PRECONDITIONING STRATEGIES FOR SOLVING ELLIPTIC DIFFERENCE EQUATIONS

1982  C K TAFT

UNCLASSIFIED

UNCLASSIFIED  AFIT/NE-82-27
Preconditioning Strategies for Solving Elliptic Difference Equations on a Multiprocessor

Charles Kirkland Taft, Jr.

AFIT STUDENT AT: University of Illinois

AFIT/NR
WPAFB OH 45433

1982

22 JUN 1982

AIR FORCE INSTITUTE OF TECHNOLOGY
WRIGHT-PATTERSON AFB, OH 45433

ATTACHED

82 07 07 064

S E
Abstract

PRECONDITIONING STRATEGIES FOR SOLVING ELLIPTIC
DIFFERENCE EQUATIONS ON A MULTIPROCESSOR

Charles Kirkland Taft, Jr.
Captain, United States Air Force
Master of Science
Department of Computer Science
University of Illinois
157 pages

This thesis deals with choosing preconditioning strategies to accelerate a conjugate gradient algorithm for solving elliptic difference equations, suitable for implementation on a multiprocessor. The hypothetical multiprocessor considered consists of p linearly connected processors. A variety of popular preconditioning strategies for sequential machines are examined. Numerical experiments are conducted and recommendations made.
PRECONDITIONING STRATEGIES FOR SOLVING ELLIPTIC DIFFERENCE EQUATIONS ON A MULTIPROCESSOR

BY

CHARLES KIRKLAND TAFT, JR.

B. S., University of New Hampshire, 1976

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 1982

Urbana, Illinois
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

January 1982

WE HEREBY RECOMMEND THAT THE THESIS BY

CHARLES KIRKLAND TAFT, JR.

ENTITLED... PRECONDITIONING STRATEGIES FOR SOLVING

ELLIPITC DIFFERENCE EQUATIONS ON A MULTIPROCESSOR

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF... MASTER OF SCIENCE

Ahmed Lamki
Director of Thesis Research

J. W. Reade
Head of Department

Committee on Final Examination†

Chairman

† Required for doctor's degree but not for master's.
University of Illinois at Urbana-Champaign

DEPARTMENTAL FORMAT APPROVAL

THIS IS TO CERTIFY THAT THE CONTENT, FORMAT, AND QUALITY OF PRESENTATION OF THE THESIS SUBMITTED BY CHARLES KIRKLAND TAFT, JR. AS ONE OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IS ACCEPTABLE TO THE DEPARTMENT OF COMPUTER SCIENCE

Department/Division/Unit

January 11, 1982

Date of Approval

Departmental Representative
Acknowledgements

I wish to thank the United States Air Force for giving me this opportunity to return to school for my masters degree. I am particularly indebted to my thesis advisor, Dr. Ahmed Sameh, for his help in selecting my thesis topic and his guidance and editorial comments during its preparation. Special thanks to Barbara Armstrong for her patient explanations of the UNIX system. I would like to acknowledge the financial support from the National Science Foundation under Grant MCS 79-18394. Finally, I wish to thank my wife Nancy, without whose patience, endless proofreading and constant encouragement, none of this would have been possible.
Table of Contents

1 Introduction .................................................... 1
2 Model Problem .................................................. 3
3 Background ........................................................ 4
   3.1 Conjugate Gradient Method .................................. 4
   3.2 Preconditioning ............................................. 6
   3.3 Preconditioned Conjugate Gradient Method ............... 10
4 Investigative Process .............................................. 18
   4.1 Introduction ............................................... 18
   4.2 Software ................................................... 19
   4.3 Preconditioning Strategies ................................ 21
   4.4 Phase I .................................................... 25
      4.4.1 Introduction ........................................... 25
      4.4.2 Software ............................................... 26
      4.4.3 Results ................................................ 28
   4.5 Phase II ..................................................... 42
      4.5.1 Introduction ............................................. 42
      4.5.2 Results ................................................ 45
   4.6 Phase III ..................................................... 52
      4.6.1 Introduction ............................................. 52
      4.6.2 Software ............................................... 54
      4.6.3 Results ................................................ 55
5 Conclusions ..................................................... 66

Appendices

A Grid point ordering schemes .................................... 68
B Matrix block structures .......................................... 75
C User input parameters ........................................... 81
D Definition of test problems ...................................... 87
E Cost of Conjugate Gradient Algorithm .......................... 91
F Program Listings ................................................ 94
References ...................................................... 156
1. Introduction

My thesis deals with solving systems of linear equations

\[ Ax = b, \quad (1.1) \]

where \( A \) is a sparse symmetric and positive definite matrix. Systems of this type arise from the discretization of second order self-adjoint elliptic partial differential equations. Many direct and iterative numerical methods have been developed for solving this problem; see for example [Varg62], [Wach66], [Youn71], [HaYo81] and [Birk81]. The advent of multiprocessor systems brings with it the possibility of substantial speedup in performing these types of numerical methods. This would allow us to examine problems that, until now, had been too large or complex to be computationally feasible. The new multiprocessor systems will require that new numerical methods be generated or that older methods be modified to take full advantage of their potential.

In this paper I consider only the conjugate gradient method and preconditioning strategies that are best suited for implementation on a multiprocessor system. The hypothetical multiprocessor that will be considered consists of \( p \) linearly connected processors as shown in figure 1.1.

![Diagram of linearly connected processors](image1.1)

Figure 1.1
Each processor is assumed to be capable of performing any arithmetic operation in one time step, and that it takes $p$ time steps to transfer one floating point number from one processor to either of its neighbors.

For sequential machines, the problem of preconditioning the conjugate gradient algorithm has been extensively studied in the literature. See for example [AxGu80], [CoGo76], [Eise81], [Gust78], [HaYo81], [Kers78], [Mant80], [MeVo77], [Munk80], [Reid71], and [Reid72].
2. Model Problem

Consider the second order self-adjoint partial differential equations of the form

\[ -\frac{\partial}{\partial x}\left[a(x,y)\frac{\partial u}{\partial x}\right] - \frac{\partial}{\partial y}\left[c(x,y)\frac{\partial u}{\partial y}\right] + f(x,y)u = g(x,y) \quad (2.1) \]

with \(a(x,y) > 0, c(x,y) > 0\) and \(f(x,y) > 0\); defined on the unit square, \(0 < x,y < 1\); and with boundary conditions of the form

\[ au + \frac{\partial u}{\partial n} = \gamma \quad (2.2) \]

where \(\frac{\partial u}{\partial n}\) is the derivative normal to the boundary.

Superimposing a square grid of mesh size \(h = 1/(n+1)\) and using central difference approximations to the derivatives, the problem converts to solving a linear system of equations of order \(n^2\). This process is fully derived and explained in [Varg62]. Handling boundary conditions of the form (2.2) where \(\beta \neq 0\) is discussed in [MiGr80].

Under certain boundary conditions, the resulting coefficient matrix \(A\) is a positive definite M-matrix [Vors81]. An M-matrix is defined such that given matrix \(A = (a_{ij})\),

1) \(a_{ii} > 0\) 2) \(a_{i,j} < 0\)

3) \(A^{-1}\) exists 4) \(A^{-1} > 0\).

In Appendix D, I will describe the nature of matrix \(A\) for each of my test problems. The structure of matrix \(A\) is determined by the grid point ordering scheme. Appendix A shows examples of the natural, point red/black, line red/black and 2 line red/black ordering schemes and the resulting structure of the matrix \(A\) for \(n=6\).
3. Background

3.1. Conjugate Gradient Method

The Conjugate Gradient (CG) Method was developed by Hestenes and Stiefel in 1952. The idea behind it is to approximate the solution vector $x$ by

$$x^{(m)} = x^{(0)} + \sum_{j=1}^{m} \alpha_j v_j$$

where $x^{(0)}$ is an arbitrary initial guess, the vectors $v_j$ are A-conjugate (i.e. $v_j^T A v_i = 0$ for $j \neq i$) and the $\alpha_j$'s are chosen to minimize $\|x^{(m)} - x\|_A$

where $\|z\|_A = (z, Az)^{1/2}$.

The vectors $v_{j+1}$ are constructed by orthogonalizing the residual $r_j = b - Ax^{(j)}$ with respect to $v_j$, i.e. $r_j^T v_i = 0$ for $j > i$. In this way, each iteration is attempting to minimize the components of the residual along the eigenvector corresponding to the most extreme eigenvalue. The residual then lies almost entirely in the subspace of the eigenvectors with the remaining less extreme eigenvalues. The iteration proceeds as if the most extreme eigenvectors and eigenvalues were not present [Kers78].

In the absence of round-off errors, the CG method can be considered a direct method, in that it will converge to the true solution of a system of order $n$ in exactly $n$ steps, due to the orthogonality of the
vectors \( v_j \). In fact, if the \( n \times n \) matrix \( A \) has only \( r \) distinct eigenvalues, then the method converges in only \( r \) steps. Many times, the relative error \( \|x^{(i)} - x\|/\|x\| \) will be quite small even for \( i \ll n \). Unfortunately, in the presence of round-off errors, the orthogonality of the vectors \( v_i \) can break down and the guaranteed finite convergence is lost. It was this breakdown that prevented the CG method from getting much attention. It wasn't until 1971 that interest was renewed in the CG method. At that time, Reid [Reid71] showed that the CG algorithm is very effective for handling large and sparse positive definite linear systems as arise from our model problem. Its cause was further helped when Concus, Golub and O'Leary [CoGO76] showed that it could be used as an effective tool for accelerating the convergence of various iterative methods. They pointed out that the CG method possesses some very attractive properties:

1) doesn't require prior knowledge of extreme eigenvalues to calculate optimal convergence parameters

2) takes advantage of the entire distribution of eigenvalues of matrix \( A \)

3) is optimal in the class of all algorithms for which
\[
x^{(k+1)} = x^{(0)} + P_k(\xi) \tau^{(0)}
\]
where \( \xi = I - M^{-1}N \), \( M = A - N \) is a regular splitting and \( P_k(\xi) \) is a polynomial of degree \( k \), in the sense that it minimizes \( \|x_{k+1} - x\|_A \).

See [CoGO76] for more details.
The rate of convergence of the CG algorithm depends heavily on the distribution of eigenvalues of matrix A. The fewer distinct eigenvalues or the more clustered the eigenvalues, the quicker the convergence. Unfortunately, the matrices arising from our model problem tend to have eigenvalue distributions that are widely distributed with little clustering. As a result, the CG algorithm by itself tends to do poorly. This situation can be improved by "preconditioning" matrix A.

3.2. Preconditioning

The idea behind preconditioning is to obtain a matrix C such that C is positive definite and C⁻¹A has a "better" eigenvalue distribution. It is also important to choose matrix C such that solving a system Cw = q is as easy as possible. The CG algorithm is then applied to the new preconditioned system

C⁻¹Ax = C⁻¹b.

This notation has one problem in that C⁻¹A may no longer be symmetric. It is better to consider the preconditioned system

(C⁻¹/²AC⁻¹/²)(C¹/²x) = C⁻¹/²b, or

(L⁻¹AL⁻T)(L¹x) = L⁻¹b,

where C = LLᵀ.

Obviously the best eigenvalue distribution for C⁻¹A would be achieved when C = A, then C⁻¹A = I. This does not help us much, however, in that solving a system Cw = q is no easier than solving the
original system. The idea then is to choose matrix $C$ as close as possible to $A$, such that $C^{-1}A$ would have a few extreme eigenvalues with the rest clustered around unity, while still requiring $Cw = q$ be easy to solve.

When matrix $A$ is an $M$-matrix, Meijerink and van der Vorst [MeVo77] introduced a set of preconditioning strategies based on an incomplete factorization of matrix $A$. The idea is to choose $C = LU$, such that matrix $C$ resembles matrix $A$, $A = C - R$, with $L$ and $U$ almost as sparse as matrix $A$. The sparsity of $L$ and $U$ is controlled by forcing certain predetermined positions within $L$ and $U$ to be zero. These positions are defined by a set $P$ of places $(i,j)$ such that

$$P \subseteq P_n \equiv \{ (i,j) \mid i \neq j \land 1 \leq i \leq n, 1 \leq j \leq n \}$$

where $P_n$ contains all pairs of indices of off-diagonal matrix elements.

When matrix $A$ is symmetric, we add the restriction to the set $P$ that if $(i,j) \in P$ then so must $(j,i) \in P$ and consider an incomplete Cholesky factorization ($LL^T$ or $LDL^T$). Meijerink and van der Vorst proved that if matrix $A$ is an $M$-matrix, then this process is stable and the resulting factorization

$$C = LL^T \text{ or } LDL^T$$

is positive definite.

Using this set $P$ notation, we can describe most of the basic preconditioning strategies. On the extremes, we have $P = P_n$ and $P = \phi$. 
which result in diagonal scaling, \( C = \text{diag}(A) \) and preconditioning by complete Cholesky factorization, \( C = A \), respectively. In between we have

\[
P^* = \{ (i,j) \mid A(i,j) = 0 \}
\]

which is the preconditioning strategy used by the ICCG(0) algorithm of Meijerink and van der Vorst [MeVo77].

When Matrix A is positive definite, but not an M-matrix, non-positive or small diagonal elements can result during the factorization process, causing matrix C to be no longer positive definite. A number of modifications have been proposed to solve this problem. Kershaw [Kers78] recommends simply replacing the non-positive diagonal elements by suitable positive numbers. He has found that a few diagonal elements can become non-positive and be so replaced without distracting from the incomplete factorization, as long as most of the pivots remain positive. Another approach is simply to add \( \alpha D \) to matrix A before attempting the incomplete factorization, where \( D = \text{diag}(A) \) and \( \alpha \) is a positive scalar. This idea was proposed by Manteuffel [Mant80] in developing his shifted incomplete Cholesky factorization. If \( \alpha \) is large enough, then the factorization is guaranteed to be positive definite. However, choosing \( \alpha \) too large results in very slow convergence of the resulting conjugate gradient algorithm. Unfortunately, the only way to determine a "good" value of \( \alpha \) for a given problem is through trial and error. For the test problems considered by Manteuffel, good results were achieved for \( \alpha \) of \( 0(10^{-2}) \).
A number of variations on the incomplete factorization idea of Meijerink and van der Vorst have been proposed. Gustafsson [Gust78] introduced the concept of the modified incomplete factorization. Here the elements created during the incomplete factorization that correspond to entries in the set P are added to the diagonal elements of matrix C prior to being discarded. The process is known as diagonal modification. The MICCG(0) algorithm results when adding diagonal modification to the ICCG(0) algorithm. Gustafsson reports that a faster asymptotic rate of convergence can be achieved.

