

$$e = [1\ 1\ \cdots\ 1]^T,$$

$$R_j = \text{diag}\,[d_1, d_2, \cdots, d_p],$$

$$\hat{U}_j = \hat{U}_j + R_j.$$

The preconditioning of $Ax = b$ is then computed as follows:

Set $y_1 = b_1$; for $j = 2, 3, \cdots, m$ compute

$$y_j = b_j - A_{j,j-1}\hat{U}_{j-1}^{-1}y_{j-1}.$$

Set $x_m = \hat{U}_m^{-1}y_m$; for $j = m-1, m-2, \cdots, 1$ compute

$$x_j = \hat{U}_j^{-1}(y_j - A_{j,j+1}y_{j+1}).$$

For the commonly analyzed model problems of the Poisson operator with Dirichlet or Neumann boundary conditions, discretized on an m by n rectangle, we were able to prove that the condition number of the preconditioned matrix is $O(m)$ [11]. This fact does not hold for block incomplete factorization methods which do not apply the row-sum correction.

The latter result is an extension of Gustafsson's theorems [10] related to pointwise incomplete decomposition. It is evident that the block method suggested here is superior for rectangular problems where $m < n$. In this case the condition number of the matrix preconditioned by the pointwise method is $O(n)$, which is worse than the $O(m)$ that holds for MBID.

The computational tests presented in Section 5 indicate that the validity of MBID encompasses a much larger class of matrices than the class of M matrices. Our experience suggests that the asymptotic results hold for many two-phase models, with varying transmissibility, even such that their matrices are nonsymmetric. It is certainly desirable to discover the correct limits of the class of matrices for which MBID reduces the condition number of $O(m^2)$ to $O(m)$.

• *MBID for three-dimensional problems*

The Finite Difference discretization of a differential operator on a Cartesian k by m by n grid results in a matrix which may be viewed as block tridiagonal or block pentadiagonal. The first alternative corresponds to a partition of the state vector by planes, the second to a partition by lines. It turns out that an efficient and vectorizable extension of the MBID algorithm to three-dimensional problems is only possible if we choose the block pentadiagonal organization.

In presenting the algorithm we adopt the following notation: The vector of row sums of a matrix B is denoted by RS(B). This quantity is computed as in the two-dimensional case. The diagonal matrix, the diagonal elements of which are the corresponding elements of the vector b, is called diag [b]. Also, an approximate inverse of a matrix B which is of the form (9) is denoted by INV(B).

Consider a block pentadiagonal matrix of order N by N, $N = mn$, with k by k blocks,

$$A = \begin{bmatrix}
A_{11} & A_{12} & 0 & \cdot & \cdot & \cdot & 0 & A_{1,m+1} & 0 & \cdot & \cdot & 0 \\
A_{21} & A_{22} & A_{23} & 0 & \cdot & \cdot & \cdot & 0 & \cdot & & & \\
\cdot & \cdot & & & & & & & & \cdot & & \cdot \\
\cdot & & & & & & & & & & \cdot & \\
\cdot & & & & & & & & & & & A_{N-m,N} \\
A_{m+1,1} & 0 & & & & & & & & & & 0 \\
0 & \cdot & & & & & & & & & & \cdot \\
0 & 0 & & & & & & & & & & \cdot \\
\cdot & & & & & & & & & & & \cdot \\
\cdot & & 0 & \cdot & 0 & & & & & & \cdot & A_{N-1,N} \\
0 & \cdot & \cdot & 0 & A_{N,N-m} & 0 & & & & 0 & \cdot & A_{N,N}
\end{bmatrix}, \tag{10}$$

arising from a seven-point Finite Difference discretization of the operator mentioned above. Now consider the equation

$$Ax = b. \tag{11}$$

Partitioning the vectors x and b in accordance with the block organization of A, we have the following preconditioning algorithm for (11):

1. For $i = 1$ to N,

   $U_i = A_{ii}$,

   $R_i = \mathrm{RS}(A_{ii})$.

