TOWARD EFFICIENT IMPLEMENTATIONS OF PCCG
(PRECONDITIONED CONJUGATE GRADIENT) (U) PITTSBURGH UNIV
PA INST FOR COMPUTATIONAL MATHEMATICS AND APP.

UNCLASSIFIED R MELHEM OCT 86 ICMA-86-101

F/G 12/1 NL
INSTITUTE FOR COMPUTATIONAL
MATHEMATICS AND APPLICATIONS


Toward Efficient Implementations of PCCG Methods
on Vector Supercomputers *)

by

Rami Melhem

Department of Mathematics and Statistics
University of Pittsburgh
Technical Report ICMA-86-101

October 1986

Toward Efficient Implementations of PCCG Methods on Vector Supercomputers *)

by

Rami Melhem

Department of Computer Science

Department of Mathematics and Statistics

The University of Pittsburgh

Pittsburgh, PA 15260

*) This work is, in part supported under ON contract N00014-85-K-0339
and Air Force contract AFOSR-84-0131. The experimental results were obtained on the CRAY X-MP of the Pittsburgh Supercomputing Center.
TOWARD EFFICIENT IMPLEMENTATIONS OF PCCG METHODS ON VECTOR SUPERCOMPUTERS

Rami Melhem

Department of Computer Science

and

Department of Mathematics and Statistics

The University of Pittsburgh.

ABSTRACT

We consider large, sparse linear systems which result from the discretization of partial differential equations on regular and irregular domains, and we focus on the application of the preconditioned conjugate gradient (PCCG) method to the solution of such systems. More specifically, the goal of this paper is the efficient implementation of the PCCG method on vector supercomputers. The contribution to the above goal is made by 1) the introduction of a data structure which may be effectively manipulated on vector machines, 2) the utilization of preconditioning matrices which are obtained by incomplete factorization with diagonal update sets, and 3) the introduction of new numbering schemes for both regular and irregular grids.

\注 This work is in part supported under ONR contract N00014-85-K-0339 and Air Force contract AFOSR-84-0131. The experimental results were obtained on the CRAY X-MP of the Pittsburgh Supercomputing Center.

**) On leave from the Department of Computer Science, Purdue University, West Lafayette, IN 47907
1. INTRODUCTION

Consider the linear system

\[ A \mathbf{x} = \mathbf{b} \]  

where \( A \) is a large \( n \times n \) symmetric, positive definite matrix. It is widely accepted that the conjugate gradient method (CG) may compete with direct methods for the solution of (1) only if the matrix \( A \) is suitably preconditioned. In other words, in order to speed up the convergence of the CG iteration, it is essential to find a preconditioning matrix \( M \approx A \), and then apply the CG method to the solution of \( M^{-1} A \mathbf{x} = M^{-1} \mathbf{b} \). The resulting preconditioned CG method (PCCG) is described as follows:

**ALG1 - PCCG:**

Chose an initial guess \( \mathbf{x}_0 \).

\[ \mathbf{r}_0 = \mathbf{b} - A \mathbf{x}_0 \quad : \quad \mathbf{h}_0 = \mathbf{p}_0 = M^{-1} \mathbf{r}_0. \]

Repeat for \( i = 0, \cdots \) until convergence.

1) \( \alpha_i = \frac{\langle \mathbf{r}_i, \mathbf{h}_i \rangle}{\langle A \mathbf{p}_i, \mathbf{p}_i \rangle} \)

2) \( \mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i \quad : \quad \mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i A \mathbf{p}_i \)

3) \( \mathbf{h}_{i+1} = M^{-1} \mathbf{r}_{i+1} \)

4) \( \beta_i = \frac{\langle \mathbf{r}_{i+1}, \mathbf{h}_{i+1} \rangle}{\langle \mathbf{r}_i, \mathbf{h}_i \rangle} \)

5) \( \mathbf{p}_{i+1} = \mathbf{h}_{i+1} + \beta_i \mathbf{p}_i \)

Where, \( \langle \mathbf{x}, \mathbf{y} \rangle \) denotes the inner product of \( \mathbf{x} \) and \( \mathbf{y} \). If the preconditioning matrix is chosen such that \( M = U^T DU \), for some upper triangular matrix \( U \) and some diagonal matrix \( D \), then it is possible to reduce the linear system \( M \mathbf{h}_{i+1} = \mathbf{r}_{i+1} \) in step 3 of ALG1 into two triangular systems. The solution of these systems is the price which is to be paid for speeding up the convergence of the basic CG method (for which \( M = I \)).

Unfortunately, this price may be quite high when the PCCG is implemented on vector supercomputers. More specifically, the solution of a triangular system is a recursive pro-
cess which may be very inefficient on vector and pipelined computers. In order to overcome this difficulty, alternative preconditioners have been considered (see e.g. [1, 3, 16, 20, 23]). Also, multicoloring techniques have been used to reorder the rows and columns of the matrix such that to minimize the data dependencies, and thus minimize the effect of recursion (see e.g. [9, 17, 21, 22]). However, experimental results [9, 17, 22] show that multicolor orderings decrease the rate of convergence of the PCCG method. These results raise the following challenge which was suggested by David Young and al [25]: "Can we find an iterative algorithm which is substantially better on supercomputers than the basic CG method?".

Numerical solution techniques of partial differential equations (PDE) are major sources of large linear systems. In any of these systems, the coefficient matrix $A$ is generated by the discretization of a PDE on a finite grid $G$ such that each row of $A$ corresponds to a node in $G$. The specific discretization used defines a neighboring relation between the nodes of $G$. For example, if finite element analysis is used, then two nodes are neighbors if they belong to the same element. This neighboring relation determines the sparsity structure of $A$. More specifically, an element $a_{i,j}$ of $A$ is non-zero only if nodes $i$ and $j$ are neighbors in $G$.

For rectangular domains covered by regular grids, the non-zero elements in $A$ are confined to few diagonals. In this case, good preconditioning matrices may be obtained by the incomplete Cholesky factorization of $A$ [7, 11, 12]. Namely a Cholesky factorization in which only selected elements of $A$ are modified during the factorization process. The positions of these selected elements are specified by an update set $P$, and the corresponding factorization is denoted by $IC(P)$. For example, if $P = P_a = \{(i,j): a_{i,j} \neq 0\}$, the set of positions which contain non-zeroes in $A$, then the corresponding Cholesky factorization, $IC(P_a)$, does not allow any fill-ins in the matrix $A$. The conjugate gradient method which uses a preconditioned matrix obtained by an incomplete Cholesky factorization with update set $P$ is denoted by $IC(P)/PCCG$. 
If, on the other hand, the matrix $A$ is generated from irregular grids, then, its non-zero elements are not confined to any regular pattern. In this case, a suitable preconditioning of (1) may be obtained by choosing $M$ to be the SSOR matrix [8].

In order to obtain efficient implementations of the $IC(P)/PCCG$ methods on supercomputers, advances have to be made in three fronts. 1) suitable data structures which are efficiently manipulated on vector computers have to be chosen. 2) better preconditioning matrices have to be used. 3) renumbering schemes have to be found which satisfy a balance between the advantages of increasing the recursion span in step 3.3 of ALG1 and the disadvantages of slowing down the convergence of the PCCG method.

In this paper, we consider each of the above three fronts. More specifically, we present in Section 2 a data structure which may be used efficiently for the manipulation of general sparse matrices on vector computers. Then in Section 3, we examine the class of preconditioning matrices obtained by incomplete Cholesky factorizations. By using the notion of update sets rather than the fill-in sets, we show that the SSOR matrices are incomplete factorization matrices with an empty update set. Also, the zero extension factorizations $ICCG(0)$ [13] and $MICCG(0)$ [7] are special instances of an incomplete factorization $IC(P_d)$ in which the update set is given by $P_d = \{(i,i) : i = 1,...,n\}$. This $IC(P_d)$ factorization is, in fact, general, and may be applied to matrices which are generated from irregular grids.

In Section 4.1, column-wise multicolor schemes are introduced for rectangular grids, and in Section 4.2, numbering schemes for pierced rectangular grids are described and their effect on the vectorization potential of the PCCG method is analyzed. In Section 4.3, the multicolor numbering schemes are generalized and applied to irregular grids, and finally, in Section 5, we present some experimental results which have been obtained on the CRAY X-MP vector computer.
2. THE STRIPE DATA STRUCTURE

In [14], stripe data structures are introduced as a means to include all the non-zero elements of a matrix in a structure which is suitable for parallel processing on linear computational arrays. In this section, we will demonstrate that this same structure may be used effectively to process sparse matrices on pipelined and vector computers.