Another variation has been proposed by Munksgaard [Munk80]. Here, instead of dropping a predetermined set of elements P during the factorization, he proposes developing criteria for dropping only the "smaller" fill-ins while retaining the "larger" ones. The philosophy here is that the number of iterations required to reach a solution is more sensitive to the size of the elements dropped than to the number dropped. He suggests dropping fill-in elements if their numeric value relative to the diagonal elements of their row and column is less than a relative drop tolerance. In the kth pivot step we drop \( l_{1j}^{(k+1)} \) if

\[
|l_{1j}^{(k+1)}| < c(d_{11}^{(k)})^{1/2} d_{jj}^{(k)}.
\]

The amount of fill-in is determined by the size of c. If c is close to zero, we obtain almost a complete factorization, while c = 1 produces a factorization where no fill-ins are added and L has the same sparsity pattern as matrix A.
3.3. Preconditioned Conjugate Gradient Method

Given a preconditioning matrix $C$ and an initial guess $x_0$, the standard preconditioned conjugate gradient (PCG) method can be described in the following algorithmic format:

Algorithm 3.1

a) Initial step

1) $r_0 = b - Ax_0$
2) $z_0 = C^{-1}r_0$
3) $p_0 = z_0$

b) For $k = 0, 1, \ldots$

1) $\alpha_k = (r_k, z_k)/(p_k, Ap_k)$
2) $x_{k+1} = x_k + \alpha_k p_k$
3) $r_{k+1} = r_k - \alpha_k Ap_k$
4) $z_{k+1} = C^{-1}r_{k+1}$
5) $\beta_k = (r_{k+1}, z_{k+1})/(r_k, z_k)$
6) $p_{k+1} = z_{k+1} + \beta_k p_k$

A commonly used stopping criterion for this algorithm is to calculate $||r_k|| = (r_k, r_k)^{1/2}$ each iteration and stop when $||r_k|| < \varepsilon$, where $\varepsilon$
is a user specified parameter.

One choice for the initial iterate $x_0$ is a random vector. A more creative approach is to choose $x_0 = C^{-1}b$. This uses the fact that if matrix $C$ is close to matrix $A$, then $x_0 = C^{-1}b$ will be a reasonably accurate estimate for $x = A^{-1}b$. Starting with a more accurate estimate for $x$ will hopefully reduce the number of iterations required to generate an answer of desired accuracy.

The ICCG(0) and MICCG(0) algorithms utilize incomplete LDLT factorization of the form

$$C = (\tilde{D} + L)\tilde{D}^{-1}(\tilde{D} + L)^T,$$

where $A \equiv L + D + L^T$, $L$ is strictly lower triangular and $D$ and $\tilde{D}$ are positive diagonal matrices. To define $\tilde{D}$, I will use figure A5 and use $a_i$, $b_i$ and $c_i$ to denote the elements of the main diagonal, upper-diagonal and $m^{th}$ upper diagonal respectively, where $i$ is the row index and $m$ is the half band width of the matrix. Then $\tilde{D} = \text{diag}(\tilde{d}_1, \ldots, \tilde{d}_n)$ is defined for ICCG(0) as:

$$\tilde{d}_i = a_i - b_{i-1}^2 \tilde{d}_{i-1}^{-1} - c_{i-m}^2 \tilde{d}_{i-m}^{-1},$$

$$i = 1, 2, \ldots, n$$

and for MICCG(0) as:

$$\tilde{d}_i = a_i - b_{i-1}^2 \tilde{d}_{i-1}^{-1} - c_{i-m}^2 \tilde{d}_{i-m}^{-1} - r_i - r_{i-m+1},$$
$$r_i = c_{i-1}b_{i-1} \tilde{d}_{i-1}^{-1},$$

$$i = 1, 2, \ldots, n$$
where in both cases, elements not defined (i.e. subscripts \( \leq 0 \)) should be replaced by zeroes. For those algorithms where the incomplete LDL\(^T\) factorization can be described in the form (3.1), Eisenstat [Eise81] proposes a different implementation of our standard PCG Algorithm 3.1. His method reduces the number of multiply-adds required per iteration by a factor approaching one half. This is done by restating the original problem (1.1) in the form

\[
[(\tilde{\Omega} + L)^{-1}A(\tilde{\Omega} + L)^{-T}][(\tilde{\Omega} + L)^T x] = [(\tilde{\Omega} + L)^{-1}b] 
\]

or

\[
\hat{A}\hat{x} = \hat{b}. \tag{3.2}
\]

It can then be shown that applying PCG to (1.1) with preconditioning (3.1) is equivalent to applying PCG to (3.2) with preconditioning \( C = \tilde{\Omega}^{-1} \) and setting \( x = (\tilde{\Omega} + L)^{-T}\hat{x} \). The algorithm can now be written as:

Algorithm 3.2

a) Initial step

1) \( \hat{r}_0 = (\tilde{\Omega} + L)^{-1}(b - \hat{A}x_0) \)

2) \( \hat{p}_0 = \hat{z}_0 = \tilde{\Omega}\hat{r}_0 \)

b) For \( k = 0, 1, \ldots \)

1) \( \hat{\alpha}_k = (\hat{r}_k, \hat{z}_k)/(\hat{p}_k, \hat{\beta}_k) \)
2) \[ x_{k+1} = x_k + \hat{a}_k(\bar{D} + L)^{-T}\hat{p}_k \]

3) \[ \hat{r}_{k+1} = \hat{r}_k - \hat{a}_k\hat{p}_k \]

4) \[ \hat{z}_{k+1} = \bar{D}\hat{r}_{k+1} \]

5) \[ \hat{\beta}_k = \frac{(\hat{r}_{k+1}, \hat{z}_{k+1})}{(\hat{r}_k, \hat{z}_k)} \]

6) \[ \hat{p}_{k+1} = \hat{z}_{k+1} + \hat{\beta}_k\hat{p}_k \]

To calculate \( \hat{\beta}_k \), the matrix \( \hat{a} \) does not have to be explicitly calculated. The product can be computed efficiently by taking advantage of the following identity:

\[ \hat{\beta}_k = (\bar{D} + L)^{-1}[\bar{D} + L] + (\bar{D} + L)^T - (2\bar{D} - D)](\bar{D} + L)^{-T}\hat{p}_k \]

This can be simplified, and results in the following two step calculation:

\[ \hat{r}_k = (\bar{D} + L)^{-T}\hat{p}_k \]

\[ \hat{\beta}_k = \hat{r}_k + (\bar{D} + L)^{-1}(\hat{p}_k - KE_k), \]

where \( K \equiv 2\bar{D} - D \).

This version requires \( 8N + NZ(A) \) multiply-adds, versus \( 6N + 2NZ(A) \) for Algorithm 3.1, where \( NZ(A) \) = number of non-zero elements in matrix \( A \).

Another \( 3N \) multiply-adds can be saved by symmetrically scaling the problem so that \( \bar{D} = I \), [Eise81].
Rutishauser considered a version of the PCG algorithm, where $x_i$ and $r_i$ are calculated using a 3-term recurrence relation. It can be represented in the following algorithmic format:

Algorithm 3.3

a) Initial step

1) choose initial guess $x_0$

2) $x_{-1} = 0$

3) $\omega_1 = 1$

4) $r_0 = b - Ax_0$

5) $z_0 = C^{-1}r_0$

b) For $k = 0, 1, \ldots$

1) $\alpha_k = (z_k, r_k)/(z_k, Cz_k)$

2) $\omega_{k+1} = 1/[1 - \frac{\alpha_k}{\alpha_{k-1}} \frac{(z_{k-1}, r_{k-1})}{(z_k, r_k)} \omega_k^{-1}] \quad (k>1)$

3) $x_{k+1} = x_{k-1} + \omega_{k+1}(\alpha_k z_k + x_k - x_{k-1})$

4) $r_{k+1} = r_{k-1} + \omega_{k+1}(-\alpha_k A z_k + r_k - r_{k-1})$
This version is particularly useful when considering the conjugate gradient method as a means of accelerating other iterative methods, as in [CoGO76]. In general, Reid [Reid72] showed that this version required more storage to implement than does our standard PCG algorithm 3.1.

When matrix $A$ possesses "Property A", Reid [Reid72] showed how algorithm 3.3 could be modified to reduce the amount of work per iteration by approximately one half. In general, the same results can be obtained if our problem (1.1) can be partitioned such that:

$$\begin{bmatrix}
    C_1 & F \\
    F^T & C_2
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2
\end{bmatrix} =
\begin{bmatrix}
    b_1 \\
    b_2
\end{bmatrix}. \tag{3.3}
$$

This can also be represented by the two matrix equations:

$$C_1 x_1 = b_1 - F x_2 \tag{3.4}$$
$$C_2 x_2 = b_2 - F^T x_1. \tag{3.5}$$

The idea behind Reid's modification is to choose an initial guess $x_1^{(0)}$ and then use it to calculate $x_2^{(0)}$ via (3.5). This then implies that $z_2^{(0)}$ and forces $\alpha_0 = 1$, where I assume $z$ is partitioned in the same fashion as (3.3). A simple inductive argument shows that for $j = 0, 1, 2, \ldots$

$$\alpha_j = 1 \text{ and } z_1^{(2j+1)} = z_2^{(2j)} = 0.$$
As a result, algorithm 3.3 can be reduced to:

Algorithm 3.4

a) Initial step

1) choose initial guess $x^{(0)}_1$

2) $\omega_1 = 1$

3) $x^{(0)}_2 = C_2^{-1}(b_2 - F^T x^{(0)}_1)$

4) $r^{(0)}_1 = (b_1 - F x^{(0)}_2) - C_1 x^{(0)}_1$

5) $z^{(0)}_1 = C_1^{-1} r^{(0)}_1$

6) $\theta^{(0)}_1 = (z^{(0)}_1, r^{(0)}_1)$

b) For $k = 0, 1, 2, \ldots$

1) $r^{(2k+1)}_2 = (1 - \omega_{2k+1}) r^{(2k-1)}_2 - \omega_{2k+1} F^T z^{(2k)}_1$

2) $z^{(2k+1)}_2 = C_2^{-1} r^{(2k+1)}_2$

3) $\theta^{(2k+1)}_2 = (z^{(2k+1)}_2, r^{(2k+1)}_2)$

4) $\omega_{2k+2} = [1 - \omega_{2k+1} \theta^{(2k+1)}_2 / \theta^{(2k)}_1]^{-1}$
\[ \Delta x_1^{(2k)} = \omega_{2k+1} \omega_{2k+2} [z_1^{(2k)} + (1 - \omega_{2k}) (1 - \omega_{2k+1}) \Delta x_1^{(2k-2)}] \]

6) \[ x_1^{(2k+2)} = x_1^{(2k)} + \Delta x_1^{(2k)} \]

7) \[ r_1^{(2k+2)} = (1 - \omega_{2k+2}) r_1^{(2k)} - \omega_{2k+2} z_2^{(2k+1)} \]

8) \[ z_1^{(2k+2)} = c_1^{-1} r_1^{(2k+2)} \]

9) \[ \theta_1^{(2k+2)} = (z_1^{(2k+2)}, r_1^{(2k+2)}) \]

10) \[ \omega_{2k+3} = [1 - \omega_{2k+2} \theta_1^{(2k+2)} / \theta_2^{(2k+1)}]^{-1} \]

c) Once \( x_1^{(m)} \) has been obtained to the desired accuracy, calculate \( x_2^{(m)} \) using:

\[ x_2^{(m)} = c_2^{-1} (b_2 - F^T x_1^{(m)}). \]

The main advantages with this approach are that the algorithm is working with 1/2 the number of unknowns each iteration, and that each iteration of Algorithm 3.4 is equivalent to two iterations of Algorithm 3.3.
4. Investigative Process

4.1. Introduction

The investigation will be divided into three phases. In the first phase I examine a group of preconditioning strategies arising from the ideas of Meijerink and van der Vorst, Gustafsson, and Munksgaard, and determine how they compare to one another on a given set of problems. The preconditioning strategies will be judged on how they influence the eigenvalue distribution of our test matrices, and their effect on the rate of convergence and amount of work required by a standard CG algorithm to obtain a given relative error.

The second phase consists of analyzing each preconditioning strategy and determining which ones might be easily adaptable to our multiprocessor system. A prime consideration is to identify those preconditioning strategies that minimize the total amount of work, including the amount of interprocessor communication required to construct the preconditioning matrix C and to solve the systems \( z = C^{-1} r \).

From the results of the first two phases, I will narrow the list of possible strategies to two or three prime candidates for preconditioning on multiprocessors. The third phase then consists of analyzing the effects of these strategies on larger and more complex test problems. I will also examine what effect various values of \( \psi \), the interprocessor
communications cost parameter, might have on our choice of a preconditioning strategy. The numerical experiments required during Phase I and Phase III will be conducted on the CDC-Cyber 175 at the University of Illinois, for which the arithmetic precision is roughly 14 decimal digits.

4.2. Software

In conducting the numerical experiments, I relied heavily on the Harwell sparse matrix routines MA31 and EA14A. The MA31 package served as the basis for the incomplete factorization and conjugate gradient routines. A complete description of these routines can be found in [Munk80]. The program listings and on-line write-ups for the MA31 package are available in the Cyber Harwell library under the name MA31A.

The conjugate gradient routine MA31F, contained in this package, was slightly modified. Originally, it chose as its initial guess

\[ x_0 = C^{-1}b. \]

In order to make it more difficult for the algorithm and to get a better idea of how the preconditioning would effect convergence, I replaced \( b \) by a vector with random entries between 0 and 2.

The eigenvalues of our symmetrically preconditioned matrices were found using a Lanczos algorithm as implemented in the Harwell routine EA14A. This algorithm finds the eigenvalues without regard to their multiplicity. A complete description of this routine can be found in
[PaRe81]. The only modification made was to replace the Harwell random number function FA01AS by the CDC Fortran function RANF. The complete program listings and write-ups for this routine should be available shortly in the Cyber Harwell library.

This routine requires that the user supply the necessary code to calculate \( u = u + Av \) each iteration, where the subroutine EA14A supplies the vectors \( u \) and \( v \). Since we are working with a symmetrically preconditioned matrix \( A \), we actually need to calculate

\[
 u = u + L^{-T}A^{-1}L^{-1}v \quad (4.1)
\]

where \( C = LL^T \).