2. For $i = 1$ to $N - 1$,

   $U_{i+1} = U_{i+1} - A_{i+1,i}\,\mathrm{INV}(U_i)A_{i,i+1}$,

   $R_{i+1} = R_{i+1} - \mathrm{RS}(A_{i+1,i}U_i^{-1}A_{i,i+1})$;

   if $i + m \le N$, then

   $U_{i+m} = U_{i+m} - A_{i+m,i}\,\mathrm{INV}(U_i)A_{i,i+m}$,

   $R_{i+1} = R_{i+1} - \mathrm{RS}(A_{i+1,i}U_i^{-1}A_{i,i+m})$,

   $R_{i+m} = R_{i+m} - \mathrm{RS}[A_{i+m,i}U_i^{-1}(A_{i,i+1} + A_{i,i+m})]$,

   $U_{i+1} = U_{i+1} - \mathrm{RS}(U_{i+1}) + \mathrm{diag}[R_{i+1}]$.

3. Set $y_1 = b_1$.

   For $i = 2$ to N,

   $y_i = y_i - A_{i,i-1}U_{i-1}^{-1}y_{i-1}$;

   if $i > m$, then $y_i = b_i - A_{i,i-m}U_{i-m}^{-1}y_{i-m}$.

4. Set $x_N = U_N^{-1}y_N$.

   For $i = N - 1$ to 1,

   $z_i = y_i - A_{i,i+1}x_{i+1}$;

   if $i + m \le N$, then $z_i = z_i - A_{i,i+m}x_{i+m}$,

   $x_i = U_i^{-1}z_i$.

During the preparation of this manuscript, the work by Concus et al. [4] was brought to our attention. One of the algorithm variants suggested there is similar to MBID. Extensive tests presented there confirm the importance that we attribute to the row-sum agreement correction. The work by Concus et al. does not refer to the vectorizability potential of block methods. It is limited to two-dimensional problems and does not present analytical results about the resulting condition number.

4. Parallel execution of iterative methods

The computation of preconditioned iterative methods comprises matrix-vector multiplications and the preconditioning operation. In this section we present, for the convenience of the reader, two known methods for the parallel computation of matrix-vector products and the solution of tridiagonal linear systems. We also suggest some methods for an efficient parallel preconditioning on computers with a limited number of processors.

• *Parallel matrix vector product*

It is pointed out in [12] that if the nonzero pattern of the matrix is several diagonals, then its product by a vector may be performed very effectively on vector or pipeline processors.

To outline the method, assume $A \in R^{N\times N}$ to be a matrix which is nonzero only on m diagonals. Let V be a displacement vector such that its components $V_k = j - i$ for the elements a_{ij} on the kth diagonal. This matrix may be stored efficiently in a rectangular array C of dimensions $N \times m$. $C(I, K) = A_{i,i+v_k}$.

The product $Ax = y$ is then implemented in FORTRAN as

```
      DO 100 K=1,M
        BOT=MAX(1,1-V(K))
        TOP=MIN(N,N-V(K))
        DO 100 I=BOT,TOP
          Y(I)=Y(I)+C(I,K)*X(I+V(K))
100   CONTINUE
```

The inner loop is performed very efficiently on parallel or pipeline processors, since it references elements which are stored contiguously in memory. Similar rearrangement of the operations can implement sparse matrix-matrix products.

In some applications where sparse linear systems result from a discretization of differential equations, it is unreasonable to require diagonal ordering. If the matrix nonzero pattern is not along several diagonals, then a different packed storage scheme must be used. For instance, it may be stored in three vectors: I, J, V, where $V(k)$ is the numerical value of the $[I(k), J(k)]$th element. Such organization is hard to parallelize on pipeline or synchronously parallel processors. On asynchronous machines, however, it is easy to divide I, J, V into several sections and let each processor multiply a section. If the division is done along rows, then no memory locking is necessary, since each processor writes its own part of the output vector.

• *Matrix preconditioning*

The LU decomposition phase of iterative algorithms is harder to perform in parallel, because it is inherently recursive. Several methods for parallel preconditioning are presented.

Parallel MBID

The block nature of the MBID algorithm allows one to perform most of the computations as block matrix-vector or matrix-matrix products. These operations can be parallelized by a straightforward application of the methods given in the section on the parallel matrix-vector product. The one exception is the solution of the equation $\hat{U}_j x_j = y_j$, where $\hat{U}_j$ is a p-banded matrix.

Feilmeyer and Hatzopoulous [13] give some of the parallel tridiagonal solvers which were introduced during the last two decades and have been tailored for various vectors and

parallel architectures. For completeness we outline the Cyclic Reduction process [14], which is very competitive for vector computers.