Very briefly, a stripe in an \( n \times n \) matrix \( A \) is a set of positions \( S = \{(i, \sigma(i)): i \in I \subseteq I_n\} \), where \( I_n = \{1, \ldots, n\} \) and \( \sigma \) is a strictly increasing function, that is if \((i, \sigma(i))\) and \((j, \sigma(j))\) are in \( S \), then

\[
i < j \implies \sigma(i) < \sigma(j)
\]

(1)

Note that a stripe need not include a position for every row of the matrix. If, however, it does contain a position for every row, then it is called a complete stripe. Two stripes, \( S_1 = \{(i, \sigma_1(i))\} \) and \( S_2 = \{(i, \sigma_2(i))\} \) are ordered by \( S_1 < S_2 \) if, for any \((i, \sigma_1(i)) \in S_1 \) and \((j, \sigma_2(j)) \in S_2 \) we have

\[
i \leq j \implies \sigma_1(i) < \sigma_2(j)
\]

(2)

With this, a stripe structure of \( A \) is defined as a set of stripes, \( \Sigma_A = \{S_{-\pi_1}, \ldots, S_{\pi_2}\} \), such that 1) \( S_{-\pi_1} < \cdots < S_{\pi_2} \) and 2) the position \((i, j)\) of any non-zero element \( a_{i,j} \) of \( A \) is in some stripe \( S_k \), \(-\pi_1 \leq k \leq \pi_2\). The number of stripes \( \pi = \pi_1 + \pi_2 + 1 \) is called the stripe count. In [14], an algorithm is given for the construction of stripe structures for sparse matrices, in general, and for finite element matrices, in particular [15].

The elements of a striped matrix \( A \) may be stored in an \( n \times \pi \) array, \( AE \), and the position of each element may be specified in another \( n \times \pi \) array, \( AP \). More specifically, we may set

\[
AE(i, k + \text{offset}) = \begin{cases} 
\sigma_k(i) & \text{if } (i, \sigma_k(i)) \in S_k \\
0 & \text{otherwise}
\end{cases} \quad (3.a)
\]

\[
AP(i, k + \text{offset}) = \begin{cases} 
\sigma_k(i) & \text{if } (i, \sigma_k(i)) \in S_k \\
n + 1 & \text{otherwise}
\end{cases} \quad (3.b)
\]
Here, offset = n+1, and AP(i, k + offset ) = n + 1 is used to indicate that S_k does not contain a position in row i. In Figure 1, we give an example which illustrates the concept of stripe data structures. Clearly, this stripe storage scheme is a generalization of the diagonal scheme used in [25] and [10]. It is also a more restricted form of the data structure used in [19], in which neither the strictly increasing property (1), nor the non-intersecting property (2) holds. The former property is crucial for the efficient manipulation of symmetric matrices on vector computers, while the latter is only useful for the manipulation of sparse matrices on linear computational networks [14].

If A is a symmetric matrix with non-zero diagonal elements, then it is possible to construct a stripe structure \( S_A = \{ S_{-\tau}, \ldots, S_\tau \} \), such that the lower stripes, \( S_{-\tau}, k = 1, \ldots, \tau \), are mirror images of the upper stripes, \( S_k, k = 1, \ldots, \tau \). In this case, only the elements in \( S_0, \ldots, S_\tau \) need to be stored in \( AE \) and \( AP \) (with \( offset = 1 \) in equations (3)).

Given a symmetric matrix \( A \) in stripe storage form, the computation of the product vector \( y = Ax \), for any vector \( x \), may be accomplished by the multiplication of each element \( a_{i, \sigma_k(i)} \) in an upper stripe \( S_k \) with both \( x \) and \( x_{\sigma_k(i)} \), and then the accumulation of the results in \( y_{\sigma_k(i)} \) and \( y_j \), respectively. Using a pseudo CRAY-fortran language, the matrix/vector multiplication algorithm may be written as follows:

**ALG2: Symmetric Matrix/Vector Multiplication**

1) \( x(n+1) = 0 \)
2) \( y(i) = AE(i, offset) \cdot x(i) \quad i = 1, \ldots, n \)

3) DO \( k = 1, \ldots, \pi_i \)
   
   3.1) call gather\((n, w \cdot x \cdot AP(1, k, offset))\)
   
   3.2) \( y(i) = y(i) + AE(i, k + offset) \cdot w(i) \quad i = 1, \ldots, n \)
   
   3.3) \( z(i) = AE(i, k + offset) \cdot x(i) \quad i = 1, \ldots, n \)
   
   3.4) \( w(i) = 0 \quad i = 1, \ldots, n \)
   
   3.5) call scatter\((n, w \cdot AP(1, k + offset), z)\)
   
   3.6) \( y(i) = y(i) + w(i) \quad i = 1, \ldots, n \)

Step 2 in ALG2 accumulates in \( y \) the contribution of \( S_0 \), steps 3.1 and 3.2 accumulate the contribution of \( S_k \), and steps 3.3-3.6 accumulate the contribution of \( S_{-k} \). The strictly increasing property (1) is essential for the correctness of the above algorithm because it ensures that if \( i \neq j \), then \( \sigma_k(i) \neq \sigma_k(j) \). If this is not satisfied, then \( z(\sigma_k(i)) \) and \( z(\sigma_k(j)) \) will be scattered to the same location in \( w \), and thus only one of them will be accumulated in \( y \).

Besides matrix/vector multiplication, the stripe structure may be used in the solution of triangular linear systems. For example, consider the upper triangular system \((D + U)x = b\), where \( D \) is a diagonal matrix, and \( U \) is a strictly upper triangular matrix with a stripe structure \( \Sigma_U = \{S_1, \ldots, S_w\} \). The solution of this system is a recursive process in which the calculation of \( x_i \) proceeds in the order \( i = n, \ldots, 1 \), with \( x_i \) depending on the previously calculated values \( x_j, j > i \). The minimum recursion span, that is the minimum integer \( d \) such that \( x_i \) does not depend on \( x_{i+1}, \ldots, x_{i+d} \), is equal to the upper zero stretch of the matrix defined as follows:

**Definition:** The upper zero stretch of a matrix \( A \) is the largest integer \( \Delta_2 \) such that \( a_{i,j} = 0 \) for \( i = 1, \ldots, n \) and \( i < j < i + \Delta_2 \). In other words, \( \Delta_2 \) is the size of the maximum band above (and including) the main diagonal which contains zero elements.

For the upper triangular matrix \( U \), the upper zero stretch is found, by properties (1) and (2), to be
\[
\Delta_2 = \max \{ \sigma_1(i) - i : (i, \sigma_1(i)) \in S_1 \}
\]

Given \( \Sigma_U \) and \( \Delta_2 \), the triangular system \((D + U)x = b\) may be solved using operations on vectors of length \( \Delta_2 \). Similarly, given a strictly lower triangular matrix \( L \) and a stripe structure \( \Sigma_L = \{S_{-1}, \ldots, S_{-\pi} \} \), a lower triangular system of the form \((L + D)x = b\) may be solved using operations on vectors of length \( \Delta_1 \), where \( \Delta_1 \) is the lower zero stretch of \( L \) given by

\[
\Delta_1 = \max \{ i - \sigma_{-1}(i) : (i, \sigma_{-1}(i)) \in S_{-1} \}
\]

When the IC/PCCG method is applied to symmetric matrices, it is necessary to solve in each iteration two triangular systems of the forms \((L + D)x = b\) and \((D + U)y = x\), with \( L = U^T \). If \( \Sigma_U = \{S_1, \ldots, S_{\pi} \} \) is a stripe structure for \( U \), then it is possible to construct a stripe structure \( \Sigma_L = \{S_{-1}, \ldots, S_{-\pi} \} \) for \( L \) such that each stripe \( S_{-k} \), \( 1 \leq k \leq \pi \), is the mirror image of \( S_k \). In this case, only \( U \) needs to be stored. Let \( UE \) and \( UP \) be the arrays used to store \( U \) in a way analogous to equations (3) (with offset = 0), and let \( \Delta_1 \) be the lower (upper) zero stretch of \( L \) (\( U \)).