This was done using the Harwell subroutines MA31G and MA31H. The subroutine MA31G solves the system

\[
 x = (L^T)^{-1}y
\]

using backward and forward substitution. I broke this into two separate subroutines; MA31G1 to do the backward substitution, and MA31G2 to do the forward substitution. The subroutine MA31H is used to calculate \( Ax = y \). Using these three routines, we can solve equation (4.1) in the following four steps:

1) solve \( t_1 = L^{-1}v \) using MA31G2

2) calculate \( t_2 = At_1 \) using MA31H
3) solve $t_3 = L^{-T}t_2$ using MA31G1

4) calculate $u = u + t_3$.

Appendix F contains source listings for the programs I created, and those Harwell routines which I modified.

4.3. Preconditioning Strategies

The following is a list of abbreviations and descriptions of the preconditioning strategies that I have examined.

1) DS - Diagonal Scaling
   This method uses $C = \text{diag}(A)$ as its preconditioning matrix.

2) BDS - Block Diagonal Scaling
   Similar to diagonal scaling, this method uses $C = \text{block diag}(A)$, where each principle submatrix is tri-diagonal.

3) IC(s) - Incomplete Cholesky factorization with $s$ diagonals added
   This technique was developed by Meijerink and van der Vorst [MeVo77]. It is normally associated with matrices generated using the natural grid point ordering scheme. The case when no fill-ins are kept during the factorization ($s=0$), can easily be generalized for matrices using other grid point ordering schemes. Here I will limit myself to the cases $s = 0, 1$ and $3$. They utilize set $P$'s of
the form:

\[ p^0 = \{(i,j) \mid A(i,j) = 0 \} \]
\[ p^1 = \{(i,j) \mid |i-j| \neq 0, 1, m-1, m \} \]
\[ p^3 = \{(i,j) \mid |i-j| \neq 0, 1, 2, m-2, m-1, m \} \]

where \( m \) is the half band width of the outer diagonal.

4) MIC(s) - Modified Incomplete Cholesky factorization with \( s \) diagonals added
   Developed by Gustafsson [Gust78], it represents an extension of the IC(s) algorithm to include diagonal modification.

5) HARWELL(c) - Harwell package MA31 with drop tolerance \( c \)
   This performs the incomplete Cholesky factorization as proposed by Munksgaard [Munk80] and implemented by the Harwell routine MA31C. It uses a numeric drop tolerance to control fill-ins, and includes diagonal modification. It also incorporates minimum degree pivoting to minimize the number of potential fill-ins generated. I will limit myself to the two cases \( c=0 \) and \( c=10^{-2} \). The case \( c=0 \) generates a complete Cholesky factorization.

6) MICD(c) - Modified Incomplete Cholesky factorization with Drop tolerance \( c \)
   Similar to the HARWELL(c) algorithm, in this case the minimum degree pivoting has been eliminated.
7) RBIC(s) - Reduced Block Incomplete Cholesky factorization with s diagonals added

This is similar to the IC(s) algorithm, except that only portions of matrix A are used to calculate the incomplete Cholesky factorization. Parts of matrix A are ignored in order to break matrix A into n/2 uncoupled systems of equations. The incomplete Cholesky factorization on each system can then be performed independently. Using the notation in Appendix B, the following is how the matrices arising from the various grid point ordering schemes will be partitioned:

a) Point Red/Black Ordering (Figure B4)

The elements in blocks $E_1^T$, $E_1^T$, $F_1$ and $F_1^T$ ($i = 1, \ldots, (n/2 - 1)$) will be ignored during the factorization.

b) Line Red/Black Ordering (Figure B2)

The elements in blocks $E_{2i}$ ($i = 1, \ldots, (n/2 - 1)$) will be ignored.

c) 2 Line Red/Black Ordering (Figure B3)

The elements in blocks $E_{2i}$ ($i = 1, \ldots, (n/2 - 1)$) will be ignored.

I will limit myself to the case $s=0$, except when working with the 2 Line Red/Black matrices, where I will also examine the case $s=3$. 

Each of these preconditioning strategies was not necessarily matched with each of the grid point ordering schemes. Table 4.1 shows which combinations were examined.

<table>
<thead>
<tr>
<th>Preconditioning Strategy</th>
<th>Grid Point Ordering Schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Natural</td>
</tr>
<tr>
<td>DS</td>
<td>X</td>
</tr>
<tr>
<td>BDS</td>
<td>X</td>
</tr>
<tr>
<td>IC(0)</td>
<td>X</td>
</tr>
<tr>
<td>IC(1)</td>
<td>X</td>
</tr>
<tr>
<td>IC(3)</td>
<td>X</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>X</td>
</tr>
<tr>
<td>MICD(10^{-2})</td>
<td>X</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>X</td>
</tr>
<tr>
<td>HARWELL(10^{-2})</td>
<td>X</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>X</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>X</td>
</tr>
<tr>
<td>RBIC(3)</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 4.1
4.4. Phase I

4.4.1. Introduction

During this phase, I was interested in determining how the various chosen preconditioning strategies compare to one another. I limited myself to comparing them relative to test matrices of order 64 arising from test problem 1 with n=8 (see appendix D). Appendix C outlines which combinations of preconditioning strategies and grid point ordering schemes I looked at.

A prime consideration when choosing an algorithm for this type of problem is the amount of work required to generate an acceptable answer. Keeping this in mind, I determined the amount of time and number of iterations required by our PCG algorithm to produce an answer such that

$$\|r_i\| < 10^{-6}$$

where $r_i = Ax_i - b$.

This was subdivided into the time required to compute the preconditioning matrix and that required to actually perform the PCG iterations.

Another means of comparing preconditioning strategies is to examine their effect on the eigenvalue distribution of the test matrices. Ideally, the eigenvalues of the symmetrically preconditioned test matrices should be clustered around one. In an effort to gauge this, I used the Harwell routine EA14A to calculate all the distinct eigenvalues
(λ₁) of the symmetrically preconditioned matrices to an accuracy of $10^{-4}$. I then calculated the range, mean, and standard deviation of $(λ₁ - 1.0)$. The more successful the strategy, the closer these values will be to zero.

The conclusions reached during this phase are not necessarily intended to hold for larger and more complex problems. A much wider variety and size of test problems would have to have been considered. Such a comprehensive study is beyond the scope of this paper. More exhaustive studies comparing various subsets of these preconditioning strategies with respect to sequential machines only can be found in [MeVo77], [Gust78], and [Munk80].

4.4.2. Software

A modified version of the Harwell incomplete factorization routine MA31C will be used to generate all the various types of factorizations required during this phase. As written, it performed the incomplete factorization using a numeric drop tolerance, diagonal modification, and minimum degree pivoting. To allow the routine to handle a wider variety of factorizations, I made the minimum degree pivoting and diagonal modification user controlled options. I also allowed the user to choose either a numeric drop tolerance or a user defined function FILL to control fill-ins. The function FILL would decide if a zero should be
destroyed by considering only its coordinates, and would be similar in nature to the set $P$ of Meijerink and van der Vorst [MeVo77].

The routine MA31A, used to activate MA31C, was also changed. It had been used to prepare the data structures required during the incomplete factorization. Its duties were taken over by my routine FACTOR. Eliminated was the automatic diagonal scaling of matrix $A$. This necessitated a change to another Harwell routine MA31H, used to calculate $Ax = y$. No longer does this routine assume $\text{Diag}(A) = I$. I also added to FACTOR an option to allow the user to specify which portions of matrix $A$ would be used in calculating the incomplete factorization. This was done using a user defined function EUSE which, when activated, identifies which portion of matrix $A$ is to be passed on to subroutine MA31C.

It should be noted that, while these modifications do allow a greater variety of preconditioning strategies to be implemented, the process at times is far from efficient. As a result, the time required to perform some of the incomplete factorizations will be inflated. This is especially true for the IC(s) and RBIC(s) factorizations. Normally, the locations of the non-zero entries in the factorization are known beforehand, and only those values need be calculated. Here, most of the work required to generate a potential fill-in is done before the program decides to keep it or not. This results in more values being calculated than need be.
The execution time of the factorization (MA31C) and the preconditioned conjugate gradient (MA31F) routines will be determined using the CDC Fortran function SECOND. This function returns the central processor time from start-of-job in seconds. The difference between the values recorded at the start and end of a routine will be its execution time. The values returned by function SECOND are usually accurate to two decimal places.

The statistics on the calculated eigenvalues will be generated using the CDC Math/Science Library routines DSCRPRT and DSCRP2. The source code for both routines is in the Cyber MSL Library.

### 4.4.3. Results

The results of the numerical experiments have been tabulated and placed in Tables 4.2 - 4.9 and Graphs 4.1 - 4.5 at the end of this section. First, I will discuss some general observations about the data. I will then look at each preconditioning strategy separately, discuss how it relates to the other preconditioning strategies, and what effect the different grid point ordering schemes may have had upon it.

There exists a definite correlation between the number of iterations required to solve the preconditioned system and the size of the spectral radius, and the range and standard deviation of the resulting eigenvalues. The smaller the spectral radius, the range, and
the standard deviation of the eigenvalues, the fewer the number of iterations. In most cases, the mean is also reasonably close to zero. This supports the idea that the closer the eigenvalues of the symmetrically preconditioned matrix are clustered around one, the faster the method will converge. Such observations, however, did not hold for the MIC(0) preconditioning strategy. Unfortunately, I have not been able to explain why. From this data, it is clear that the distribution, rather than the number of distinct eigenvalues is the characteristic relevant to the rate of convergence. In fact, the non-preconditioned matrix is the matrix with the fewest distinct eigenvalues.

As a result of the relatively small size of our test system, the times consumed by the various preconditioned C.G. algorithms are clustered together. If any method could be classified as the fastest, the Harwell($10^{-2}$) would probably be the one. It registered a time of 0.03 second when matched with the point red/black matrix and the 2-line red/black matrix. From this data alone however, it is difficult to conclude whether the difference in times resulting from the various ordering schemes is significant. When the Harwell($10^{-2}$) method is compared to its sister method MICD($10^{-2}$), the benefits of minimum degree pivoting (Harwell($10^{-2}$)) are clearly evident. In each case, the Harwell($10^{-2}$) method produced better results in every category than did the MICD($10^{-2}$) method. The MICD($10^{-2}$) method also proved extremely sensitive to the type of ordering scheme used.
Had the IC(n) methods been more efficiently implemented, they would have matched the efficiency of the Harwell(10^{-2}) method. That aside, they were still very competitive. The IC(0) method proved to be a substantial improvement over the BDS method in all areas. The results for the IC(0) method fluctuate slightly depending on the grid point ordering scheme used. It is unclear whether or not these changes are significant. Additional tests would have to be conducted. The IC(1) method made modest additional improvements to both the eigenvalue distribution and rate of convergence. The timing data between the IC(0) and IC(1) methods is so close, that it is impossible to tell which is more efficient. The IC(3) method, on the other hand, while making additional improvements on the eigenvalue distribution, did not improve the rate of convergence enough to outweigh the increased cost of the factorization. As a result, it is less desirable than the IC(0) or IC(1) methods. However, the results of Meijerink and van der Vorst [MeVo77] show that for larger systems, the IC(3) method is indeed superior. How the IC(3) method would compare to the Harwell(10^{-2}) method on larger systems has, to my knowledge, not been thoroughly explored.

The MIC(0) method proved extremely sensitive to the type of grid point ordering scheme used. It had the most trouble with the point red/black matrix. Here the process became unstable and six diagonal elements had to be changed, using Kershaw's technique [Kers78], to keep the factorization positive definite. On the other hand, with a
naturally ordered matrix, it seemed to be fairly competitive as far as the time required to obtain an answer. The eigenvalue distribution, however, suffered as compared to the IC(n) methods. While the data here indicates the MIC(O) method slightly inferior to the IC(O) method, Gustafsson [Gust78], using naturally ordered matrices, showed that for larger systems the MIC(n) methods required fewer iterations than the corresponding IC(n) methods.

The Harwell(O) method seems to be surprisingly competitive, considering that it represents a complete factorization. However, the results of Munksgaard [Munk80] show that, as would be expected, this competitiveness does not extend to larger systems. When compared to the other complete factorization method, MICD(O), the benefits of minimum degree pivoting are again clearly evident. In each case, the Harwell(O) method produced fewer fill-ins and required substantially less time to perform the factorization. Another interesting observation is that the Harwell(O) method was not influenced by the grid point ordering scheme used, while the MICD(O) was definitely sensitive to the ordering scheme used. For the MICD(O) method, the number of elements in the lower triangular part of the factorization varied from 584 to 326. This was reflected in the time required to calculate the factorization, which varied accordingly.

The DS and BDS methods were somewhat disappointing. The DS method, while improving the eigenvalue distribution tremendously, did little to
improve the rate of convergence associated with our conjugate gradient routine. The BDS method was equally ineffectual. It produced almost no improvement in the eigenvalue distribution over the DS method, and only a modest improvement in the rate of convergence. Unfortunately, this improvement in the rate of convergence was overshadowed by the cost of the factorization. As we will see in section 4.6.3, the BDS method is not as worthless as these results would indicate.