Let $Ux = f$ be a tridiagonal linear system and $U = D(I - C)$, where D is the diagonal part of U, and $C_{ij} = 0$ for $|i - j| \neq 1$. The system may be rewritten as follows:

$$x = Cx + D^{-1}f = C^2x + CD^{-1}f + D^{-1}f. \quad (12)$$

Observe that $(C^2)_{ij} \neq 0$ only for $|i - j| \in \{0, 2\}$. Consequently, the odd elements in the system,

$$(I - C^2)x = CD^{-1}f + D^{-1}f, \quad (13)$$

are decoupled from the even ones. The even rows and columns may hence be deleted to produce a reduced tridiagonal system. This reduction process can be repeated recursively. After the odd elements are found, the even ones are computed explicitly by (12).

The cyclic reduction process involves $\log_2 n$ stages and is stable for strongly diagonally dominant matrices. If the number of diagonals $p > 3$, then U should be viewed as a block tridiagonal matrix, and a block tridiagonal solver may be applied.

We conclude this section with a parallel-operations count of the MBID method. Parallel operations are defined in terms of a computational model with p concurrent processors, each performing one arithmetic operation per time unit. Thus, the number of parallel operations equals the number of time units required by the model.

Assume that the algorithm is applied to a matrix representing a five-point discretization of a differential operator on an $n \times n$ grid and that the length of the vector registers is q. Each product of an m-diagonal $n \times n$ matrix by a vector costs $m\lceil n/q \rceil$ parallel operations, where $\lceil n/q \rceil$ is the least integer greater than n/q.

The solution of $\hat{U}^{-1}x = y$ for a tridiagonal $\hat{U}$ consumes less than $8\lceil n/q \rceil \log_2 n$ parallel operations. The total cost of the factorization is $n(9 + 8 \log_2 n)\lceil n/q \rceil$, and each solution of $LUx = b$ costs $n(4 + 8 \log_2 n)\lceil n/q \rceil$ parallel operations. Notice for comparison that, for the scalar computation of MICCG (relating to the same problem), $8n^2$ floating-point operations are required for the incomplete factorization, and about $9n^2$ are needed for each solution of $LUx = b$.

Conditioning with partial nested dissection ordering

The nested dissection ordering which is so effective for speeding up direct solutions [15] may be used for parallel conditioning if the number of processors is relatively small.

To this end, consider **Figure 1(a).** If the state vector is reordered by placing the entries of region B after those of A, followed by those of C, then the matrix takes the shape shown in **Figure 1(b).**

When the Choleski factorization is performed, the blocks corresponding to regions A' and B' can be decomposed in parallel, and then the submatrix C' has to be factored separately. For an $n \times n$ square domain, the computation cost of the serial part is proportional to n^2. This is the case because the decomposition is of the incomplete kind, and thus the original sparsity pattern is retained.

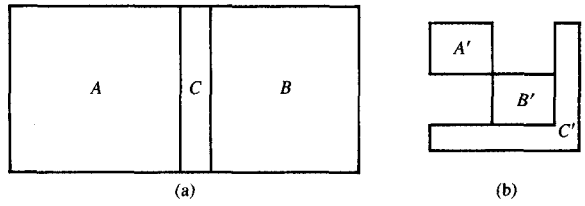

Figure 1

(a) Rectangular domain. (b) Nested dissection reordering.

The equation $Lv = r$ can be solved for regions A and B in parallel, and then for the C region. When $Ux = v$ is being solved, entries of C are solved first, and then A and B in parallel. As long as the connecting region is relatively small, this parallelization is very effective. There exist powerful algorithms for the automatic generation of nested dissection reordering [15].

Matrix splitting

If the matrix were reducible to several uncoupled submatrices, then each block could be factored independently, on a separate processor. Motivated by this idea, we suggest decoupling the matrix into several submatrices. As long as the number of splits is small relative to the space dimension, a conditioning of the decoupled matrix may serve as a reasonable conditioner of the original matrix.