The solution of \((L + D)x = b\) is affected by the absence of an explicit storage for the elements of \( L \). More specifically, consider the usual forward substitution algorithm. In this algorithm, the solution proceeds such that, after the computation of \( x_1, \ldots, x_{r \Delta_1} \), for some \( r \), the next \( \Delta_1 \) elements of \( x \) are computed by, first, computing the \( \Delta_1 \) component \( z_i, i = r \Delta_1 + 1, \ldots, (r + 1) \Delta_1 \), of the vector \( z = b - Lx \), and then dividing each \( z_i \) by \( d_i \), the \( i^{th} \) component of \( D \). However, the above scheme is not efficient because it is difficult to access in \( UE \), the elements of \( \Sigma_L \) which belong to rows \( r \Delta_1 + 1, \ldots, (r + 1) \Delta_1 \). In order to overcome this difficulty, we use a column sweep algorithm (\( ji \) algorithm according to [5]). More specifically, we let \( L_j \) be the \( j^{th} \) column of \( L \) and we compute the vector \( z = b - Lx = b - \sum_{i=1}^{n} L_ix_i \) progressively by accumulating \( \sum_{i=r \Delta_1 + 1}^{(r + 1) \Delta_1} L_ix_i \) into \( z \) as soon as the appropriate values of \( x \) are computed. This requires the access of the elements of \( \Sigma_L \) which
belong to columns \( r \Delta_t + 1, \ldots, (r + 1)\Delta_t \). That is the elements of \( \Sigma_U \) which belong to rows \( r \Delta_t + 1, \ldots, (r + 1)\Delta_t \). These elements may be easily accessed in \( UE \).

More precisely, the solution of \( (L + D)x = b \) may be described by the following algorithm, in which we assume, for simplicity, that \( n = \Delta_t \cdot m \) for some integer \( m \):

**ALG3: Forward Substitution**

1) \( z(i) = b(i) \quad i = 1, \ldots, n \)
2) \( x(i) = z(i) / d(i) \quad i = 1, \ldots, \Delta_t \)
3) FOR \( r = 0, \ldots, m - 1 \) DO
   3.0) Let \( i_s = r \Delta_t + 1 \) and \( i_e = (r + 1)\Delta_t \)
   3.1) FOR \( k = 1, \ldots, \pi_r \) DO
   3.1.1) \( w(i) = UE(i, k) \cdot x(i) \quad i = i_s, \ldots, i_e \)
   3.1.2) \( u(i) = 0 \quad i = i_s, \ldots, n \)
   3.1.3) call scatter(\( \Delta_t \cdot u \cdot UP(i, k), w(i) \))
   3.1.4) \( z(i) = z(i) - u(i) \quad i = i_s, \ldots, n \)
   3.2) \( x(i) = z(i) / d(i) \quad i = i_s + 1, \ldots, i_e + \Delta_t \)

Note that in 3.1.3, the \( \Delta_t \) elements of \( w \) may be scattered anywhere in \( u \). For this reason the summation in 3.1.4 runs up to \( i = n \). An alternative way for writing the inner loop in step 3.1 is as follows

3.1.1) call gather(\( \Delta_t \cdot u \cdot (i, k), z(i, UP(i, k)) \))
3.1.2) \( u(i) = u(i) - UE(i, k) \cdot x(i) \quad i = i_s, \ldots, i_e \)
3.1.3) call scatter(\( \Delta_t \cdot z(i, UP(i, k)), u(i) \))

Clearly, this second alternative reduces the number of operations. Moreover, the subtraction and the multiplication are performed in the same step which allows for the chaining of the two operations. The choice between the two alternatives should depend on the relative execution time of the different vector instructions on the specific computer used. For example, actual measurements on the CRAY X-MP show that the slow down due to the additional call to "gather" in the second alternative more than offsets the gain obtained from the
chaining of the operations and the lower operation count.

3. INCOMPLETE FACTORIZATION WITH DIAGONAL UPDATE SETS

Given a symmetric positive definite matrix $A$ of order $n$, the incomplete factorization of $A$ is a splitting of the form

$$A = M + R$$

where

$$M = (I + U)^T D (I + U)$$

and $I$ is the identity matrix, $D$ is a diagonal matrix and $U$ is a strictly upper triangular matrix. The matrices $D$ and $U$ are determined from $A$ by applying a factorization procedure in which only specific elements of $A$ may be updated during the factorization. More specifically, if $P = \{(i,j) : i,j \in [1,n]\}$ is any subset of positions in $A$, then the corresponding incomplete factors of $A$ are obtained as follows:

**ALG4: Incomplete Cholesky Factorization**

FOR $i = 1, \ldots, n$ DO

$$d_{i,i} = a_{i,i}$$

FOR $j = i + 1, \ldots, n$ DO

$$u_{i,j} = a_{i,j} / d_{i,i}$$

FOR $j = i + 1, \ldots, n$ DO

FOR $k = j, \ldots, n$ DO

IF $(j,k) \in P$ THEN $a_{j,k} = a_{j,k} - u_{i,j} * u_{i,k} * d_{i,i}$

We call $P$ the update set and we call the corresponding factorization (4.b) and splitting (4.a) the $IC(P)$ factorization and splitting of $A$, respectively. A possible choice of $P$ is the set $P_0 = \{(i,j) : a_{i,j} \neq 0\}$ of positions which contain non-zero elements in $A$.

Manteuffel [11] suggested a shifted incomplete Cholesky splitting ($SIC$) in which the off diagonal elements of $A$ are scaled by some factor $\omega$ before the factorization. In order to be more specific, let $F$ and $\Lambda$ be the upper triangular and the diagonal parts of $A$.
respectively. That is
\[ A = F^T + \Lambda + F. \] (5)

With this, consider the shifted matrix
\[ \bar{A} = \Lambda + \omega (F^T + F) \]
and let \( \bar{U} \) and \( \bar{D} \) be the factors produced by the application of ALG4 to \( \bar{A} \). The \( SIC(P, \omega) \) splitting of \( A \), corresponding to the update set \( P \) and the shift factor \( \omega \), is then given by
\[ A = (I + \bar{U})^T \bar{D} (I + \bar{U}) + \bar{R} \]

Note that \( SIC(P, \omega) \) reduces to \( IC(P) \) for \( \omega = 1 \).

Factorizations with update sets larger than, or equal to, \( P_o \) have been considered in the literature (e.g. [13]) for matrices which are generated from rectangular grids. For these matrices, ALG4 may be greatly simplified. However, for matrices which are generated from irregular grids, the choice of \( P \supseteq P_o \) does not seem useful because, even though the execution of ALG4 is rather costly, only few of the terms \( u_{i,j} * u_{i,k} \) in the inner loop are non-zeroes, thus causing only few updates in the entries of \( A \). In this case, the SSOR splitting of \( A \) may be used.

**SSOR splitting as incomplete factorization.**

If the update set \( P \) is taken to be the empty set \( \Phi \), then it is easy to see that the factors \( U_\Phi \) and \( D_\Phi \) resulting from ALG4 are given by
\[ D_\Phi = \Lambda \]
\[ U_\Phi = \Lambda^{-1} F \]

Also, if the off-diagonal elements of \( A \) are shifted by \( \omega \) before factorization, then the resulting factors are given by
\[ \bar{D}_\Phi = \Lambda \] (6.a)
\[ \bar{U}_\Phi = \omega \Lambda^{-1} F \] (6.b)

Hence, the \( SIC(\Phi, \omega) \) preconditioning matrix is
\[ \overline{M}_g = (I + \omega \Lambda^{-1} F)^T \Lambda (I + \omega \Lambda^{-1} F) \]
\[ = (\Lambda + \omega F)^T \Lambda^{-1} (\Lambda + \omega F) \]  

(7)

Which is equal to the SSOR preconditioning matrix, up to a scalar factor of \( \omega(2-\omega) \). This scalar factor does not have any effect from the point of view of the PCCG method.

**Diagonal update Cholesky factorization.**

Consider non-shifted incomplete factorization. We may envision a linear scale on which the preconditioners obtained from ALG4 are laid according to the size of the update set \( P \) (see Fig 2). Clearly, the exact factorization \( IC(P_e) \) corresponding to \( P_e = \{(i,j): i,j = 1,...,n \} \), and the SSOR splitting \( IC(P_o) \) are laid at the two ends of the scale, with \( IC(P_d) \) somewhere in between. The results of Manteuffel [11] for \( M \) matrices show that, if \( P_1 \subseteq P_2 \), then the \( IC(P_1)/PCCG \) may not outperform \( IC(P_2)/PCCG \). In other words, the performance of the \( IC/PCCG \) improves when we move to the right on the scale of Fig 2.