The RBIC(O) method proved to be reasonably successful. Where they could be compared, its results fall almost exactly half way between those of the BDS and IC(0) methods. Only minor fluctuations in results occurred between the various grid point ordering schemes. The RBIC(3) method, on the other hand, proved to be a major disappointment. In every category, it was inferior to the RBIC(0) method. It is unclear whether these results are characteristic of the RBIC(3) method, or simply a consequence of the size of the test problem.
<table>
<thead>
<tr>
<th>Table 4.2</th>
<th>Timing and convergence data resulting from solving the test problem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditioning Method</td>
<td>Number of Elements in L</td>
</tr>
<tr>
<td>None</td>
<td>0</td>
</tr>
<tr>
<td>DS</td>
<td>0</td>
</tr>
<tr>
<td>BDS</td>
<td>56</td>
</tr>
<tr>
<td>IC(0)</td>
<td>112</td>
</tr>
<tr>
<td>IC(1)</td>
<td>161</td>
</tr>
<tr>
<td>IC(3)</td>
<td>245</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>112</td>
</tr>
<tr>
<td>MICD(10^-2)</td>
<td>249</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>455</td>
</tr>
<tr>
<td>HARWELL(10^-2)</td>
<td>210</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>290</td>
</tr>
</tbody>
</table>

| Table 4.3 | Data on the eigenvalue distribution of the symmetrically preconditioned test matrix. |
|-----------|--------------------------------------------------------------------------------|---|---|---|
| Preconditioning Method | Spectral Radius | Number of Distinct Eigenvalues | Range | Mean | Std. Dev. |
| None | 8.876 | 33 | 7.518 | 3.00 | 2.059 |
| DS | 2.219 | 33 | 1.879 | -0.26E-6 | 0.515 |
| BDS | 2.377 | 64 | 1.773 | -0.32E-6 | 0.399 |
| IC(0) | 1.329 | 54 | 0.858 | -0.0219 | 0.164 |
| IC(1) | 1.205 | 55 | 0.512 | -0.0065 | 0.074 |
| IC(3) | 1.108 | 45 | 0.245 | -0.0029 | 0.036 |
| MIC(0) | 2.960 | 49 | 1.499 | 0.361 | 0.347 |
| MICD(10^-2) | 1.220 | 57 | 0.169 | 0.025 | 0.033 |
| MICD(0) | 1.00 | 1 | 0.0 | 0.0 | 0.0 |
| HARWELL(10^-2) | 1.076 | 38 | 0.057 | 0.013 | 0.014 |
| HARWELL(0) | 1.00 | 1 | 0.0 | 0.0 | 0.0 |

GETEIG Statistics on \( \lambda_1 - 1.0 \)
## Point Red/Black Ordering

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of Elements in L</th>
<th>Time</th>
<th>Number of Iterations</th>
<th>Time</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS</td>
<td>0</td>
<td>0.0</td>
<td>24</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>88</td>
<td>0.01*</td>
<td>17</td>
<td>0.04</td>
<td>0.05</td>
</tr>
<tr>
<td>IC(0)</td>
<td>112</td>
<td>0.01*</td>
<td>13</td>
<td>0.03</td>
<td>0.04</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>112</td>
<td>0.01*</td>
<td>24</td>
<td>0.06</td>
<td>0.07</td>
</tr>
<tr>
<td>MICD(10^-2)</td>
<td>241</td>
<td>0.03</td>
<td>5</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>326</td>
<td>0.04</td>
<td>1</td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>HARWELL(10^-2)</td>
<td>211</td>
<td>0.02</td>
<td>4</td>
<td>0.01</td>
<td>0.03</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>290</td>
<td>0.03</td>
<td>1</td>
<td>0.01</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 4.4 - Timing and convergence data resulting from solving the test problem.

<table>
<thead>
<tr>
<th>Method</th>
<th>Spectral Radius</th>
<th>Number of Distinct Eigenvalues</th>
<th>Statistics on (Λ - 1.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Range</td>
</tr>
<tr>
<td>DS</td>
<td>2.194</td>
<td>33</td>
<td>1.879</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>1.660</td>
<td>52</td>
<td>1.333</td>
</tr>
<tr>
<td>IC(0)</td>
<td>1.438</td>
<td>31</td>
<td>1.172</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>18.827</td>
<td>32</td>
<td>13.951</td>
</tr>
<tr>
<td>MICD(10^-2)</td>
<td>1.129</td>
<td>31</td>
<td>0.098</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>1.00</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>HARWELL(10^-2)</td>
<td>1.076</td>
<td>34</td>
<td>0.063</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>1.00</td>
<td>1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 4.5 - Data on the eigenvalue distribution of the symmetrically preconditioned test matrix.
Line Red/Black Ordering

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of Elements in L</th>
<th>Time</th>
<th>Number of Iterations</th>
<th>Time</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDS</td>
<td>56</td>
<td>0.01*</td>
<td>20</td>
<td>0.05</td>
<td>0.06</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>88</td>
<td>0.01*</td>
<td>16</td>
<td>0.04</td>
<td>0.05</td>
</tr>
<tr>
<td>IC(0)</td>
<td>112</td>
<td>0.01</td>
<td>14</td>
<td>0.04</td>
<td>0.05</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>112</td>
<td>0.04</td>
<td>6</td>
<td>0.02</td>
<td>0.06</td>
</tr>
<tr>
<td>MICD(10^{-2})</td>
<td>301</td>
<td>0.09</td>
<td>1</td>
<td>0.01</td>
<td>0.10</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>584</td>
<td>0.03</td>
<td>4</td>
<td>0.01</td>
<td>0.04</td>
</tr>
<tr>
<td>HARWELL(10^{-2})</td>
<td>219</td>
<td>0.03</td>
<td>1</td>
<td>0.01</td>
<td>0.04</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>290</td>
<td>0.03</td>
<td>1</td>
<td>0.01</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 4.6 - Timing and convergence data resulting from solving the test problem.

<table>
<thead>
<tr>
<th>Method</th>
<th>Spectral Radius</th>
<th>Number of Distinct Eigenvalues</th>
<th>Statistics on $(\lambda_1 - 1.0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Range</td>
<td>Mean</td>
<td>Std. Dev.</td>
</tr>
<tr>
<td>BDS</td>
<td>2.180</td>
<td>1.773</td>
<td>0.150</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>1.722</td>
<td>1.363</td>
<td>-0.0047</td>
</tr>
<tr>
<td>IC(0)</td>
<td>1.406</td>
<td>1.000</td>
<td>-0.013</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>13.192</td>
<td>9.174</td>
<td>0.653</td>
</tr>
<tr>
<td>MICD(10^{-2})</td>
<td>1.377</td>
<td>0.288</td>
<td>0.045</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>1.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>HARWELL(10^{-2})</td>
<td>1.072</td>
<td>0.056</td>
<td>0.013</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>1.00</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 4.7 - Data on the eigenvalue distribution of the symmetrically preconditioned test matrix.
2-Line Red/Black Ordering

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>Number of Elements in L</th>
<th>Time</th>
<th>Number of Iterations</th>
<th>Time</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDS</td>
<td>56</td>
<td>0.01*</td>
<td>20</td>
<td>0.05</td>
<td>0.06</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>88</td>
<td>0.01*</td>
<td>15</td>
<td>0.04</td>
<td>0.05</td>
</tr>
<tr>
<td>RBIC(3)</td>
<td>164</td>
<td>0.02*</td>
<td>16</td>
<td>0.05</td>
<td>0.07</td>
</tr>
<tr>
<td>IC(0)</td>
<td>112</td>
<td>0.01*</td>
<td>11</td>
<td>0.03</td>
<td>0.04</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>112</td>
<td>0.01</td>
<td>12</td>
<td>0.03</td>
<td>0.04</td>
</tr>
<tr>
<td>MICD(10^{-2})</td>
<td>280</td>
<td>0.03</td>
<td>7</td>
<td>0.03</td>
<td>0.06</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>562</td>
<td>0.09</td>
<td>1</td>
<td>0.01</td>
<td>0.10</td>
</tr>
<tr>
<td>HARWELL(10^{-2})</td>
<td>208</td>
<td>0.02</td>
<td>4</td>
<td>0.01</td>
<td>0.03</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>290</td>
<td>0.03</td>
<td>1</td>
<td>0.01</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 4.8 - Timing and convergence data resulting from solving the test problem.

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>Spectral Radius</th>
<th>Number of Distinct Eigenvalues</th>
<th>Statistics on ($\lambda_1 - 1.0$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDS</td>
<td>2.196</td>
<td>64</td>
<td>Range: 1.773, Mean: -0.12E-6, Std.Dev: 0.399</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>1.770</td>
<td>55</td>
<td>Range: 1.363, Mean: -0.004, Std.Dev: 0.324</td>
</tr>
<tr>
<td>RBIC(3)</td>
<td>2.052</td>
<td>49</td>
<td>Range: 1.575, Mean: -0.128E-3, Std.Dev: 0.349</td>
</tr>
<tr>
<td>IC(0)</td>
<td>1.319</td>
<td>60</td>
<td>Range: 0.904, Mean: -0.015, Std.Dev: 0.163</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>5.449</td>
<td>51</td>
<td>Range: 3.341, Mean: 0.440, Std.Dev: 0.562</td>
</tr>
<tr>
<td>MICD(10^{-2})</td>
<td>1.484</td>
<td>53</td>
<td>Range: 0.376, Mean: 0.046, Std.Dev: 0.072</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>1.00</td>
<td>1</td>
<td>Range: 1.0, Mean: 0.0, Std.Dev: 0.0</td>
</tr>
<tr>
<td>HARWELL(10^{-2})</td>
<td>1.073</td>
<td>34</td>
<td>Range: 1.058, Mean: 0.014, Std.Dev: 0.015</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>1.00</td>
<td>1</td>
<td>Range: 1.0, Mean: 0.0, Std.Dev: 0.0</td>
</tr>
</tbody>
</table>

Table 4.9 - Data on the eigenvalue distribution of the symmetrically preconditioned test matrix.

* Tests using the routines from Phase III show that these values could be reduced by up to a factor of 3 if the corresponding preconditioning strategy had been efficiently implemented.
Graph 4.1 - Shows the log base 10 of the residual norm as a function of the iteration number.
Graph 4.2 - Shows the log base 10 of the residual norm as a function of the iteration number.
Point Red/Black Ordering

Graph 4.3 - Shows the log base 10 of the residual norm as a function of the iteration number.
Graph 4.4 - Shows the log base 10 of the residual norm as a function of the iteration number.
Graph 4.5 - Shows the log base 10 of the residual norm as a function of the iteration number.
4.5. Phase II

4.5.1. Introduction

During this phase, I attempted to analyze each of the conditioning strategies and determine how easily they could be adapted to our multiprocessor. I assumed that matrix $A$ is of order $nm$ and that my multiprocessor consisted of $n/2$ processors ($p=n/2$). Under these assumptions, Table E1 (Appendix E) shows the steps involved in solving a system of equations using our preconditioned conjugate gradient algorithm. Also included are their relative cost in arithmetic operations and the amount of data that must be passed between processors. Where appropriate, the relative cost of performing a particular factorization was determined, as well as the cost of using it to solve the system of equations $z = C^{-1}r$.

As a matter of terminology, I assumed that each factorization produced a preconditioning matrix of the form

$$ C = LDL^T $$

where $L$ is a unit lower triangular matrix and $D$ is a positive diagonal matrix.

The system $z = C^{-1}r$ was solved using forward and backward substitution in the following manner:

$$ Lt = r $$
$$ L^Tz = D^{-1}t. $$
In attempting to analyze each of these events, I relied heavily on the notation defined in figures B1 - B4 of Appendix B. Furthermore, I assume that vectors x, b, r, z, and t, and matrices L and D are partitioned in the same manner as matrix A. Also, if matrix A contains a block $E_i$ and an element $a_{ij}$, the $\tilde{E}_i$ and $\tilde{a}_{ij}$ represent the corresponding block and element in $\tilde{L}$, respectively. Figure 4.1 shows which blocks of matrix A, of the unknown vector x, and of the right-hand side vector b are stored in processor $i$ ($i = 2, \ldots, \frac{n}{2} - 1$) for each of the grid point ordering schemes. For processor 1 and $n/2$, the storage requirements are slightly different, in that certain blocks mentioned in figure 4.1 are undefined. Processor $i$ is used to calculate and store the portions of vectors r, z, and t, and matrix D corresponding to those portions of vector x referred to in figure 4.1, as well as portions of $\tilde{L}$ that correspond to those blocks of matrix A cited in figure 4.1.
Data Initially stored in Processor $i$ for a

<table>
<thead>
<tr>
<th>Naturally Ordered Matrix</th>
<th>Line Ordered Matrix</th>
<th>Point Ordered Matrix</th>
<th>2-Line Ordered Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{2i-1}$</td>
<td>$T_{2i-1}$</td>
<td>$D_i$</td>
<td>$Q_i$</td>
</tr>
<tr>
<td>$T_{2i}$</td>
<td>$T_{2i}$</td>
<td>$D_{i+n/2}$</td>
<td>$F_j$</td>
</tr>
<tr>
<td>$E_{2i-2}$</td>
<td>$E_{2i-2}$</td>
<td>$B_i$</td>
<td>$C_k$</td>
</tr>
<tr>
<td>$E_{2i-1}$</td>
<td>$E_{2i-1}$</td>
<td>$E_{i-1}$</td>
<td>$x_{2i-1}$</td>
</tr>
<tr>
<td>$E_{2i}$</td>
<td>$E_{2i}$</td>
<td>$E_i$</td>
<td>$x_{2i}$</td>
</tr>
<tr>
<td>$x_{2i-1}$</td>
<td>$x_{2i-1}$</td>
<td>$F_{i-1}$</td>
<td>$b_{2i-1}$</td>
</tr>
<tr>
<td>$x_{2i}$</td>
<td>$x_{2i}$</td>
<td>$F_i$</td>
<td>$b_{2i}$</td>
</tr>
<tr>
<td>$b_{2i-1}$</td>
<td>$b_{2i-1}$</td>
<td>$x_{i+n/2}$</td>
<td>$b_i$</td>
</tr>
<tr>
<td>$b_{2i}$</td>
<td>$b_{2i}$</td>
<td></td>
<td>$b_{i+n/2}$</td>
</tr>
</tbody>
</table>

Figure 4.1
4.5.2. Results

First, I looked at those preconditioning strategies whose suitability for our multiprocessor is not influenced by the grid point ordering scheme used. These include the DS, BDS, Harwell(c) and RBIC(n) methods. The Harwell(c) method is the only one from this group that would be extremely difficult to implement. The minimum degree pivoting would require exorbitant amounts of interprocessor communications.

The remaining three methods from this group can all be easily adapted to our multiprocessor. The DS is by far the simplest. No work is required during the factorization phase, with \( \tilde{L} = I \) and \( \tilde{D} = \text{diag}(A) \). Solving the system \( z = C^{-1}r \) is simply a matter of calculating \( z_i = D_i^{-1}r_i \), which can be done in \( m \) arithmetic operations with no interprocessor communications required. Each processor would solve two such systems for a total of \( 2m \) arithmetic operations.

The BDS method is equally simple. Here, processor \( i \) is required to perform the factorization of two uncoupled tri-diagonal matrices \( T_{2i-1} \) and \( T_{2i} \). This will require \( \sim 6m \) arithmetic operations per processor. Solving the system \( z = C^{-1}r \) is equivalent to solving \( n \) uncoupled systems of the form \( z_i = T_i^{-1}r_i \) (\( i = 1, \ldots, n \)). Each processor will then solve two of these systems, requiring a total of \( \sim 10m \) arithmetic operations and no interprocessor data transfers.