We now describe the method formally: Let A be an $N \times N$ matrix. Choose break points K_i, $1 \leq K_0 \leq K_1 \leq \cdots \leq K_m = N$. Then define a split matrix

$$B = \sum_{l=1}^{m} A_l$$

so that

$$A_{l_{ij}} = \begin{cases} a_{ij} & \text{if } K_{l-1} \leq i,\ j < K_l, \\ 0 & \text{otherwise.} \end{cases}$$

A row-sum agreement correction may be applied if the main diagonals of B are modified to satisfy $\sum_j b_{ij} = \sum_j a_{ij}$ for all i. A reasonable policy will be to divide the domain into halves, quarters, eighths, etc. This kind of parallel processing will have no communications overhead during the preconditioning phase, because each processor will read and write its private subset of the vector.

Table 1 Condition number computations for Dirichlet boundary conditions.

Grid	$\kappa(K^{-1}A)$	$\kappa(A)$
10×10	1.4	49
20×20	2.3	179
40×40	4.4	681
60×60	6.6	1508
80×80	8.8	2659
100×100	10.9	4134

Table 2 The effect of anisotropy (Dirichlet boundary conditions).

ε	$\kappa(K^{-1}A)$
−0.24	1.006
−0.15	1.15
−0.05	1.29
0	1.33
0.05	1.35
0.15	1.28
0.24	1.027

Table 3 The effect of anisotropy (Neumann boundary conditions).

ε	$\kappa(K^{-1}A)$
−0.15	3.5
0	4
0.15	2.6

Table 4 Test results with three-dimensional CG preconditioned by three-dimensional MBID.

Grid size	*Number of iterations*	
	No row-sum agreement	*Row-sum agreement*
$5 \times 5 \times 5$	7	9
$10 \times 10 \times 10$	13	12
$20 \times 20 \times 20$	22	18

Although the matrix splitting degrades the conditioning, and thus the convergence rate is decreased, our experience is that the added computation is more than offset by the gains achieved by parallelization, and significant speedup factors may be achieved. The results of applying this method to a two-phase problem are described in the next section.

5. Computational experiments

• *MBID tests*

The MBID algorithm is found to have a better numerical performance than the pointwise MICCG [10], even as a sequential algorithm. The Poisson operator with Neumann boundary conditions was discretized on a 40×40 grid. To avoid a singular matrix, we added 1/1600 to the main diagonal. This addition is equivalent to integrating the time-dependent problem over a time step of unity.

To reduce the error to a relative level of 10^{-6}, 27 MICCG0 iterations were needed, whereas with block preconditioning MBID this accuracy was obtained after just 12 iterations. For 10 digits of relative accuracy the MICCG0 required 41 iterations and MBID took 18 iterations. Note that the sequential computational cost of an iteration is similar for the two methods.

The reduction in the number of iterations required by MBID is explained by the condition number improvement of the (symmetrizable) matrix $K^{-1}A$. Namely, it is found that for a 10×10 grid, $\kappa(K^{-1}A) = 1.8$ if $\varepsilon = 10^{-2}$ is added to the main diagonal. If $\varepsilon = 10^{-4}$ is added to the main diagonal, then $\kappa(K^{-1}A) = 5.3$.

For Dirichlet boundary conditions the results shown in **Table 1** were obtained.

In order to test the effect of anisotropy we took the five-point discretization of Poisson's operator with Dirichlet boundary conditions over a 10×10 grid and modified its nonzero diagonals so that the nonzero diagonals of A_ε are $(-0.25 - \varepsilon, -0.25 + \varepsilon, 1, -0.25 + \varepsilon, -0.25 - \varepsilon)$. **Table 2** gives the computed condition number of several A_ε preconditioned by MBID.

Similar tests were performed on Poisson's operator with Neumann boundary conditions, as shown in **Table 3** (0.001 was added to the main diagonal to avoid matrix singularity). These results show that weakly anisotropic problems are harder to condition than strongly anisotropic ones. This is the case regardless of the orientation of the line method to the anisotropic directions.

Table 4 shows the number of iterations required to solve the system corresponding to the three-dimensional Poisson operator with Dirichlet boundary conditions. Iteration was stopped when solutions accurate to 10 digits were obtained. The table makes evident the advantage of applying the row-sum correction for larger problems. The convergence rate of the three-dimensional MBID is slightly lower than that of the two-dimensional version.