<table>
<thead>
<tr>
<th>( \Phi )</th>
<th>( P_d )</th>
<th>( P_a )</th>
<th>( P_e )</th>
<th>size of the update set</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSOR</td>
<td>diagonal update</td>
<td>standard IC</td>
<td>exact IC</td>
<td>Cholesky</td>
</tr>
</tbody>
</table>

Fig 2 - The spectrum of incomplete Cholesky factorizations

The diagonal update factorization is the factorization obtained from ALG4 with the update set given by \( P_d = \{(i,i): i = 1,...,n \} \). That is, only the diagonals of \( A \) are updated in ALG4. If we shift the matrix \( A \) by \( \omega \) before applying ALG4, then it is easy to see that the factors \( \overline{D}_d \) and \( \overline{U}_d \) resulting from the factorization are given by

\[
\overline{D}_d = \tilde{\Lambda} \]
\[
\overline{U}_d = \omega (\tilde{\Lambda})^{-1} F
\]

where, by ALG4, the elements \( \tilde{\lambda}_i \), \( i = 1,...,n \) of the diagonal matrix \( \tilde{\Lambda} \) are given by

\[
\tilde{\lambda}_i = a_{i,i} - \sum_{j=1}^{i-1} u_{i,j} / \tilde{\lambda}_j \quad i = 1,...,n
\]
The equations which express the solution of upper triangular linear systems are similar to the above equations, with the division replaced by a multiplication. Assuming that the execution of a vector divide operation is at most four times slower than a vector multiply (on the CRAY the ratio is found to be 2.5 for vectors of length 6400), then it is clear that the cost of computing $\tilde{A}$ is at most equal to the cost of one PCCG iteration. Given $\tilde{A}$, the $SIC(P_d,\omega)$ preconditioning matrix is

$$\tilde{M}_d = (I + \omega \tilde{A}^{-1} F)^T \tilde{A} (I + \omega \tilde{A} F) \quad (8)$$

In addition to being easy to compute, both SSOR and $SIC(P_d,\omega)$ preconditioners have an advantage over factorizations with update sets larger than $P_d$. Namely, they may lead to reduced computational work in each PCCG iteration. More specifically, each iteration in ALG1 requires, in addition to $O(n)$ operations, both the matrix/vector multiplication $Ap$, and the solution of two triangular systems in step 3. However, for SSOR and $SIC(P_d,\omega)$ preconditioners, ALG1 may be rewritten such that it requires only the solution of two triangular systems, thus reducing the work per iteration by a factor of, almost 2. For example, if the SSOR matrix (7) is used in ALG1, and the following substitution is made

$$H = (\Lambda + \omega F)^{-T} A (\Lambda + \omega F)^{-1} \quad (9)$$

then, it is possible to rewrite the algorithm such that it only involves, besides $O(n)$ operations, the multiplication of a vector $p$ by $H$ (see [6] for details). From (9) and (5), this multiplication may be computed as follows:

$$H p = \omega^2 \left( v + \left( \frac{1}{\omega} \Lambda + F \right)^{-T} \left( p + \frac{\omega-2}{\omega} \Lambda v \right) \right) \quad (10)$$

where $v = \left( \frac{1}{\omega} \Lambda + F \right)^{-1} p$. That is the multiplication $Hp$ may be performed by solving two triangular systems.

The same argument applies to the $SIC(P_d,\omega)$ matrix (8), which has the same form as (7), except that $\Lambda$ is replaced by $\tilde{A}$. In this case, the multiplication $Hp$ may be obtained from equation (10) with $\Lambda$ replaced by $\tilde{A}$. 
Given that $P_d \geq \Phi$, then according to [11], $IC(P_d)/PCCG$ is expected to outperform $SSOR/PCCG$ for $M$ matrices. The examples considered in Section 5 suggest that, in general, $IC(P_d)/PCCG$ is at least as efficient as the $SSOR/PCCG$.

4. MULTICOLOR NUMBERING SCHEMES

The major difficulty in the vectorization of the IC/PCCG method concerns the recursive nature of the solution of triangular linear systems. In order to overcome this difficulty, many multicolor numbering schemes have been suggested [9, 17, 21, 22] for rectangular grids. Their goal is to obtain a coefficient matrix $A$ which may be partitioned into submatrices $A_{i,j}, i, j = 1, ..., p$, for some $p$, such that $A_{i,j}, i = 1, ..., p$, are diagonal matrices. This goal is achieved by applying an algorithm which has the following form:

**ALG5: A Global Multicolor Numbering Scheme**

1) Assign to each node in the grid a color from a set of $p$ colors such that neighboring nodes have different colors. Choose $p$ to be as small as possible.

2) For each color $c = 1, ..., p$, number all the nodes which have color $c$ in a column-wise sequential order.

The above numbering scheme is global in the sense that all the nodes which have the same color are numbered consecutively. Although global numbering schemes produce matrices with large zero stretch (approximately $\frac{n}{p}$, where $n$ is the number of nodes in the grid), experience [9, 17, 22] shows that these schemes worsen the condition of the matrix, thus slowing down the convergence of the IC/PCCG method. In this section, we suggest a numbering scheme which compromises between the zero stretch of the matrix and the convergence properties of the PCCG. We first introduce the scheme for rectangular grids.

4.1. Column-wise multicolor numbering of rectangular grids

Let $T$ be an upper (or a lower) triangular matrix given in striped form. It was shown in Section 2 that the larger the zero stretch of $T$, the more efficient is the solution of $Tx = b$.
on vector computers. However, the advantage of having large zero stretches when the
IC/PCCG is applied to the solution of linear systems should be put in its proper prospective. More specifically, it is important to note the following:

1) the zero stretch does not affect the matrix/vector multiplication, the inner products computation or the vector addition operations in ALG1.

2) the solution of a triangular linear system includes data movement operations (e.g. scatter and gather in ALG3). These operations are slower than vector arithmetic operations on existing vector computers, and hence they dominate the execution time of the solution.

3) the advantage of having long vectors is relatively limited in data movement operations compared to arithmetic operations. More specifically, the execution time of a vector arithmetic operation on a pipeline computer is usually specified by $\tau_s + \delta \tau_p$, where $\tau_s$ is a vector setup time, $\tau_p$ is pipe unit time and $\delta$ is the vector length. Usually, $\tau_s$ is much larger than $\tau_p$, which makes it very advantageous to use long vectors. For example, a multiply/add vector operation on the CRAY X-MP consumes 70 and 170 $\mu$-sec. for $\delta = 640$ and 6400, respectively. On the other hand, using long vector in data movement operations is less advantageous because the pipe unit time for vector data movement operations is relatively large, and depends on the distribution of data in memory. For example, a specific gather operation on the CRAY X-MP consumes 40 and 370 $\mu$-sec for $\delta = 640$ and 6400, respectively. The corresponding times for a scatter operation are 90 and 410 $\mu$-sec., respectively.

Given the above facts, our main goal in the vectorization of the IC/PCCG method should be to increase the zero stretch in the matrices to a point which prevents the recursive solution of triangular systems from dominating the entire process. However, once the execution time of ALG1 is not dominated by the time for step 3, any further increase of the zero stretches will have a relatively limited effect on the execution time of the IC/PCCG iteration. For instance, when step 3 consumes only one fourth of the execution time of ALG1, the doubling of the speed of step 3 speeds up the entire algorithm by less than 1.15.
The above discussion suggests that a numbering scheme which produces "reasonably conditioned" matrices with "reasonable" zero stretches may be better than a numbering scheme which produces "ill conditioned" matrices with very large zero stretches. The column-wise multicoloring scheme belongs to the former class. It is described, for an \( r \times m \) rectangular grid, by the following algorithm:

**ALG6: A Column-Wise Multicolor Numbering Scheme**

1) Assign to each node in the grid a color from a set of \( q \) colors such that neighboring nodes that are in the same column have different colors. Chose \( q \) to be as small as possible.

2) For \( j = 1, \ldots, m \)
   
   For each color \( c = 1, \ldots, q \), number, sequentially, the nodes in column \( j \) which have color \( c \).

It may be shown that ALG6 produces matrices with zero stretches equal to \( \frac{r}{q} \), while ALG5 produces matrices with zero stretches \( \frac{r m}{p} \). The minimal values of \( q \) and \( p \) for different discretization stencils are given in Table 1, where \( FD_5 \) denotes the 5-point finite difference discretization, and \( FE_3, FE_4, FE_6 \) and \( FE_9 \) denote finite element discretizations with 3-node triangles, 4-node rectangles, 6-node triangles and 9-node rectangles, respectively.

<table>
<thead>
<tr>
<th>( p ) for global schemes</th>
<th>( FE_3 )</th>
<th>( FE_4 )</th>
<th>( FE_6 )</th>
<th>( FE_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( FD_5 )</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>( q ) for column-wise schemes</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1 - minimal number of colors in multicoloring schemes

4.2. Numbering pierced rectangular grids

Pierced rectangular grids (see Fig 3) are defined in [15] to be rectangular grids from which some subrectangles are removed. This type of grids is useful because any irregular domain may be covered by a pierced rectangular grid, or by a grid which is isomorphic to a
pierced rectangular grid.