The RBIC(n) method, by its very design, is ideally suited for our
multiprocessor. The n/2 uncoupled systems allow each processor to work
totally independently, while performing the factorization and solving
the system \( z = C^{-1}r \). Each of the uncoupled systems will be of order 2m
with 3m-2 non-zero off-diagonal elements in its upper triangular part.
To perform the RBIC(0) factorization, each off-diagonal element \( a_{ij} \) will
be involved in the following operations:

\[
\tilde{a}_{ij} := a_{ij} / d_i
\]

\[
\tilde{d}_j := \tilde{d}_j - \tilde{a}_{ij} a_{ij}
\]

where initially \( \tilde{d} \) is set to \( \text{diag}(A) \).

This results in an expenditure of 3 arithmetic operations per off-
diagonal element. Thus, the RBIC(0) factorization requires a total of
~9m arithmetic operations per processor. The RBIC(3) factorization is
slightly more complicated. I assume that I am working with a 2-line
red/black matrix. If \( \tilde{L}_i^T = \text{diag}(\tilde{L}_1^T, \tilde{L}_2^T, \ldots, \tilde{L}_{n/2}^T) \), then processor \( i \)
factors \( Q_i \) into \( L_i \tilde{D}_i \tilde{L}_i^T \), where figure 4.2 shows the structure of \( \tilde{L}_i^T \) and
\( \tilde{D}_i = \text{diag}(\tilde{d}_1, \ldots, \tilde{d}_{2m}) \). The elements of \( \tilde{D}_i \) and \( \tilde{L}_i^T \) are calculated in the
following manner:

\[
\tilde{d}_j := a_j - \tilde{b}_{j-1} b_{j-1} - \tilde{g}_{j-2} \tilde{g}_{j-2} - \tilde{f}_{j-m+2} \tilde{f}_{j-m+2}
- \tilde{e}_{j-m+1} e_{j-m+1} - \tilde{c}_{j-m} c_{j-m}
\]

\[
b_j := b_j - \tilde{e}_{j-m+1} c_{j-m+1} - \tilde{f}_{j-m+2} e_{j-m+2} - \tilde{g}_{j-1} \tilde{g}_{j-1}
\tilde{b}_j := b_j / \tilde{d}_j
\]

\[
\tilde{g}_j := -\tilde{f}_{j-m+2} c_{j-m+2}
\tilde{g}_j := g_j / \tilde{d}_j
\]

\[
f_j := -\tilde{g}_{j-1} e_{j-1}
\tilde{f}_j := f_j / \tilde{d}_j
\]
\[
e_j := -b_{j-1}^c j^{-1} \quad \tilde{e}_j := \frac{e_j}{d_j} \\
\tilde{c}_j := \frac{c_j}{d_j} \quad \tilde{a}_j := 1
\]

for \( j = 1, \ldots, 2m \)

where any elements not defined (ie. subscripts < 0)

are assumed to be zero.

When simplified, we find that the RBIC(3) factorization requires \(-27m\)

arithmetic operations per processor.

Solving \( z = C^{-1}r \), when matrix \( C \) is given as \( LDL^T \), requires approximately

\(2(NZL)\) arithmetic operations to solve \( \tilde{L}t = r \) and \( 2(NZL) + 2m \) to solve \( \tilde{L}^Tz = \tilde{D}^{-1}r \), where \( NZL \) is the number of non-zero off-diagonal elements in \( \tilde{L} \). After the RBIC(0) factorization, \( NZL \) will equal \( 3m-2 \), while after
the RBIC(3) factorization NZL will equal $-6m$. This means that $-14m$
arithmetic operations per processor are required if the RBIC(0) is used,
while if the RBIC(3) is used, $-26m$ arithmetic operations per processor
are needed. In either case, no interprocessor data transfers are
required during the factorization phase or while solving $z = C^{-1}r$.

Next, I will look at those preconditioning strategies that require
a certain number of fill-ins be kept, or at least calculated, during the
factorization. These methods include MICD(c), MIC(s), and IC(s) for
$s > 0$. Unfortunately, including fill-ins greatly complicates the process.
They increase the interdependence between processors both during the
factorization phase and while solving $z = C^{-1}r$. For example,
processor $i$ may be forced to wait for processor $i-1$ to finish
calculating before it can proceed with its work. As a result, only a
fraction of our $n/2$ processors may be able to operate concurrently.
This greatly reduces the advantage of having those $n/2$ processors. The
choice of grid point ordering scheme can reduce the severity of this
problem somewhat, but not enough to make any of these methods suitable
for our multiprocessor.

Finally we come to the IC(0) method. Unlike the other methods, its
suitability is influenced by the grid point ordering scheme used. If we
are working with a naturally ordered matrix, the factorization process
is recursive in nature. We find that processor $i$ cannot start its part
of the factorization process until processor $i-1$ has started calculating
These values are needed by processor \( i \) before it can start calculating \( \tilde{D}_{2i-1} \). In essence, only two processors will be able to function concurrently while performing the factorization. A similar problem arises when solving \( z = C^{-1}r \).

Changing to the 2-line red/black ordering does not help the situation that much. The only advantage gained is that now \( \left\lfloor n/4 \right\rfloor \) processors can be working concurrently while performing the factorization. The remaining \( \left\lfloor n/4 \right\rfloor \) processors must still wait until these processors have calculated the data they need. This is still an undesirable situation.

The line red/black ordering produces a matrix much more suited for performing the IC(0) factorization on our multiprocessor. Notice that the blocks \( T_{2i-1} \) (\( i = 1, \ldots, n/2 \)) are not directly interrelated. This means that processor \( i \) can factor \( T_{2i-1} \) without any interprocessor communication. This requires \( \sim 3m \) arithmetic operations. For processor \( i \) to complete the factorization, it must now get the values \( \tilde{D}_{2i+1} \) from processor \( i+1 \). With these \( m \) values, processor \( i \) can finish the factorization in \( \sim 9m \) arithmetic operations. An additional \( m \) arithmetic operations are required to calculate \( \tilde{E}_{2i-2} = \tilde{D}_{2i-1}^{-1}E_{2i-2} \). These last values will be needed by processor \( i \) to solve \( z = C^{-1}r \). This makes a total of \( \sim 13m \) arithmetic operations and \( m \) data transfers per processor to calculate the IC(0) factorization.

Solving the system \( z = C^{-1}r \) can also be easily done in this case.
During the forward substitution phase \((Lt = r)\), \(t_{2i-1}\) can be found in \(-2m\) arithmetic operations with no interprocessor communications. The elements of \(t_{2i}\) can then be found in \(-6m\) arithmetic operations, as long as the values \(t_{2i+1}\) are obtained from processor \(i+1\). The backward substitution process \((L^Tz = D^{-1}t)\) is very similar in nature. The elements of \(z_{2i}\) are first calculated using \(-3m\) arithmetic operations and no interprocessor communications. The values \(z_{2i-2}\) are then obtained from processor \(i-1\). Then the values \(z_{2i-1}\) are calculated using \(-7m\) arithmetic operations. The entire process requires a total of \(-18m\) arithmetic operations and \(2m\) data transfers.

The point red/black matrix is equally suited for performing the IC(0) factorization on our multiprocessor. In fact, it has one advantage over the line red/black matrix in that the factorization can be done without interprocessor communications. The structure of the point red/black matrix is such that blocks \(D_1\) through \(D_k\) are not altered during the factorization, \(D_i = D_i\) \((i = 1, \ldots, k)\). This allows us to store those values of \(D_{i-1}\) and \(D_{i+1}\) needed by processor \(i\) during the set-up phase. Thus, if processor \(i\) has blocks \(B_i, E_i-1, E_i, F_i-1, F_i, D_i, H(i-1)2, H(i+1)1\), and \(D_{k+1}\) available to it, the factorization can be performed without any data being transferred between processors. Processor \(i\) will perform the following calculations:
\[
\begin{align*}
\tilde{D}_1 &= D_1, \quad \tilde{H}_{(i-1)2} = H_{(i-1)2}, \quad \tilde{H}_{(i+1)1} = H_{(i+1)1} \\
\tilde{E}_{i-1}^T &= \tilde{E}_{i-1}^T, \quad \tilde{F}_i^T = \tilde{F}_i^T, \quad \tilde{B}_i = D_1^{-1} \tilde{B}_i \\
\tilde{E}_i^T &= D_1^T E_1^T, \quad \tilde{F}_{i-1}^T = D_1^T \tilde{F}_{i-1}^T \\
\tilde{D}_{i+k} &= D_{i+k} - \text{diag}(E_{i-1}^T E_{i-1}) - \text{diag}(B_i \tilde{B}_i^T) - \text{diag}(F_i \tilde{F}_i^T)
\end{align*}
\]
for a total of \(\sim 13m\) arithmetic operations.

Solving the system \(z = C^{-1}r\) will still require that some interprocessor data transfer occur. During the forward substitution phase, processor \(i\) will need from processor \(i-1\) the \(m/2\) elements of \(t_{i-1}\) corresponding to \(H_{(i-1)2}\), and from processor \(i+1\) the \(m/2\) elements of \(t_{i+1}\) corresponding to \(H_{(i+1)1}\). A similar set of transfers will be required during backward substitution, except involving elements from \(z_{i+k-1}\) and \(z_{i+k+1}\). The entire process of solving \(z = C^{-1}r\) will require \(\sim 18m\) arithmetic operations and \(2m\) data transfers per processor.

As I have indicated, only a handful of the chosen preconditioning strategies can be efficiently implemented on our multiprocessor. The DS, BDS and RBIC(n) methods can be implemented regardless of the grid point ordering scheme used. The IC(0) method, on the other hand, is sensitive to the structure of the matrix \(A\). Only when matrix \(A\) has a structure similar to that of the point red/black matrix or line red/black matrix can the IC(0) factorization be done efficiently. Implementation using the point red/black matrix has an added advantage in that the factorization can be performed without interprocessor communication.
4.6. Phase III

4.6.1. Introduction

From the results of Phase I and Phase II, the following combinations of preconditioning strategies and grid point ordering schemes are chosen for further analysis:

1) IC(0) with a point red/black matrix

2) BDS with a line red/black matrix

3) RBIC(0) with a 2-line red/black matrix.

For comparison purposes, I also consider a naturally ordered matrix with no preconditioning. These combinations are compared relative to test matrices of order \( \sim 1000 \) arising from test problem 1 (n=32), test problem 2, and test problem 3 (see appendix D). The numerical experiments are similar to those conducted during Phase I.

The size of these problems made calculating all the distinct eigenvalues of the symmetrically preconditioned test matrices extremely expensive. I therefore limit myself to examining only the extreme eigenvalues. In each case, I calculate the number of distinct eigenvalues in the interval \([0.0 , 1.2]\). Then, using the estimate of the spectral radius \(\rho\) generated by the Harwell routine EA14A, I calculate the number of distinct eigenvalues in the upper part of the spectrum defined by the interval \([0.8\rho , \rho]\). Of primary interest is the
number of eigenvalues that migrated into the lower part of the spectrum as a result of the preconditioning. The greater the number of eigenvalues in the interval \([0.0, 1.2]\), the more successful the preconditioning strategy.

Finally, the effect of different values of \(\psi\), the cost in time units to transfer a piece of data between neighboring processors, on the efficiency of each of the preconditioning strategies is examined. For each problem, I calculate the total number of time units required by a typical processor in our system to generate our answer. This was done using the following equation:

\[
\text{Total Time} = \text{Preprocessing Time} + [\text{Number of Iterations} \times \text{Time Units per Iteration}]
\]

where,

\[
\text{Preprocessing Time} = \text{Number of Arithmetic Operations} + [\psi \times \text{Number of Data Transfers}],
\]

\[
\text{Time Units per Iteration} = \text{Number of Arithmetic Operations/iteration} + [\psi \times \text{Number of Data Transfers/iteration}].
\]

Appendix E outlines the number of arithmetic operations and data transfers required by each processor to perform each step of our preconditioned conjugate gradient algorithm.
4.6.2. Software

Most of the software used during this phase is similar to that used during Phase I. However, more efficient routines BDIAG, ICCGO, and RBICO are developed to implement the BDS, IC(O), and RBIC(O) factorizations, respectively. Each of these three routines is based on the following algorithm:

Algorithm 4.1

1) $\tilde{D} = \text{diag}(A)$

2) For $i = 1$ to $N$ do

3) For $j \in R^A_i$ do

4) $\tilde{r}_{ij} = a_{ij}/d_i$

5) $d_j = d_j - \tilde{r}_{ij}a_{ij}$

where $N = \text{order}(A)$, and set $R^A_i$ defines which columns in row $i$ are to be used in calculating the factorization.

For these three routines, the following is how set $R^A_i$ is defined:

- $\text{BDS} - R^A_i = \{ j \mid j > i + 1 \text{ and } a_{ij} \neq 0 \}$
- $\text{IC(O)} - R^A_i = \{ j \mid i < j \text{ and } a_{ij} \neq 0 \}$
- $\text{RBIC(O)} - R^A_i = \{ j \mid i < j < i+m \text{ and } a_{ij} \neq 0 \}$. 
4.6.3. Results

The data from the numerical experiments can be found at the end of this section. Tables 4.10, 12, and 14 contain the timing and convergence data pertaining to solving each of the test problems. Tables 4.11, 13, and 15 contain the corresponding data on the extreme eigenvalues of the symmetrically preconditioned test matrices. Graphs 4.6 - 4.8 show the $\log_{10}$ of the norm of the residual as a function of the iteration number. Graphs 4.9 - 4.11 show what effect the interprocessor communications cost $(\psi)$ can have on the amount of work required by each processor to calculate an acceptable answer.

Notice that in these cases, the time required to perform the desired factorization is trivial when compared to that required to actually solve the system. This would indicate that the savings incurred by using the point red/black ordering with the IC(0) method, as opposed to using the line red/black ordering or block cyclic reduction, may not be that significant in the long run. However, unless circumstances dictate otherwise, there is no reason not to utilize the point red/black ordering and enjoy what savings it can provide.