We conclude this section on MBID tests with results of two-phase linear systems. A grid of 7×7 was used to discretize the linearization of Eqs. (3). In the case of weak convection the real part of the spectrum lies within $(-3.6, -0.04)$, and its imaginary part in $(-1.1, 1.1)$. The MBID preconditioning transforms all the eigenvalues into the $(1, 1.58)$ interval, with the imaginary part smaller than 10^{-4}.

Table 5 Matrix splitting tests.

Segments	*Iterations*	*KFLOPs*	*Speedup*
1	32	1942	1
2	45	2580	1.5
4	55	2814	2.76

Note that the MICBCG algorithm (pointwise) cannot be applied to such matrices. We also tried a strong convection case, the matrix spectrum of which has a real part in (−3.94, −0.026), and an imaginary part in (−1.01, 1.10). The MBID conditioned matrix has only real eigenvalues, all of them within (1, 1.57). Both examples required nine PBCG iterations to reach nine accurate digits.

• *Two-phase tests*

A PBCG solver was used to solve two-phase problems. Equations (3) were discretized using a block five-point Finite Difference scheme. The permeability function $T(x, y)$ was taken as shown in **Figure 2.**

In addition to estimating the behavior of some of the parallel algorithms described above, the numerical tests below display the sensitivity of the PBG method to some parameter variations. Comparisons were made with the preconditioning methods of ICCG [16] and MICCG [10]. We call their extension to Biconjugate Gradients ICBCG and MICBCG, respectively.

The method presented in the section on matrix splitting was tested on a 20 × 20 domain. High diffusion in the saturation equation was implemented by forcing $\partial C/\partial S = 0.5$. Iteration was stopped when the average error dropped below 10^{-6} of the average solution. The underlying method is MICBCG0. (MICBCGk is the Biconjugate Gradients method conditioned by the factorization method of MICCGk, as described in [10].) **Table 5** shows the speedup factors for two and for four processors. (The speedup factors were computed by dividing the CPU time needed for the split-matrix solution, by the number of segments, and comparing the result to the CPU time needed for a straight serial solution. These figures are theoretical maxima, and because synchronization is ignored, actual implementations will yield smaller gains.)

We now proceed to the computational results of various preconditioning versions as summarized in **Table 6.** It is evident that MICBCG is superior to ICBCG for our test cases. The numbers appended to MICBCG signify the number of added diagonals in the LU factors. MICBCG1 gave the lowest computation counts for the 20 × 20 domain, while for the 30 × 30 domain MICBCG2 was best.

Finally, the sensitivity of MICBCG to matrix asymmetry was tested by varying the magnitude of the convection terms. Convection becomes dominant as $\partial C/\partial S$ approaches zero. The results show that MICBCG0 is more sensitive to asymmetry than MICCG1. Neither version suffered considerable degradation when the diffusion (symmetric) terms became negligible compared to the convection (asymmetric) terms. The problem domain for all tests summarized in **Table 7** was 20 × 20.

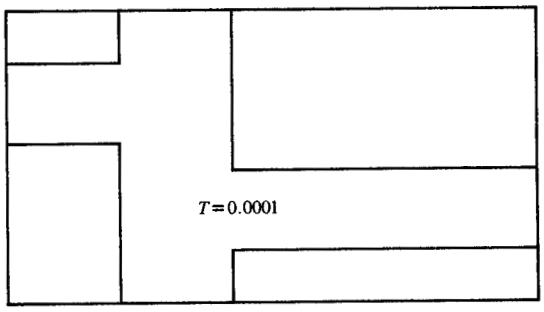

Figure 2

Layout of low-transmissivity region. T = 1 outside the central region.

Table 6 Number of iterations and total computation needed to reach six digits of accuracy.

Domain	*Version*	*Iterations*	*KFLOPs*
20 × 20	ICBCG	45	2725
"	MICBCG0	32	1942
"	ICBCG1	26	2320
"	MICBCG1	14	1307
"	ICBCG2	16	2011
"	MICBCG2	12	1573
"	MICBCG3	8	1521
30 × 30	MICBCG1	19	3924
"	MICBCG2	13	3284

Table 7 The sensitivity of MICBCG to varying convection terms.