Fig 3 - A pierced rectangular grid numbered with a 2-color column-wise scheme.

Let $Q$ be a rectangular grid which contains $n_Q$ nodes numbered by the integers $1,...,n_Q$ according to some numbering scheme, and let $\Omega$ be a grid which is obtained by piercing $Q$ and which contains $n_\Omega$ nodes. A renumbering of the nodes which are in $Q$ may then be defined by a function $\nu$ that assigns to each node $i$, which is in both $Q$ and $\Omega$, a unique number $\nu(i), 1 \leq \nu(i) \leq n_\Omega$. This renumbering is said to be deduced from the numbering of $Q$ if $\nu$ is derived as follows:

**ALG7: Numbering Pierced Rectangular Grids**

\[ \begin{align*}
& l = 0 \\
& \text{FOR } i = 1, \ldots, n_Q \text{ DO} \\
& \quad \text{IF node } i \text{ is in } \Omega \text{ THEN } \{ l = l + 1 : \nu(i) = l \} \\
& \quad \text{ELSE } \{ \nu(i) = \top \text{ (undefined)} \} \\
\end{align*} \]

The upper (or lower) zero stretch of the symmetric matrix generated from the pierced rectangular grid $\Omega$ is then given by

\[ \Delta = \min \{ |\nu(i) - \nu(j)| : i \text{ and } j \text{ are neighboring nodes in } \Omega \} \quad (11) \]
The specific value of $\Delta$ in (11) depends on the shape of $\Omega$ and on the numbering scheme. However, due to the regularity of global and column-wise multicolor numberings, it is possible, in both schemes, to establish upper and lower bounds for $\Delta$. In the remainder of this section, we will find such bounds for the $FE_4$ and $FE_9$ discretizations. Similar bounds may be derived for other discretizations. We start by deriving bounds on $\Delta$ for column-wise multicolor numberings.

**Theorem 1:** If either $FE_4$ or $FE_9$ discretization is used, and a column-wise multicoloring scheme is applied with $q=2$ or $3$, respectively, then

$$\frac{ce_{\text{min}}}{2} \leq \Delta \leq ce_{\text{min}}$$

where $ce_{\text{min}}$ is the minimum number of elements in any column of elements in $\Omega$.

**Proof:** see the appendix.

For global coloring schemes, the bounds on $\Delta$ may be given in terms of the total number of nodes $n_r$ in $\Omega$. For this, we first estimate the number of nodes in $\Omega$ which are colored by any particular color.

**Lemma 1:** Let the $n_r$ nodes in a pierced rectangular domain $\Omega$ be numbered by first using a global multicolor scheme to number the smallest rectangular domain $Q$ which enclose $\Omega$, and then deducing the node numbers in $\Omega$ using ALG7. If $FE_4$ or $FE_9$ discretizations are used with $p=4$ and $9$, respectively, then for any particular $l$, $1 \leq l \leq p$, the number, $n'_r$, of nodes given the color $l$ is bound by

$$\left| \frac{1}{2\sqrt{p} - 1} \right|^2 n_r \leq n'_r \leq \left| \frac{2}{\sqrt{p} + 1} \right|^2 n_r.$$  \hspace{1cm} (12)

**Proof:** see the appendix.

In Figure 4, we color some specific rectangular grids to show that the bounds in Lemma 1 are tight (the numbers in Fig 4 refer to colors). More specifically, for $FE_4$, the grid in Figure 4(a) gives $n'_1 = \frac{1}{9} n_r$ and $n'_4 = \frac{4}{9} n_r$. Similarly, for $FE_9$, $n'_9 = \frac{1}{25} n_r$ in Fig.
ure 4(b), and $n_A = \frac{1}{4}n_n$ in Figure 4(c). Clearly, the grids in Fig 4 are rectangular grids, which are special cases of pierced rectangular grids.

![Grids](image)

Figure 4 - Tightening the bounds of Lemma 1

The following theorem uses Lemma 1 to bound the zero stretch of the matrix resulting from the finite element discretization.

**Theorem 2**: Given the hypothesis of Lemma 1, the zero stretch of the matrix resulting from the finite element discretization is bounded by

$$\left| \frac{1}{2\sqrt{p} - 1} \right|^2 n_n - 1 \leq \Delta \leq \left| \frac{2}{\sqrt{p} + 1} \right|^2 [n_n + p].$$

**Proof**: see the appendix.

Given a pierced rectangular grid $\Omega$, the above bounds may be used for the apriori estimation of the zero stretch which result from specific numbering schemes. The experiments of Section 5 show that the convergence rate of the PCCG method is inversely related to the zero stretch, $\Delta$, of the matrix, and that beyond a specific value of $\Delta$, the reduction in the execution time of each PCCG iteration does not offset the increase in the number of iterations. In this context, apriori estimates of $\Delta$ may help in choosing the best numbering scheme for $\Omega$.

### 4.3. A multicolor numbering for irregular grids

In finite element analysis, it is often useful to change the density of the discretization points among the different regions of the domain. This is especially advantageous if a rough estimate of the solution is known or if an adaptive solution technique is applied. In both cases, it is preferable to increase the density of the discretization points in the regions where
the solution is known to (or is found to) change rapidly. This desirable variation in density may not, usually, be accomplished without the use of irregular grids.

All previous schemes for numbering irregular grids aimed at the reduction of the band-width and the profile of the resulting matrices. This is particularly useful if direct methods are used for the solution of the linear systems resulting from the grids. However, as we explained in Section 2, if iterative solution methods are applied and vector computers are used, then the efficiency of the solution process is determined by the zero stretch of the matrix. In this section, we describe a multicoloring technique which aims at increasing the zero stretch of the matrix.

Let $G$ be the graph corresponding to a given irregular grid. That is, the nodes in $G$ correspond to the nodes in the grid, and any two neighboring nodes in the grid are connected by an edge in $G$. Guided by the global multicoloring scheme for rectangular grids, our goal is to partition $G$ into disjoint subsets of nodes $G^l, l=1,...,p$, such that 1) $p$ is as small as possible, 2) no two node in any partition $G^l$ are neighbors, and 3) the variation in the size of the partitions $G^l, l=1,...,p$ is minimal. Given such a partitioning, it is possible to number the nodes in each subset consecutively. However, this does not guarantee a large zero stretch if the numbering of the nodes within each partition $G^l$ is done arbitrarily. For example, if the node numbered first in $G^l$ is connected to the node numbered last in $G^{l-1}$, then the zero stretch is equal to unity. In other words, the relation between the nodes in $G^l$ and $G^{l-1}$ should be taken into consideration in the numbering process.

The numbering scheme which we suggest consists of the following four steps:

1) The generation of a level structure $\{V_1, V_2, \cdots\}$, such that the nodes in each level $V_u$ are not connected to the nodes in levels $V_w, w > u + 1$ or $w < u - 1$. In other words, nodes in $V_u$ may only be connected to nodes in $V_{u-1}$ and $V_{u+1}$.

2) For each level $V_u$, the partitioning of the nodes in $V_u$ into the minimum number of independent sets $V_u^1, V_u^2, \cdots$ such that the nodes in each $V_u^\lambda$ are independent. That is no two nodes in $V_u^\lambda$ are connected by an edge in $G$. 
3) The formation of the partitions $G^1$, and

4) The actual numbering of the nodes.

In order to generate the level structure, we start from an arbitrary initial level $V_1$, and we proceed in a way very similar to the Cuthill-McKee Algorithm [4]. More specifically, given a level $V_u$, we construct the next level $V_{u+1}$ by including in it any node which is connected to a node in $V_u$ and which is not in any previous level. A criteria which should guide the choice of $V_1$ is the desire to have $\sum_{u=\text{even}} |V_u| \approx \sum_{u=\text{odd}} |V_u|$, where $|V_u|$ is the number of nodes in $V_u$. For example, if we denote by $CN_u$ the $u^{th}$ column of nodes in a pierced rectangular domain, then for $FE_4$ discretization, $|CN_1, CN_2, \ldots|$ is a level structure in which the ratio of $\sum_{u=\text{odd}} |CN_u|$ to $\sum_{u=\text{even}} |CN_u|$ may be proven to be larger than 0.5 and smaller than 2 (see proof of Lemma 1 in the appendix).