For these test problems, the RBIC(0) method prove at least as equal to the IC(0) method in efficiency. Only in the case of test problem 1 does the IC(0) prove more efficient than the RBIC(0) method. The two methods are extremely close in the number of iterations required to solve the test problems. The RBIC(0), therefore, has a slight advantage
in that each iteration requires fewer arithmetic operations due to the fewer non-zero elements in the upper triangular part of its factorization. The BDS method is consistently a distant third, though it does represent a improvement over no preconditioning.

Unfortunately, the matrices symmetrically preconditioned by the BDS and RBIC(0) methods consistently require more than the 750 iterations I have allotted for calculating their eigenvalues. As a result, these counts may be incomplete, but should be reasonably close. The BDS method results in substantial improvement in the eigenvalue distribution as compared to the matrix without preconditioning. The RBIC(0) and IC(0) methods then each register moderate subsequent improvements. The IC(0) method, as would be expected, produces the "best" eigenvalue distribution of the three. It records the smallest spectral radius and causes the greatest number of eigenvalues to migrate into the lower interval.

Looking at Graph 4.9 - 4.11, we see that as the cost to transfer a piece of data between neighboring processors (ψ) increases, the advantages of using the RBIC(0) factorization also increase. For ψ = 10, which may not be unrealistic for loosely connected processors, the RBIC(0) saves between 1500 and 60,000 time units over the IC(0) method. This advantage stems from the fact that the RBIC(0) method requires no data transfers to solve the system z = C⁻¹r, while the IC(0) method requires 2m data transfers.
Test Problem 1 (n=32)

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>FACTOR</th>
<th>SOLVE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of Elements in L</td>
<td>Time</td>
</tr>
<tr>
<td>None</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>BDS</td>
<td>992</td>
<td>0.03</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>1504</td>
<td>0.03</td>
</tr>
<tr>
<td>IC(0)</td>
<td>1984</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Table 4.10 - Timing and convergence data pertaining to solving the given test problem, such that \(||r|| < 10^{-6}\).

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>Spectral Radius</th>
<th>Number of Eigenvalues in lower interval</th>
<th>Number of Eigenvalues in upper interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>8.664</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>BDS</td>
<td>2.281</td>
<td>30*</td>
<td>4</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>1.796</td>
<td>32*</td>
<td>6</td>
</tr>
<tr>
<td>IC(0)</td>
<td>1.591</td>
<td>46</td>
<td>3</td>
</tr>
</tbody>
</table>

* Number of Eigenvalues found after 750 iterations.

Table 4.11 - Data on the extreme Eigenvalues
Test Problem 2

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>Number of Elements in L</th>
<th>Time</th>
<th>Number of Iterations</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>0.0</td>
<td>99*</td>
<td>2.98</td>
</tr>
<tr>
<td>BDS</td>
<td>960</td>
<td>0.03</td>
<td>104</td>
<td>3.75</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>1456</td>
<td>0.03</td>
<td>75</td>
<td>2.86</td>
</tr>
<tr>
<td>IC(0)</td>
<td>1921</td>
<td>0.02</td>
<td>75</td>
<td>2.99</td>
</tr>
</tbody>
</table>

* $\|r\|_9 = 0.105\times10^{-2}$

Table 4.12 - Timing and convergence data pertaining to solving the given test problem, such that $\|r\|<10^{-6}$.

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>Spectral Radius</th>
<th>Number of Eigenvalues in lower interval</th>
<th>Number of Eigenvalues in upper interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>8.579</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>BDS</td>
<td>2.257</td>
<td>29*</td>
<td>5</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>1.798</td>
<td>35*</td>
<td>5</td>
</tr>
<tr>
<td>IC(0)</td>
<td>1.684</td>
<td>49</td>
<td>3</td>
</tr>
</tbody>
</table>

* Number of Eigenvalues found after 750 iterations.

Table 4.13 - Data on the extreme Eigenvalues.
Test Problem 3

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>Number of Elements in L</th>
<th>Time</th>
<th>Number of Iterations</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>0.0</td>
<td>99*</td>
<td>3.02</td>
</tr>
<tr>
<td>BDS</td>
<td>960</td>
<td>0.03</td>
<td>93</td>
<td>3.64</td>
</tr>
<tr>
<td>RBIC(O)</td>
<td>1456</td>
<td>0.02</td>
<td>68</td>
<td>2.60</td>
</tr>
<tr>
<td>IC(O)</td>
<td>1921</td>
<td>0.02</td>
<td>66</td>
<td>2.66</td>
</tr>
</tbody>
</table>

* $||r|| = 0.288E-02$

Table 4.14 - Timing and convergence data pertaining to solving the given test problem, such that $||r|| < 10^{-6}$.

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>Spectral Radius</th>
<th>Number of Eigenvalues in lower interval $^1$</th>
<th>Number of Eigenvalues in upper interval $^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>22.977</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>BDS</td>
<td>2.236</td>
<td>31*</td>
<td>7</td>
</tr>
<tr>
<td>RBIC(O)</td>
<td>1.903</td>
<td>31*</td>
<td>5</td>
</tr>
<tr>
<td>IC(O)</td>
<td>1.634</td>
<td>49</td>
<td>6</td>
</tr>
</tbody>
</table>

* Number of Eigenvalues found after 750 iterations.

Table 4.15 - Data on the extreme Eigenvalues.

$^1$ - lower interval defined as [0.0, 1.2]

$^2$ - upper interval defined as [0.8p, p], where p = spectral radius.
Test Problem 1 \( (n=32) \)

Graph 4.6 - The log base 10 of the norm of the residual as a function of the iteration number.
Test Problem 2

Graph 4.7 - The log base 10 of the norm of the residual as a function of the iteration number.
Test Problem 3

<table>
<thead>
<tr>
<th>Graph Number</th>
<th>Preconditioning Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>BDS</td>
</tr>
<tr>
<td>3</td>
<td>RBIC(0)</td>
</tr>
<tr>
<td>4</td>
<td>IC(0)</td>
</tr>
</tbody>
</table>

Graph 4.8 - The log base 10 of the norm of the residual as a function of the iteration number.
Graph 4.9 - Number of time units required to solve the given test problem vs. the cost in time units to transfer one piece of data between two processors.
Test Problem 2

Graph 4.10 - Number of time units required to solve the given test Problem vs. the cost in time units to transfer one piece of data between two processors.
Graph 4.11 - Number of time units required to solve the given test problem vs. the cost in time units to transfer one piece of data between two processors.
5. Conclusions

As we have seen, only a limited number of our original preconditioning strategies proved suitable for implementation on our multiprocessor. The DS, BDS, and RBIC(0) methods proved acceptable no matter which grid point ordering scheme was used. The IC(0) method, on the other hand, was only feasible when teamed with point red/black or line red/black matrices. When point red/black matrices were used, the IC(0) factorization could be performed without any interprocessor communications.

The numerical experiments showed that, for our given test problems of order ~1000, the RBIC(0) method, in most cases, was more efficient than the IC(0) method. This was especially true when viewed from the standpoint of our hypothetical multiprocessor. For values of \( \phi > 1 \), the RBIC(0) method was substantially faster.

While I realize that these few test results do not prove that the RBIC(0) method is a superior method in all cases, they do indicate that the RBIC(0) method could be an efficient tool for preconditioning on a multiprocessor. More testing is needed to identify the scope of its potential.
Appendices
Appendix A

Grid point ordering schemes
Natural Ordering for n=6

<table>
<thead>
<tr>
<th>6</th>
<th>12</th>
<th>18</th>
<th>24</th>
<th>30</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>11</td>
<td>17</td>
<td>23</td>
<td>29</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>16</td>
<td>22</td>
<td>28</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>15</td>
<td>21</td>
<td>27</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>14</td>
<td>20</td>
<td>26</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>13</td>
<td>19</td>
<td>25</td>
<td>31</td>
</tr>
</tbody>
</table>

Figure A1

Point Red/Black Ordering for n=6

<table>
<thead>
<tr>
<th>+21</th>
<th>6</th>
<th>+27</th>
<th>12</th>
<th>+33</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>+3</td>
<td>+24</td>
<td>9</td>
<td>+30</td>
<td>15</td>
<td>+36</td>
</tr>
<tr>
<td>+20</td>
<td>5</td>
<td>+26</td>
<td>11</td>
<td>+32</td>
<td>17</td>
</tr>
<tr>
<td>+2</td>
<td>+23</td>
<td>8</td>
<td>+29</td>
<td>14</td>
<td>+35</td>
</tr>
<tr>
<td>+19</td>
<td>4</td>
<td>+25</td>
<td>10</td>
<td>+31</td>
<td>16</td>
</tr>
<tr>
<td>+1</td>
<td>+22</td>
<td>7</td>
<td>+28</td>
<td>13</td>
<td>+34</td>
</tr>
</tbody>
</table>

Figure A2
Line Red/Black Ordering for n=6

\[
\begin{array}{cccccc}
6 & 12 & 24 & 30 & 18 & 36 \\
5 & 11 & 23 & 29 & 17 & 35 \\
4 & 10 & 22 & 28 & 16 & 34 \\
3 & 9 & 21 & 27 & 15 & 33 \\
2 & 8 & 20 & 26 & 14 & 32 \\
1 & 7 & 19 & 25 & 13 & 31 \\
\end{array}
\]

Figure A3

Two Line Red/Black Ordering for n=6

\[
\begin{array}{cccccc}
6 & 12 & 24 & 30 & 18 & 26 \\
5 & 11 & 29 & 23 & 17 & 32 \\
4 & 10 & 28 & 34 & 16 & 30 \\
3 & 9 & 27 & 33 & 15 & 29 \\
2 & 8 & 26 & 32 & 14 & 28 \\
1 & 7 & 25 & 31 & 13 & 27 \\
\end{array}
\]

Figure A4
Non-zero Structure of Matrix with Natural Ordering

Figure A5
Non-zero Structure of Matrix with Point Red/Black Ordering

Figure A6
Non-zero Structure of Matrix with Line Red/Black Ordering

<table>
<thead>
<tr>
<th>X</th>
<th>X</th>
<th></th>
<th>I</th>
<th>X</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Figure A7
Non-zero Structure of Matrix with 2-line Red/Black Ordering

Figure A8
Appendix B

Matrix block structures

Appendix B shows the block structure of the matrices associated with the four grid point ordering schemes defined in Appendix A. I assume discretization took place on a n×m grid, with n being even.
Natural Ordering

\[
\begin{bmatrix}
T_1 & E_1 \\
E_1 & T_2 & E_2 \\
& \ddots & \ddots & \ddots \\
E_{i-1} & T_i & E_i & \\
& E_i & T_{i+1} & E_{i+1} & \\
& E_{i+1} & T_{i+2} & E_{i+2} & \ddots \\
& & \ddots & \ddots & \ddots \\
E_{n-2} & T_{n-1} & E_{n-1} & \\
E_{n-1} & T_n
\end{bmatrix}
\]

where \(E_i\)'s are diagonal matrices of order \(m\) and \(T_i\)'s are tri-diagonal matrices of order \(m\).

Figure B1
Line Red/Black Ordering

where $T_1$ and $E_1$ are the same as those used in figure B1

Figure B2
2-line Red/Black Ordering

\[
\begin{array}{c|c}
Q_1 & C_1^T \\
Q_3 & F_2 C_3^T \\
\vdots & \vdots \\
Q_{2i-1} & \cdots C_{2i-1}^T \\
Q_{2i+1} & F_{2i} C_{2i+1}^T \\
\vdots & \cdots \\
Q_{p-3} & \cdots C_{p-3}^T \\
Q_{p-1} & F_{p-2} C_{p-1}^T \\
\end{array}
\]

where \( k = n/2 \) and \( p = 2(\lceil k/2 \rceil) \)

\[
Q_i = \begin{bmatrix}
T_{2i-1} & E_{2i-1} \\
E_{2i-1} & T_{2i}
\end{bmatrix}, \quad C_i = \begin{bmatrix}
0 & E_{2i} \\
0 & 0
\end{bmatrix}, \quad F_i = \begin{bmatrix}
0 & 0 \\
E_{2i} & 0
\end{bmatrix}, \quad Q_1 = \begin{bmatrix}
T_1 & E_1 \\
E_1 & T_1
\end{bmatrix}
\]

where \( T_1 \) and \( E_1 \) are the same as those used in figure B1.

If \( p \neq k \), ignore last row and column.

Figure B3
Relating the block structure of the point red/black matrix to that of the naturally ordered matrix is not as easy as with the line red/black matrix and the 2-line red/black matrix. The integrity of the $T_i$ and $E_i$ blocks is not maintained during the reordering. A relationship does exist between the two, but not at the block level. We find that the point red/black matrix ($A'$) and the naturally ordered matrix ($A$) are related such that

$$A' = P^T A P$$

where $P$ is the permutation matrix

$$P = [P_1, P_2, \ldots, P_n; Q_1, Q_2, \ldots, Q_n],$$

in which for $k = 1, 2, \ldots, n/2$

$$P_{2k-1} = [e_{j(k)}, e_{j(k)+2}, e_{j(k)+4}, \ldots, e_{j(k)+m-2}],$$

$$Q_{2k-1} = [e_{j(k)+1}, e_{j(k)+3}, \ldots, e_{j(k)+m-1}],$$

$$P_{2k} = [e_{1(k)}, e_{1(k)+2}, \ldots, e_{1(k)+m-2}],$$

$$Q_{2k} = [e_{1(k)-1}, e_{1(k)+1}, \ldots, e_{1(k)+m-3}],$$

with $j(k) = 2(k-1)m+1$ and $1(k) = j(k)+m+1$.