∂C/∂S	*MICBCGx*	*Iterations*	*KFLOPs*
0.5	0	32	1942
0.5	1	14	1307
0.01	0	39	2359
0.01	1	15	1392
0.0001	0	42	2537
0.0001	1	15	1392
0.0001	2	10	1355

Acknowledgments

The authors wish to thank V. Amdurski, B. Bachelis, and G. Kozlovski for many valuable discussions. We also thank I. Tsur for his assistance in implementing the algorithms.

References

1. R. Fletcher, "Conjugate Gradient Methods for Indefinite Systems," *Proceedings, Dundee Biannual Conference on Numerical Analysis*, Springer-Verlag New York, 1975, pp. 73–89.
2. D. W. Peaceman, *Foundations of Numerical Reservoir Simulation*, Elsevier, Amsterdam, 1977.
3. S. Schechter, "Quasi-Tridiagonal Matrices and Type-Insensitive Difference Equations," *Quart. Appl. Math.* **18,** 285–295 (1960).
4. P. Concus, G. M. Golub, and G. Meurant, "Block Preconditioning for the Conjugate Gradient Method," *SIAM J. Sci. Statist. Comp.* **6,** No. 1, 220–252 (1985).
5. O. Axelsson, S. Brinkklemer, and V. P. Il'in, "On Some Versions of Incomplete Block-Matrix Factorization Iterative Methods," *Lin. Alg. & Appl.* **58,** 3–15 (1984).
6. H. A. van der Vorst, "A Vectorizable Variant of Some ICCG Methods," *Siam J. Statist. Comp.* **3,** No. 3, 350–356 (1982).
7. O. G. Johnson, C. A. Micchelli, and G. Paul, "Polynomial Preconditioning for Conjugate Gradient Calculations," *Siam J. Numer. Anal.* **20,** No. 2, 362–376 (1983).
8. P. F. Dubois, A. Greenbaum, and G. H. Rodrigue, "Approximating the Inverse of a Matrix for Use in Iterative Algorithms on Vector Processors," *Computing* **22,** 257–268 (1979).
9. S. Demko, W. F. Moss, and P. W. Smith, "Decay Rates for Inverses of Band Matrices," *Math. Comp.* **43,** 491–499 (1984).
10. I. Gustafsson, "A Class of First Order Factorization Methods," *BIT* **18,** 142–156 (1978).
11. M. Tismenetsky and I. Efrat, "An Efficient Preconditioning Algorithm and Its Analysis," *Technical Report 88-172*, IBM Israel Scientific Center, Haifa, December 1985.
12. N. K. Madsen, G. H. Rodrigue, and J. I. Karush, "Matrix Multiplication by Diagonals on a Vector/Parallel Processor," *Info. Proc. Lett.* **5,** No. 2, 41–45 (1976).
13. D. J. Evans, *Parallel Processing Systems*, Cambridge University Press, Cambridge, England, 1982.
14. B. L. Buzbee, G. H. Golub, and C. W. Nielson, "On Direct Methods for Solving Poisson's Equations," *Siam J. Numer. Anal.* **7,** 627–656 (1970).
15. A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.
16. J. A. Meijerink and H. A. van der Vorst, "An Iterative Method for Linear Systems of Which the Coefficient Matrix Is a Symmetric M Matrix," *Math. Comp.* **31,** 148–162 (1977).

Received July 5, 1985; accepted for publication November 7, 1985

Ilan Efrat *IBM Israel, Scientific Center, Technion City, Haifa, Israel.* Mr. Efrat received an M.S. in aeronautical engineering in 1978 from the Massachusetts Institute of Technology, Cambridge, Massachusetts. He worked for Intermetrics Inc. in Cambridge on validation of the Space Shuttle software from 1978 to 1980. From 1980 until 1982 he worked for DSI ISRAEL on flight simulators and flight models. Mr. Efrat has been at the IBM Haifa Scientific Center since 1982, working on numerical methods for linear algebra.

Miron Tismenetsky *IBM Israel, Scientific Center, Technion City, Haifa, Israel.* Dr. Tismenetsky joined IBM Israel in 1983 and is currently a research staff member at the Haifa Scientific Center. He received the M.Sc. in 1977 and the Ph.D. in 1981, both in mathematics, from the Technion, Haifa. From1981 to 1983 he was a Research Fellow in the Mathematics and Statistics Department of the University of Calgary, Canada. His main interests are in numerical linear algebra and matrix theory. Dr. Tismenetsky is a member of the American Mathematical Society.