Given a specific level, $V_u$, the optimal generation of the independent sets $V_u^1, V_u^2, \ldots$ is known to be NP complete. However, a simple algorithm which may be used for this generation proceeds by considering the nodes in $V_u$ in any given order. For each node $i$, $i$ is added to the first set $V_u^\lambda$ which does not contain a node that is a neighbor to $i$. If this condition is not satisfied for any of the sets constructed so far, then, a new set which contains only node $i$ is created. Let $s_u$ be the number of independent sets constructed for $V_u$, and let $|V_u^\lambda|$, $1 \leq \lambda \leq s_u$ be the number of nodes in the set $V_u^\lambda$.

For pierced rectangular domains and $FE_4$ discretizations, the number of independent subsets is constant for any level $u$. Namely $s_u = 2$. However, for general grids, $s_u$ may not be constant. In this case, we define $s_{\text{max}} = \max_u \{s_u\}$. Note that, for $s_u < \lambda \leq s_{\text{max}}$, the sets $V_u^\lambda$ are empty.

Given the independent sets, the partitions $G_1, G_2, \ldots$ may be constructed by the combination of the appropriate independent sets such that the variation in the size of the different partitions is as small as possible. For this, the following algorithm may be applied...
ALG8: Creation of independent partitions

For $l = 1, \ldots, 2s_{\text{max}}$, initialize $G^l$ to the empty set.

For $u = 1, 3, 5, \ldots$ Do

For $\lambda = 1, \ldots, s_{\text{max}}$ Do

Choose the partition $G^l$, $1 \leq l \leq s_{\text{max}}$ which contains, so far, the minimum number of nodes (break ties arbitrarily), and set $G^l = G^l \cup V_u^\lambda$.

End Do

End For

For $u = 2, 4, 6, \ldots$ Do

For $\lambda = 1, \ldots, s_{\text{max}}$ Do

Choose the partition $G^l$, $s_{\text{max}} < l \leq 2s_{\text{max}}$ which contains, so far, the minimum number of nodes (break ties arbitrarily), and set $G^l = G^l \cup V_u^\lambda$.

End Do

End For

After obtaining the $2s_{\text{max}}$ partitions, we may then start the actual numbering of the nodes. For this, we consider the partitions in the order $G^1, G^2, \ldots$, and within each partition, $G^l$, we number the nodes sequentially such that if $V_u^k$ and $V_u^\lambda$ are two subsets in $G^l$ with $v > u$, then the nodes in $V_u^k$ are numbered before the nodes in $V_u^\lambda$.

Clearly, ALG8 guarantees that the sizes of the partitions $G^l$, $l = 1, \ldots, s_{\text{max}}$ does not differ by more than the size of the largest subset among all subsets $V_u^\lambda$, $\lambda = 1, \ldots, s_{\text{max}}$, $u = 1, 2, \ldots$. The same applies to the partitions $G^l$, $l = s_{\text{max}} + 1, \ldots, 2s_{\text{max}}$. However, no relation may be established between the size of two partitions $G^l$ and $G^\lambda$, $l \leq s_{\text{max}}$ and $\lambda > s_{\text{max}}$, without an explicit assumption about the size of each level $V_u$ with respect to the following level $V_{u+1}$. If an assumption of this kind is made, then it may be possible, following a reasoning similar to that of Theorem 2, to find a lower bound on the value of the zero stretch of the matrix resulting from this multicolor numbering scheme. We will not pursue this issue any further.

We illustrate our numbering scheme by applying it to the grid of Figure 5. In this figure, a level structure $\{V_1, \ldots, V_6\}$ is indicated by bold dashed lines, and within each level $V_u$, the nodes which are assigned to the subsets $V_u^1, V_u^2$ and $V_u^3$ are marked by the
symbols $\bullet$, $\square$ and $\Delta$, respectively. The application of ALG8, then, results in the following partitions:

\[
\begin{align*}
G^1 &= V^1_1 \cup V^2_1 \cup V^3_1 \cup V^4_1 \\
G^2 &= V^1_2 \cup V^2_2 \cup V^3_2 \cup V^4_2 \\
G^3 &= V^1_3 \cup V^2_3 \cup V^3_3 \cup V^4_3 \\
G^4 &= V^1_4 \cup V^2_4 \cup V^3_4 \\
G^5 &= V^1_5 \cup V^2_5 \cup V^3_5 \\
G^6 &= V^1_6 \cup V^2_6 \cup V^3_6
\end{align*}
\]

which leads to the numbering shown. It is easy to check that the zero stretch resulting from this numbering is $\Delta = 8$. How good is this result compared with the maximum possible zero stretch for this specific grid?, and is there any numbering scheme which is not NP complete, and which will produce the maximum zero stretch?. We do not know the answer to these questions, but some preliminary results lead us to believe that the answer to the second question is negative.

Finally, we should note that the above numbering scheme is global in the sense that all the nodes which are given a particular color are numbered consecutively. A level-wise numbering scheme analogous to the column-wise scheme of Section 4.1 may also be applied. However, the construction of the subsets $V^1_a$, $V^2_a$, ... in this case should be done such that
5. NUMERICAL RESULTS

In order to evaluate the diagonal update technique of Section 3, and to test the column-wise multicolor numbering scheme of Section 4, we implemented the PCCG method on the CRAY X-MP using the stripe structure approach of Section 2. The implementation allows for the solution of general symmetric sparse systems and provides for no preconditioning, SSOR, $SIC(P, \omega)$ or $SIC(P_d, \omega)$ preconditioning. The PCCG iteration is stopped when the 2-norm of both the residual and the step size fall below $10^{-5}$.

The linear systems used in our experiments are generated by using a modified version of the finite element code of [2]. The modification incorporates the stripe data structure and provides options for 1) the automatic specification of pierced rectangular domains and grids, 2) the choice of the numbering scheme (one color, column-wise multicolor or global multicolor), and 3) the application of finite difference discretization.

In this section, we report the results obtained for the following three problems:

Problem 1:

\[ \nabla^2 u - u = 0 \quad \text{on } D \]
\[ u = 1 + x + y \quad \text{on } \partial D \]

where $D$ is the unit square. For this problem we consider both $FD_5$ and $FE_4$ discretizations. For $FD_5$, we consider many grid sizes and we report in Table 2 the number of iterations $I$ and the execution time $T$ (in seconds) for the PCCG corresponding to the numbering schemes discussed in Section 4. We also report $I$ and $T$ for the basic (non-preconditioned) conjugate gradient method. The execution times are omitted for the $64 \times 64$ grid because they are relatively small and thus inconclusive. Note that the global two color numbering scheme is the usual red/black numbering scheme.

The results in Table 2 are obtained by fixing the shift factor, $\omega$, (over relaxation factor) to unity. In order to observe the effect of changing $\omega$, we report in Table 3, the results
Table 2 - The number of PCCG iterations and the execution time (in seconds) for Problem 1 with \( FD_4 \) discretization (\( \omega = 1 \)).

<table>
<thead>
<tr>
<th>grid size</th>
<th>precond. method</th>
<th>one color</th>
<th>global 2-colors</th>
<th>column-wise 2-colors</th>
<th>no prec. 1-color</th>
</tr>
</thead>
<tbody>
<tr>
<td>64x64</td>
<td>SSOR ( IC(P_d) )</td>
<td>70/59/97/97/84/83/191/</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100x100</td>
<td>SSOR ( IC(P_d) )</td>
<td>110/5.38/90/4.42/153/2.54/131/2.72/129/2.69/301/2.32/</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150x150</td>
<td>SSOR ( IC(P_d) )</td>
<td>163/17.6/136/14.0/231/8.6/231/8.7/197/7.9/193/7.8/439/5.56/</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200x200</td>
<td>SSOR ( IC(P_d) )</td>
<td>218/41.3/181/34.5/308/20.3/262/18.4/586/13.1/</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3 - The number of PCCG iterations (I) and the execution time (T) for Problem 1 with \( FD_4 \) and the optimal choice of \( \omega \).

<table>
<thead>
<tr>
<th>grid size</th>
<th>precond. method</th>
<th>one color</th>
<th>global 2-colors</th>
<th>column-wise 2-colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>200x200</td>
<td>SSOR ( IC(P_d, \omega) )</td>
<td>51/10.1/1.95/308/20.3/1.0/253/17.9/1.15/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>53/10.2/1.23/308/20.3/1.0/253/17.9/1.13/</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4 - Results for problem 1 with \( FE_4 \) on a 150x150 grid.

for the SSOR/PCCG and the \( SIC(P_d, \omega)/PCCG \) methods with \( \omega = \omega_{opt} \), the optimal value of \( \omega \) which gave the fastest convergence. In Table 4, we combine the results for \( \omega = 1 \) and \( \omega = \omega_{opt} \) in one table for the \( FE_4 \) discretization.