Figure B4 outlines the block structure for a point red/black matrix. The blocks here are different from those found in figures B1 - B3.
Point Red/Black Ordering

\[
\begin{array}{cccc}
D_1 & B_1^T & E_1^T & F_1^T \\
D_2 & B_2^T & E_2^T & F_1^T \\
\vdots & \vdots & \vdots & \vdots \\
D_{i-1} & F_{i-2}^T & B_{i-1}^T & E_{i-1}^T \\
D_i & F_i^T & B_i^T & E_i^T \\
D_{i+1} & F_{i+1}^T & B_{i+1}^T & E_{i+1}^T \\
\vdots & \vdots & \vdots & \vdots \\
D_k & F_k^T & B_k^T & \\
\end{array}
\]

where \( k = n/2 \) and

\[
B_1 = \begin{bmatrix} B_{11} & G_{11} \\ G_{12} & B_{12} \end{bmatrix}, \quad D_1 = \begin{bmatrix} H_{11} & 0 \\ 0 & H_{12} \end{bmatrix}, \quad E_1 = \begin{bmatrix} 0 & E_{11} \\ E_{11} & 0 \end{bmatrix}, \quad F_1 = \begin{bmatrix} 0 & 0 \\ 0 & F_{12} \end{bmatrix}
\]

with \( H_{11}, E_{11}, F_{12} \) and \( G_{11} \) being diagonal matrices of order \( m/2 \)

and \( B_{11} \) and \( B_{12} \) being upper and lower bi-diagonal matrices of order \( m/2 \)

Figure B4
Appendix C

User input parameters

Appendix C outlines the combinations of grid point ordering schemes and preconditioning strategies to be examined during phase I. The parameters and functions required by subroutine FACTOR to generate each of the combinations are defined. The abbreviations used to describe the various preconditioning strategies are defined in section 4.3.
Natural Ordering ($\text{NTYPE} = 0$)

<table>
<thead>
<tr>
<th>Preconditioning</th>
<th>OPTION vector</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DS</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BDS</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IC(0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IC(1)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IC(3)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>HARWELL($10^{-2}$)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MICD($10^{-2}$)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table C1
Line Red/Black Ordering (NTYPE = 1)

<table>
<thead>
<tr>
<th>Preconditioning</th>
<th>OPTION vector</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>1 2 3 4 5 6 C</td>
<td>C EUSE FILL</td>
</tr>
<tr>
<td>BDS</td>
<td>0 0 0 1 0 0 -</td>
<td>EUSE2 FILL1</td>
</tr>
<tr>
<td>IC(0)</td>
<td>0 0 0 0 0 - -</td>
<td>FILL1</td>
</tr>
<tr>
<td>HARWELL(10^{-2})</td>
<td>1 1 0 0 1 0 10^{-2}</td>
<td>- -</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>1 1 0 0 1 0 0.0</td>
<td>- -</td>
</tr>
<tr>
<td>MICD(10^{-2})</td>
<td>0 1 0 0 1 0 10^{-2}</td>
<td>- -</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>0 1 0 0 1 0 0.0</td>
<td>- -</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>0 0 0 1 0 0 - EUSE3</td>
<td>FILL1</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>0 0 0 0 1 0 - - FILL1</td>
<td></td>
</tr>
</tbody>
</table>

Table C2
Point Red/Black Ordering  (NTYPE = 2)

<table>
<thead>
<tr>
<th>Preconditioning</th>
<th>OPTION vector</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>EUSE</th>
<th>FILL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>FILL1</td>
</tr>
<tr>
<td>IC(0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>HARWELL(10^{-2})</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10^{-2}</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MICD(10^{-2})</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10^{-2}</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>FILL1</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>FILL1</td>
</tr>
</tbody>
</table>

Table C3
### 2 Line Red/Black Ordering (NTYPE = 3)

<table>
<thead>
<tr>
<th>Preconditioning</th>
<th>OPTION vector</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>1 2 3 4 5 6  C</td>
<td>EUSE  FILL</td>
</tr>
<tr>
<td>BDS</td>
<td>0 0 0 1 0 0</td>
<td>- EUSE2 FILL1</td>
</tr>
<tr>
<td>IC(0)</td>
<td>0 0 0 0 0 0</td>
<td>- FILL1</td>
</tr>
<tr>
<td>MIC(0)</td>
<td>0 0 0 0 1 0</td>
<td>- FILL1</td>
</tr>
<tr>
<td>HARWELL(10^-2)</td>
<td>1 1 0 0 1 0 10^-2</td>
<td>- -</td>
</tr>
<tr>
<td>HARWELL(0)</td>
<td>1 1 0 0 1 0 0.0</td>
<td>- -</td>
</tr>
<tr>
<td>MICD(10^-2)</td>
<td>0 1 0 0 1 0 10^-2</td>
<td>- -</td>
</tr>
<tr>
<td>MICD(0)</td>
<td>0 1 0 0 1 0 0.0</td>
<td>- -</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>0 0 0 1 0 0</td>
<td>- EUSE5 FILL1</td>
</tr>
<tr>
<td>RBIC(3)</td>
<td>0 0 0 1 0 0</td>
<td>- EUSE5 FILL3</td>
</tr>
</tbody>
</table>

Table C4
Definitions for parameters used in Tables C1 - C4.

OPTION(1) = 0 - Natural order factorization
               1 - Minimum degree factorization

OPTION(2) = 0 - Function FILL used to control fill-ins
               1 - Drop tolerance C used to control fill-ins

OPTION(3) = 0 - No diagonal scaling prior to factorization
               1 - Diagonal elements scaled by $1 + \text{ABS}(C)/N$

OPTION(4) = 0 - All matrix elements used in calculating
               the incomplete factorization
               1 - Function EUSE determines which matrix
                   elements to use in calculating the
                   incomplete factorization

OPTION(5) = 0 - No diagonal modification
               1 - Diagonal modification performed

OPTION(6) = 0 - Calculate the desired preconditioning matrix
               1 - Bypass calculating the preconditioning matrix

C - Drop tolerance used when OPTION(2) is in affect

EUSE - Function used to determine which elements of matrix A
        are to be used in calculating the incomplete
        factorization

FILL - Function used to determine which fill-ins to keep
       during the incomplete factorization
Appendix D

Definition of test problems.
Test Problem 1

Laplace Equation

\[ \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0 \]

over the unit square with Dirichlet boundary conditions:

\[(0,1) \quad u=1 \quad (1,1)\]

\[ (0,0) \quad u=1 \quad (1,0) \]

Phase I

\[ n=8 \quad m=8 \]

Matrix A of order 64

Phase III

\[ n=32 \quad m=32 \]

Matrix A of order 1024

Matrix A will be a positive definite M-matrix.
Test Problem 2

Laplace Equation

\[- \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0\]

over the unit square with boundary conditions:

\[\frac{\partial u}{\partial n} = 0\]

(0,1) \hspace{1cm} (1,1)

\[\frac{\partial u}{\partial n} = 0\]

(0,0) \hspace{1cm} (1,0)

\[u = 1\]

\[\Delta x = 1/31 \quad \Delta y = 1/31\]

n=32 \quad m=31

Matrix A is of order 992

Matrix A will be a positive definite M-matrix.
Preconception Strategies for Solving Elliptic Difference Equations
Test Problem 3

\[- \frac{\partial}{\partial x} ((x^2 + y^2 + 1) \frac{\partial u}{\partial x}) - \frac{\partial}{\partial y} (e^{xy} \frac{\partial u}{\partial y}) + u = f(x, y)\]

\[f(x, y) = e^{x^2 y} (1 - (4x^4 y^2 + 4x^2 y^4 + 2x^2 y^2 + 6x^2 y + 2y^3 + 2y) - (x^4 + x^3) e^{xy})\]

over the unit square with boundary conditions

<table>
<thead>
<tr>
<th>(0,0)</th>
<th>(1,0)</th>
<th>(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1)</td>
<td>(3)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>(2)</td>
<td>(4)</td>
<td></td>
</tr>
</tbody>
</table>

(1) \( u = 1 \)
(2) \( \frac{\partial u}{\partial n} = 0 \)
(3) \( u + \frac{\partial u}{\partial n} = e^{x^2} (1 + x^2) \)
(4) \( 2 \frac{\partial u}{\partial n} = 4ye^y \)

with \( \Delta x = 1/31 \) \( \Delta y = 1/31 \)

\( n=32 \quad m=31 \)

Matrix A is of order 992

Matrix A will be positive definite, but not an M-matrix.
Appendix E

Cost of Conjugate Gradient Algorithm

Outlines the number of arithmetic operations and data transfers required by each step of our preconditioned conjugate gradient algorithm if implemented on our multiprocessor. I assume that the system of equations being solved is of order nm, where n is even, and that our multiprocessor consists of n/2 processors as arranged in figure 1.1.
Preprocessing factorization CG Algorithm

<table>
<thead>
<tr>
<th>Preprocessing</th>
<th>Arithmetic Operations</th>
<th>Interprocessor Data Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₀ = C⁻¹b</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>r = Ax₀ - b</td>
<td>2m - 8</td>
<td>2m</td>
</tr>
<tr>
<td>(rᵀr)¹/₂</td>
<td>½m</td>
<td>m/2</td>
</tr>
<tr>
<td>g = C⁻¹r</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>e = -g</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>δ₀ = rᵀg</td>
<td>½m</td>
<td>m/2</td>
</tr>
</tbody>
</table>

Each CG Iteration

<table>
<thead>
<tr>
<th></th>
<th>Arithmetic Operations</th>
<th>Interprocessor Data Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>f = Ae</td>
<td>20m - 8</td>
<td>2m</td>
</tr>
<tr>
<td>λ = δ₀/eᵀf</td>
<td>½m</td>
<td>m/2</td>
</tr>
<tr>
<td>x = x + λe</td>
<td>4m</td>
<td>0</td>
</tr>
<tr>
<td>r = r + λf</td>
<td>4m</td>
<td>0</td>
</tr>
<tr>
<td>(rᵀr)¹/₂</td>
<td>½m</td>
<td>m/2</td>
</tr>
<tr>
<td>g = C⁻¹r</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>δ₁ = rᵀg</td>
<td>½m</td>
<td>m/2</td>
</tr>
<tr>
<td>β = δ₁/δ₀</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>δ₀ = δ₁</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e = -g + βe</td>
<td>4m</td>
<td>0</td>
</tr>
</tbody>
</table>

Table E1 - Cost breakdown of each step of a preconditioned conjugate gradient algorithm as implemented on our multiprocessor.
### Table E2

Cost breakdown of the preconditioning method dependent items from Table E1 for the preconditioning methods considered during Phase III.

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>Factorization</th>
<th>$g = \mathbf{C}^{-1}\mathbf{r}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDS</td>
<td>6m</td>
<td>0</td>
</tr>
<tr>
<td>IC(0)</td>
<td>13m</td>
<td>0</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>9m</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table E3

Outlines the costs associated with the preprocessing stage and each CG iteration for the preconditioning strategies considered during Phase III if implemented on our multiprocessor.

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>Preprocessing</th>
<th>1 CG iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>31m</td>
<td>3m</td>
</tr>
<tr>
<td>BDS</td>
<td>57m</td>
<td>3m</td>
</tr>
<tr>
<td>IC(0)</td>
<td>80m</td>
<td>7m</td>
</tr>
<tr>
<td>RBIC(0)</td>
<td>68m</td>
<td>3m</td>
</tr>
</tbody>
</table>
Hierarchy Phase I Software

PROG1

- GENA
  - NORDER
  - ISTORE
  - MA31E*
- FACTOR
  - EUSE
  - MA31C
  - MA31D*
  - FILL
- SOLVE
  - MA31F
  - MA31G*
  - MA31H

PROG2

- GENA
  - NORDER
  - ISTORE
  - MA31E*
  - FACTOR
  - EUSE
  - MA31C
  - MA31D*
  - FILL
  - GETEIG
  - EA14AD*
  - MA31G2
  - MA31H
  - MA31G1
  - DSCRPT*
  - DSCRP2*
Heirarchy Phase III Software

<table>
<thead>
<tr>
<th>PROG1A/B/C/D</th>
<th>PROG2A/B/C/D</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENA</td>
<td>GENA</td>
</tr>
<tr>
<td>. NORDER</td>
<td>. NORDER</td>
</tr>
<tr>
<td>. ISTORE</td>
<td>. ISTORE</td>
</tr>
<tr>
<td>. MA31E*</td>
<td>. MA31E*</td>
</tr>
<tr>
<td>. ICCGO/BDIAG/RBICO</td>
<td>. ICCGO/BDIAG/RBICO</td>
</tr>
<tr>
<td>. SOLVE</td>
<td>. GETEG2</td>
</tr>
<tr>
<td>. MA31F</td>
<td>. EA14AD*</td>
</tr>
<tr>
<td>. MA31G*</td>
<td>. MA31G2</td>
</tr>
<tr>
<td>. MA31H</td>
<td>. MA31H</td>
</tr>
<tr>
<td></td>
<td>. MA31G1</td>
</tr>
</tbody>
</table>

* Program listings for these routines are not included.

They maybe found in the following locations:

MA31D, MA31E and MA31G - Cyber Harwell library as part of the MA31A package.

EA14AD - Cyber Harwell library

DSCRPT and DSCRPT2 - Cyber MSL library
PROGRAM PROG1(INPUT, OUTPUT, MESS, TAPE4=INPUT, TAPE5=MESS, *TAPE6=OUTPUT)

C
SOLVE THE LINEAR SYSTEM OF EQUATIONS ARISING FROM
THE DISCRETIZATION FOR OUR MODEL PROBLEM USING A
PRECONDITIONED CONJUGATE GRADIENT ALGORITHM.

C
SUBROUTINE GENA
---------------
10 C
PERFORMS THE DISCRETIZATION OF THE CURRENT PROBLEM.
THE USER SPECIFIED INPUT PARAMETER NTYPE DETERMINES
THE TYPE OF GRID POINT ORDERING SCHEME TO BE USED:
NTYPE = 0 - NATURAL
1 - LINE RED/BLACK
2 - POINT RED/BLACK
3 - 2 LINE RED/BLACK

C
SUBROUTINE FACTOR
------------------
20 C
CALCULATES THE PRECONDITIONING MATRIX BY INCOMPLETE
FACTORIZATION. THE TYPE OF INCOMPLETE FACTORIZATION
DONE IS DETERMINED BY THE USER SPECIFIED OPTION VECTOR:
OPTION(1) = 0 - NATURAL ORDER FACTORIZATION
1 - MINIMUM DEGREE FACTORIZATION
25 C
OPTION(2) = 0 - FUNCTION FILL USED TO CONTROL FILL-INS
1 - DROP TOLERANCE C USED TO CONTROL
FILL-INS
OPTION(3) = 0 - NO DIAGONAL SCALING PRIOR TO
FACTORIZATION
30 C
1 - DIAGONAL ELEMENTS SCALED BY 1+ABS(C)/N
PRIOR TO FACTORIZATION
OPTION(4) = 0 - ALL MATRIX ELEMENTS USED IN CALCULATING
THE INCOMPLETE FACTORIZATION
1 - FUNCTION EUSE DETERMINES WHICH MATRIX
35 C
ELEMENTS TO USE IN CALCULATING
THE INCOMPLETE FACTORIZATION
OPTION(5) = 0 - NO DIAGONAL MODIFICATION
1 - DIAGONAL MODIFICATION PERFORMED
C
OPTION(6) = 0 - CALCULATE THE DESIRED PRECONDITIONING
MATRIX
1 - BYPASS CALCULATING PRECONDITIONING MATRIX

C
SUBROUTINE SOLVE
-----------------
45 C
SOLVES THE LINEAR SYSTEM USING THE HARWELL MA31F
PRECONDITIONED CONJUGATE GRADIENT ALGORITHM.
MITS - MAXIMUM NUMBER OF ITERATIONS ATTEMPTED
EPS - DESIRED ACCURACY OF SOLUTION IN TERM OF
THE NORM OF THE RESIDUAL

FOR MORE DETAILS SEE THE INDIVIDUAL SUBROUTINES.