**Problem 2:** (PROB 29 in [18])

\[
\nabla y \nabla u = 0 \quad \text{on } D
\]

\[
u = u - y \quad \text{on } \partial D
\]

where \( D \) is the unit square. The results of \( FE_4, FE_3 \) and \( FE_4 \) discretizations for this problem are given in Tables 5 and 6. At this point, we would like to note that the running times
reported for this and the previous problems seem high compared to the running times which are reported in other papers (e.g. [24, 25]). This should not be surprising because our implementation is designed for general sparse matrices, and hence does not assume any special form for the stripes. Clearly, an implementation which is designed specifically for matrices generated from rectangular grids would use the fact that all the stripes are parallel to the diagonal of the matrix, thus eliminating the need for the time consuming gather and scatter operations.

<table>
<thead>
<tr>
<th>precond. method</th>
<th>one color</th>
<th>global 4-colors</th>
<th>column-wise 2-colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSOR</td>
<td>136/22.7</td>
<td>1.0</td>
<td>140/8.5</td>
</tr>
<tr>
<td>IC(P_d)</td>
<td>126/21.1</td>
<td>1.0</td>
<td>132/8.1</td>
</tr>
<tr>
<td>IC(P_d)</td>
<td>96/17.4</td>
<td>1.0</td>
<td>109/8.1</td>
</tr>
</tbody>
</table>

| SSOR            | 39/6.8    | 1.95            | 98/5.8               |
| SIC(P_d,\omega) | 39/6.9    | 1.55            | 86/5.2               |
| SIC(P_d,\omega) | 37/7.5    | 1.2             | 71/5.7               |

| no prec.        | I/T = 647/12.8 |

Table 5 - Results for Problem 2 with $FE_4$ on a 150X150 grid

<table>
<thead>
<tr>
<th>precond. method</th>
<th>$FE_3$</th>
<th>$FE_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>one color</td>
<td>global 4-colors</td>
</tr>
<tr>
<td>SSOR</td>
<td>108 1.0</td>
<td>450 1.0</td>
</tr>
<tr>
<td>IC(P_d)</td>
<td>92   1.0</td>
<td>432 1.0</td>
</tr>
<tr>
<td>SIC(P_d,\omega)</td>
<td>35   1.9</td>
<td>135 1.3</td>
</tr>
<tr>
<td>no prec.</td>
<td>32   1.3</td>
<td>122 1.2</td>
</tr>
</tbody>
</table>

Table 6 - Results for Problem 2 with $FE_3$ and $FE_9$ on a 99X99 grid.

Problem 3 (PROB 26 in [18])

\[
\nabla \nabla (1+x^2)^3 \nabla u = 60 x \quad \text{on } D
\]
\[u = 0 \quad \text{on } \partial D \]

where $D$ is the pierced rectangular domain of Figure 3. The result of the $FE_4$ discretization for this problem using a 100X200 pierced rectangular grid is given in Table 7.
| precond. method | one color | | | | column-wise 2-colors |
|-----------------|----------|---|---|---|
| SSOR            | 90/10.8  | 1.0 | 112/6.5 | 1.0 | 106/5.0 | 1.0 |
| IC ($P_d$)      | 69/8.4   | 1.0 | 108/6.4 | 1.0 | 100/4.8 | 1.0 |
| IC ($P_2$)      | 69/9.0   | 1.0 | 113/8.3 | 1.0 | 104/5.8 | 1.0 |

| SSOR            | 29/3.6   | 1.9 | 109/6.2 | 1.2 | 102/4.7 | 1.2 |
| IC ($P_d$,$\omega$) | 29/3.8 | 1.22 | 106/6.1 | 1.15 | 95/4.5 | 1.2 |
| IC ($P_2$,$\omega$) | 30/4.4 | 1.2 | 107/8.1 | 1.25 | 86/4.8 | 1.3 |

Table 7 - Results for Problem 3 with $FE_4$

By studying the above results, we may note the following:

1) For large grid sizes, the column-wise numbering scheme is consistently better than the global numbering scheme. In fact, it seems that the global scheme may be recommended (on the CRAY) only if the grid size is relatively small. More specifically, if the zero stretch for the column-wise scheme is smaller than 64. Clearly, for such small problems, the power of a supercomputer is not needed.

2) If no adaptive procedure is incorporated in the PCCG solver to choose the optimal $\omega$, then the natural choice is $\omega=1$. In this case, it is clear from the results that the use of the IC ($P_d$) preconditioner instead of the SSOR preconditioner reduces the number of PCCG iterations for both the one color scheme and the column-wise coloring scheme. The advantage of the IC ($P_d$) preconditioner is clear, given that the cost of the computation of the IC ($P_d$) matrix is equal to the cost of one PCCG iteration.

3) The IC ($P_d$) preconditioner may reduce the number of iterations over the IC ($P_d$) preconditioner. However, the IC ($P_d$)/PCCG runs usually faster than the IC ($P_d$)/PCCG because it costs too much to compute the IC ($P_d$) factorization. Here, we would like to note that our implementation does not take advantage of the technique of Eisenstat [6], which is described at the end of Section 3, to reduce the work in SSOR/PCCG and IC ($P_d$)/PCCG.

Clearly, the use of this technique will reduce further the execution time of these methods compared to the IC ($P_d$)/PCCG.
4) If a one-color numbering scheme is applied, then it is possible to reduce the number of iterations in the SSOR/PCCG drastically by using the optimum value of $\omega$. This drastic reduction offsets the disadvantage of having a unit zero stretch. In other words, if an adaptive strategy may be used to choose the optimum $\omega$, then the one-color numbering scheme seems to be the most appropriate scheme to apply.

5) For problems that are well conditioned, as for example problem 1, the fastest way to solve the problem is the non-preconditioned CG method. However, this is not the case for problems 2 and 3 which are ill conditioned.

<table>
<thead>
<tr>
<th></th>
<th>one color</th>
<th>col-wise 3-colors</th>
<th>col-wise 2-colors</th>
<th>global 4-colors</th>
<th>global 2-colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of SSOR/PCCG iterations</td>
<td>163</td>
<td>187</td>
<td>197</td>
<td>204</td>
<td>231</td>
</tr>
<tr>
<td>number of IC ($P_d$)/PCCG iterations</td>
<td>136</td>
<td>165</td>
<td>193</td>
<td>199</td>
<td>231</td>
</tr>
<tr>
<td>the zero stretch $\Delta$</td>
<td>1</td>
<td>50</td>
<td>74</td>
<td>5624</td>
<td>11175</td>
</tr>
</tbody>
</table>

Table 8 - The effect of $\Delta$ on the solution of Problem 1 with $FD_3$ discretization on a 150×150 grid ($\omega=1$).

6) There seems to be an inverse relation between the value of the zero stretch and the convergence rate of the PCCG. In order to be more specific, we consider the $FD_3$ discretization of Problem 1, and in addition to the numbering schemes reported in Table 2, we use a 3-color, column-wise scheme, and a 4-color global scheme. The results which we report in Table 8 confirm this inverse relation.

Finally, we would like to note that only rectangular and pierced rectangular domains were chosen in the experiments because, first, we did not have a means for the automatic generation of large irregular grids, and second, the numbering algorithm of Section 4.3 is, to our knowledge, the first algorithm which aims at increasing the zero stretch for general matrices. Hence, its performance may only be compared to an algorithm which produces the maximum zero stretch. We believe that such an algorithm is NP complete.
APPENDIX

Proof of Theorem 1: Let $i$ and $j$ be two nodes in some element $e$ in $\Omega$, and let $l$, $1 \leq l \leq q$, be the color given to $i$ and $l_+$ be the color given to $j$, where $l_+ = l + 1$ if $1 \leq l < q$, and $l = 1$ if $l = q$. Note that either 1) $i$ and $j$ lie in the same column of nodes, $CN_+$, in $\Omega$ or 2) $i$ lies in $CN_+$ and $j$ lies in the following column of nodes, namely $CN_{n+1}$. Let also $CE_+$ be a column of elements in $\Omega$ which contains element $e$, and let $ce_+$ be the number of elements in $CE_+$.

From ALG6 and ALG7, it is clear that the number of nodes which are numbered between nodes $i$ and $j$, namely $\nu(j) - \nu(i) + 1$, is equal to the sum of 1) the number of nodes with color $l$ which are above node $i$ (including $i$) in column $CN_+$, and 2) the number of nodes with color $l_+$ which are below node $j$ (including $j$) in column $CN_+$ or $CN_{n+1}$, whichever applies.

Given that at most two elements in the same column may share a single node, and that for any specific color, every element should contain a node of that color, we may conclude that $\nu(j) - \nu(i) + 1$ should be larger than, or equal to, the sum of 1) half the number of elements in $CE_+$ above, and including, element $e$, and 2) half the number of elements in $CE_+$ below, and including, element $e$. In other words,

$$\nu(j) - \nu(i) + 1 \geq \frac{ce_+ + 1}{2}$$

But $\Delta = \min\{\nu(j) - \nu(i)\}$, where $e$ may be in any column in $\Omega$. Hence

$$\Delta + 1 \geq \frac{ce_{\min} + 1}{2}$$

which gives the lower bound.

In order to establish the upper bound on $\Delta$, we note that $\nu(j) - \nu(i) - 1$ is equal to the sum of nodes of color $l$ above node $i$ and the nodes of color $l_+$ below node $j$ (not including $i$ and $j$). But, each element in $\Omega$ may contain at most one node of any given color. Then

$$\nu(j) - \nu(i) - 1 \leq ce_+ - 1$$
and the upper bound follows directly. □

Proof of Lemma 1: Let $CN_u$ be the $u^{th}$ column of nodes in $\Omega$ and let $c_n$ be the number of nodes in this column. In the global multicolor numbering scheme, the colors $1, ..., \sqrt{p}$ are first used to color the nodes in $CN_{1+\sqrt{p}}, \nu=0,1,2,...$, such that any two neighboring nodes in a column are given different colors. Then, the colors $\sqrt{p}+1, ..., 2\sqrt{p}$ are used to color the nodes in $CN_{2+\sqrt{p}}, \nu=0,1,2,...$, and so on. Thus, if the given color $l$ satisfies $(k-1)\sqrt{p} < l \leq k\sqrt{p}$, for some $k, 1 \leq k \leq \sqrt{p}$, then the number of nodes in $\Omega$ which are given the color $l$, namely $n_{l}^{\Omega}$, is obtained from

$$n_{l}^{\Omega} = \sum_{\nu=0,1,...} c_n^{l}_{k+\nu\sqrt{p}}$$

where $c_n^{l}_{k}$ is the number of nodes which are given the color $l$ in column $CN_u$.

In order to estimate each term in (14), we consider a particular column of nodes $CN_u$, where $u=k + \nu\sqrt{p}$, for some $\nu$, and we let $L=(k-1)\sqrt{p}$. Clearly, the colors used to color the nodes in $CN_u$ are $\lambda=L+1,...,L+\sqrt{p}$. We also define a clique in $CN_u$ as a subset of nodes within $CN_u$ such that any two nodes in this subset are neighbors. Note that the nodes in $CN_u$ which lie on a vertical side of an element form a clique. Let $clq_u$ be the number of such cliques in $CN_u$. Given that any particular node in $CN_u$ may be in at most two cliques, we obtain

$$clq_u \geq 2c_n^{\lambda} \quad \lambda=L+1,...,L+\sqrt{p} \quad (15.\text{a})$$

Also, the coloring scheme is such that any clique may contain at most one node of any particular color. This gives

$$clq_u \leq c_n^{\lambda} \quad \lambda=L+1,...,L+\sqrt{p} \quad (15.\text{b})$$

From (15.\text{a}) and (15.\text{b}), we get, for the given color $l$,

$$c_n^{l} \leq 2c_n^{\lambda} \quad \lambda=L+1,...,L+\sqrt{p}$$

If we write the above inequality $\sqrt{p} - 1$ times with $\lambda=L+1,...,\sqrt{p}$, $\lambda \neq l$, sum these inequalities, and then add $2c_n^{l}$ to both sides of the result, we get
\[(\sqrt{p} + 1) c_n^i \leq 2 \sum_{\lambda=L}^{L+\sqrt{p}} c_n^\lambda = 2 c_n \]  

(16)

Similarly, from (15.a) and (15.b) we may obtain

\[c_n^i \geq \frac{1}{2} c_n^\lambda \quad \lambda = L + 1, \ldots, L + \sqrt{p}\]

which leads to

\[(\sqrt{p} - \frac{1}{2}) c_n^i \geq \frac{1}{2} c_n\]

(17)

Equation (16) and (17) may now be combined into

\[\frac{1}{2\sqrt{p} - 1} c_n \leq c_n^i \leq \frac{2}{\sqrt{p} + 1} c_n\]

which, together with (14) gives

\[\frac{1}{2\sqrt{p} - 1} \sum_{v=0}^{1} c_{n + \sqrt{p}} \leq n^i_{\Omega} \leq \frac{2}{\sqrt{p} + 1} \sum_{v=0}^{1} c_{n + \sqrt{p}}\]

(18)

Finally, we need to relate the sum in (18) to the total number of nodes in \(\Omega\). Instead of dealing with complex formulas, we treat each of the cases \(FE_4\) and \(FE_9\) separately. For \(FE_4\), any column of nodes, \(CN_{\cdot\cdot}\) in \(\Omega\) is enclosed between two consecutive columns of elements. This implies that \(c_n \leq c_{n-1} + c_{n+1}\). Thus

\[\sum_{u \equiv \text{odd}} c_n \leq 2 \sum_{u \equiv \text{even}} c_n\]

(19)

Noting that \(\sum_{u \equiv \text{odd}} c_n + \sum_{u \equiv \text{even}} c_n = n_{\Omega}\), we may use (19) to obtain

\[\frac{1}{3} n_{\Omega} \leq \sum_{u \equiv \text{even}} c_n \leq \frac{2}{3} n_{\Omega}\]

(20)

The same bounds (20) may be obtained for \(\sum_{u \equiv \text{odd}} c_n\), which proves the following for \(FE_4\) and \(p = 4\):

\[\frac{1}{2\sqrt{p} - 1} n_{\Omega} \leq \sum_{v=0}^{1} c_{n + \sqrt{p}} \leq \frac{2}{\sqrt{p} + 1} n_{\Omega}\]

(21)
For FE₉ discretizations, a similar procedure, starting from \( cn_u \leq cn_{u+1} + cn_{u-2} \) and \( cn_u \leq cn_{u-1} + cn_{u+2} \), may be used to prove (21) for the case \( p=9 \). The result (12), then, follows directly from (21) and (18). □

**Proof of Theorem 2:** From the definition of \( \Delta \), there exist two node \( i \) and \( j \) in an element \( e \) such that \( \nu(j) - \nu(i) = \Delta \). Let \( CE_u \) be the column of elements which contains \( e \), and let \( l \) and \( l+1 \) be the colors of nodes \( i \) and \( j \), respectively. The proof proceeds by the construction of two pierced rectangular domains \( \Omega_1 \) and \( \Omega_2 \) which overlap at element \( e \). Namely, \( \Omega_1 \) consists of element \( e \), the elements in \( CN_u \) above \( e \), and the elements in the following columns \( CN_{u+1}, CN_{u+2}, \ldots \). The grid \( \Omega_2 \) consists of element \( e \), the elements in \( CN_u \) below \( e \), and the elements in the previous columns \( CN_1, \ldots, CN_{u-1} \). Let \( n_{\Omega_1} \) and \( n_{\Omega_2} \) be the numbers of nodes in \( \Omega_1 \) and \( \Omega_2 \), respectively, and note that \( n_{\Omega_1} + n_{\Omega_2} = n_{\Omega} + p \).

From the global numbering scheme, it is clear that the difference \( \nu(j) - \nu(i) + 1 \) is equal to the number of nodes in \( \Omega_1 \) which have the color \( l \), namely \( n_{\Omega_1}^l \), plus the number of nodes in \( \Omega_2 \) which have the color \( l + 1 \), namely \( n_{\Omega_2}^{l+1} \). That is

\[
\nu(j) - \nu(i) + 1 = n_{\Omega_1}^l + 1n_{\Omega_2}^{l+1}
\]  

(23)

But, \( \Omega_1 \) and \( \Omega_2 \) are pierced rectangular domains, and thus, \( n_{\Omega_1}^l \) and \( n_{\Omega_2}^{l+1} \) obey the bound of Lemma 1. The application of these bounds on (23) gives

\[
\left| \frac{1}{2\sqrt{p} - 1} \right|^2 [n_{\Omega_1} + n_{\Omega_2}] \leq \nu(j) - \nu(i) + 1 \leq \left| \frac{2}{\sqrt{p} + 1} \right|^2 [n_{\Omega_1} + n_{\Omega_2}].
\]

Finally, (13) results by using the definition of \( \Delta \) and \( n_{\Omega} = n_{\Omega_1} + n_{\Omega_2} - p \). □
References