REAL A(650), B(64), W(64, 3), WI(64, 3)
INTEGER INI(200), INJ(650), IK(64, 4), IW(64, 4), OPTION(6)

COMMON/MA31I/DD, LP, MP
COMMON/MA31J/LROW, LCOL, NCP, ND, IPD
COMMON/MA31K/NURL, NUCL, NUAL
COMMON/MCOMM3/OPTION
COMMON/MA31L/EPSTOL, U
COMMON/MA31M/NI, NJ, NVERSNI, NTYPE
COMMON/MA31N/MITS, EPS1

EXTERNAL FILL, EUSE

DATA DD, LP, MP/1.0, 6, 5/
DATA EPSTOL, U/2.0E-6, 1.0E2/
DATA NI, NJ/8, 8/
DATA IAI, IAJ, NN/200, 650, 64/
DATA MITS, EPS1/50, 1.0E-6/

ND=NN

READ(4, *) NTYPE, NVERSNI
READ(4, *) (OPTION(I), I=1, 6)
READ(4, *) C

CALL GENA(NN, NZ, A, INI, INJ, IAI, IAJ, W, B, IK, IW)

IF (OPTION(6).EQ.1) GO TO 5

C
PERFORM THE DESIRED FACTORIZATION

C
CALL FACTOR(NN, NZ, A, INI, INJ, IAI, IAJ, IK, IW, W, C, FILL, EUSE)

GO TO 15

5 CONTINUE

C
NO PRECONDITIONING REQUESTED

C
GENERATE IDENTITY MATRIX

LROW=0
DO 10 I=1, NN
IK(I, 1)=0
IK(I, 2)=I
WI(2)=1.0
10 CONTINUE
99

15 CONTINUE
C PERFORM THE PRECONDITIONED CONJUGATE GRADIENT ITERATION
100 C CALL SOLVE(NN,NZ,A,INI,INJ,IAI,IAJ,W,IK,B,WI)
C END

SUBROUTINE GENA(NN,NZ,A,INI,INJ,IAI,IAJ,D,B,IK,IW)
C**********************************************************************
C GENAI
C------
C PERFORMS THE DISCRETIZATION OF THE LAPLACE EQUATION
C OVER THE UNIT SQUARE WITH DIRICHLET BOUNDARY CONDITIONS
10 C USING A NI X NJ GRID.
C IDENTIFIED AS PROBLEM 1 IN TEXT.
C**********************************************************************
C INPUT PARAMETERS
C------------------
C NN - ORDER OF MATRIX A
C IAI - SIZE OF ARRAY INI
20 C IAJ - SIZE OF ARRAYS INJ AND A
C OUTPUT PARAMETERS
C------------------
C NZ - NUMBER OF NON-ZERO ELEMENTS IN THE UPPER
C TRANGULAR PORTION OF MATRIX A
C A - ARRAY CONTAINING THE NON-ZERO ELEMENTS IN
C THE UPPER TRANGULAR PORTION OF MATRIX A
C IN ROW ORDER
30 C INI/INJ - ARRAYS CONTAINING THE ROW/COLUMN
C INDICES OF THE CORRESPONDING ENTRY
C IN ARRAY A (IE. INI(I) AND INJ(I)
C CONTAIN THE ROW AND COLUMN INDEX
C FOR THE ENTRY IN A(I) )
35 C D - ARRAY CONTAINING THE DIAGONAL ELEMENTS OF
C MATRIX A
C B - CONTAINS THE RESULTING RIGHHAND SIDE
C IK(I,1) - NUMBER OF ELEMENTS IN ARRAY A BELONGING
C TO ROW I
40 C IK(J,2) - NUMBER OF ELEMENTS IN ARRAY A BELONGING
C TO COLUMN J
C IW(I) - POINTS TO THE FIRST ELEMENT OF ROW I IN
C ARRAY A
C
45 C COMMN BLOCK PARAMETERS
C ------------------------
C LROW,LCOL,NCP,IPD,DD - NOT USED
C ND - ORDER OF MATRIX A
50 C LP - OUTPUT FILE UNIT NUMBER
C MP - MESSAGE FILE UNIT NUMBER
C NI - NUMBER OF GRID POINTS IN THE X DIRECTION
C NJ - NUMBER OF GRID POINTS IN THE Y DIRECTION
C NVERSN - PROBLEM IDENTIFIER
55 C NTYPE - DETERMINES GRID POINT ORDERING TO BE
C USED. SEE NORDER FOR DETAILS
C
REAL A(IAJ),B(NN),D(NN),ATYPE(4)
INTEGER INI(IAI),INJ(IAJ),IK(NN,2),IW(NN)
60 C COMMON/MA31J/LROW,LCOL,NCP,IPD,DD
COMMON/MA31I/DD,LP,MP
COMMON/MA31M/NI,NJ,NVERSN,NTYPE
C DATA ATYPE/7HNATURAL,7HLINE RB,8HPOINT RB,8H2LINE RB/
WRITE(MP,2)
2 FORMAT(1I1H GENA START)
C INITIALIZE DATA
70 C DO 5 I=1,ND
IK(I,1)=0
IK(I,2)=0
IW(I)=0
75 5 CONTINUE
C CALL TIME(AT)
CALL DATE(AD)
CALL SECOND(TIM1)
80 C NNAT=0
NZ=0
C PROCESS GRID POINTS IN NATURAL ORDER
85 C PERFORMING THE DISCRETIZATION
DO 100 J=1,NJ
DO 90 I=1,NI
NNAT=NNAT+1
90 N=NORDER(NTYPE,I,J,NNAT)
D(N)=4.0
B(N)=0.0
IF ((I.EQ.1).OR.(I.EQ.NI)) B(N)=B(N)+1.0
IF ((J.EQ.1).OR.(J.EQ.NJ)) B(N)=B(N)+1.0
95 C
IF (I.EQ.NI) GO TO 50
NZ=NZ+1
A(NZ)=-1.0
NT=NORDER(NTYPE,I+1,J,NNAT+1)
CALL ISTORE(N,NT,INI,INJ,IAI,IK,ND,NZ)
50 CONTINUE
IF (J.EQ.NJ) GO TO 90
NZ=NZ+1
A(NZ)=-1.0
NT=NORDER(NTYPE,I,J+1,NNAT+1)
CALL ISTORE(N,NT,INI,INJ,IAI,IK,ND,NZ)
90 CONTINUE
100 CONTINUE

C INITIALIZE IW(I) TO POINT JUST BEYOND WHERE THE
C LAST COMPONENT OF ROW I WILL BE STORED
C
KI=1
DO 200 I=1,ND
KI=KI+IK(I,1)
200 IW(I)=KI

C REORDER BY ROWS USING IN-PLACE SORT ALGORITHM
C
CALL MA31E(INI,INJ,NZ,IW,ND,A)

C REINITIALIZE IW(I) TO POINT TO THE BEGINNING OF ROW I
C
KK=1
DO 210 IR=1,ND
IW(IR)=KK
210 KK=KK+IK(IR,1)
DO 220 I=1,NZ
220 INI(I)=IABS(INI(I))
C
CALL SECOND(TIM2)
TIMD = TIM2 - TIM1

135 C OUTPUT STATISTICS
C WRITE(LP,250) TIMD
250 FORMAT(13H GENA TIME = ,F6.3,4H SEC)

140 WRITE(LP,260) NVERS
260 FORMAT(11H VERSION = ,I2)
WRITE(LP,265) ATYPE(NTYPE+1)
265 FORMAT(14H MATRIX A HAS ,A10,9H ORDERING)
WRITE(LP,270) AD, AT
270 FORMAT(18H DATE GENERATED = ,A10,A10)
WRITE(LP,280) ND, NZ
280 FORMAT(6H ND =,I4,6H NZ =,I4)
WRITE(LP,290)
290 FORMAT(9H GENA END)

150 C RETURN
END
SUBROUTINE ISTORE(N,NJ,INI,INJ,IAI,IK,NP,NZ)
C
C INTEGER INI(IAI),INJ(IAI),IK(NP,2)
C SUBROUTINE USED TO UPDATE ROW AND COLUMN COUNTS
C IF (N.GT.NJ) GO TO 10

160 INI(NZ)=N
IK(N,1)=IK(N,1)+1
INJ(NZ)=NJ
IK(NJ,2)=IK(NJ,2)+1
GO TO 20

165 10 INI(NZ)=NJ
IK(NJ,1)=IK(NJ,1)+1
INJ(NZ)=N
IK(N,2)=IK(N,2)+1
CONTINUE
20 RETURN
END

FUNCTION NORDER(NTYPE,I,J,N)
C
C SUBROUTINE TO PERMUTE AN ELEMENT FROM NATURAL ORDERING TO
C ONE OF THE OTHER ORDERING SCHEMES
C
C NTYPE = 0  NATURAL ORDERING
C = 1 LINE RED/BLACK ORDERING
C = 2 POINT RED/BLACK ORDERING
C = 3 2 LINE RED/BLACK ORDERING

C INTEGER PTRB,OFFST(4)
COMMON/MA31M/NI,NJ,NVERS,N,NTYP
DATA OFFST/32,512,496,496/
DATA NATURL,LINRB,PTRB,L2RB/0,1,2,3/

10 C

NTEM=N
C
IF (NTYPE.EQ.NATURL) GO TO 100
C
20 IMOD=MOD(I+1,2)
JMOD=MOD(J+1,2)
C
C DETERMINE IF LINE RED-BLACK ORDERING REQUESTED
C
25 IF (NTYPE.NE.LINRB) GO TO 20
NTEM=J+((I-1)/2)*NJ
IF (IMOD.EQ.0) GO TO 15
NTEM=NTEM+OFFST(NVERS+N+1)
15 CONTINUE
GO TO 100
20 CONTINUE
C
C DETERMINE IF POINT RED-BLACK ORDERING REQUESTED
C
35 IF (NTYPE.NE.PTRB) GO TO 30
NTEM=(N+1)/2
IF (IMOD.EQ.JMOD) GO TO 25
NTEM=NTEM+OFFST(NVERS+N+1)
25 CONTINUE
GO TO 100
30 CONTINUE
C
C DETERMINE IF TWO LINE RED-BLACK ORDERING REQUESTED
C
45 NTEM=J+IMOD*NJ+((I-1)/4)*NJ*2
NIMOD=MOD((I+1)/2,2)
IF (NIMOD.EQ.1) GO TO 100
NTEM=NTEM+OFFST(NVERS+N+1)
C
50 CONTINUE
NORDER=NTEM
RETURN
END
SUBROUTINE FACTOR(NN,NZA,A,INI,INJ,IAI,IAJ,IK,IW,*W,C,FILL,EUSE)

SUBROUTINE TO CALCULATE THE PRECONDITIONING MATRIX USING THE MODIFIED HARWELL ROUTINE MA31C. THIS SUBROUTINE PERFORMS THE SAME FUNCTIONS AS THE HARWELL ROUTINE MA31A. SEE DESCRIPTION OF THE HARWELL MA31 PACKAGE FOR MORE DETAILS.

REAL A(IAJ),W(NN,3)
INTEGER IK(NN,4),IW(NN,4),INI(IAI),INJ(IAJ),OPTION(6)
LOGICAL FILL,EUSE

COMMON/MA31I/DD,LP,MP
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31K/NURL,NUCL,NUAL
COMMON/MCOMM3/OPTION
COMMON/MA31M/NI,NJ,NVERSN,NTYPE

CALL SECOND(TIM1)

NZ=NZA
NZP1=NZA+1
IAJ1=IAJ-NZA

SAVE ROW INDEX FILE IK(K,1)

DO 5 K=1,NN
   IK(K,4)=IK(K,1)
5   IF (OPTION(4).EQ.0) GO TO 18

IF (OPTION(4).EQ.0) GO TO 18

ELIMINATE THOSE ELEMENTS NOT TO BE USED IN THE INCOMPLETE FACTORIZATION AS DETERMINED BY THE FUNCTION EUSE.

NZ1=NZ+1
KK=NZ
DO 12 K=1,NZ
   I=INI(K)
   J=INJ(K)
   IF (EUSE(I,J)) GO TO 10
   IK(I,1)=IK(I,1)-1
   IK(J,2)=IK(J,2)-1
12   GO TO 12

CONTINUE

KK=KK+1
A(KK)=A(K)
INJ(KK)=J

CONTINUE

C REBUILD THE START OF ROW I FILE IW(I,1)
C
KI=NZ1
DO 14 K=1,ND
IW(K,1)=KI
KI=KI+IK(K,1)
14 CONTINUE
C
NZ=KK-NZA
CALL SECOND(TIM2)
C
IF (NZ.NE.0) GO TO 18
C
SPECIAL CASE OF DIAGONAL SCALING
C
DO 15 I=1,ND
W(I,2)=W(I,1)
IK(I,2)=1
15 CONTINUE
C
LROW=0
LCOL=0
IFLAG=0
GO TO 45
C
CONSTRUCT COLUMN FILE IW(I,2) TO POINT JUST BEYOND WHERE THE
LAST COMPONENT OF COLUMN I WILL BE STORED
C
KJ=IAI-NZ+1
DO 20 I=1,ND
KJ=KJ+IK(I,2)
IW(I,2)=KJ
20 CONTINUE
C
CONSTRUCT COLUMN FILE IN HIGH ORDER PART OF INI
C
DO 30 IR=1,ND
KPP=IW(IR,1)
KLL=KPP+IK(IR,1)-1
IF (KPP.GT.KLL) GO TO 30
DO 25 K=KPP,KLL
J=INJ(K)
KR=IW(J,2)-1
IW(J,2)=KR
25 CONTINUE

INI(KR)=IR
CONTINUE
30 CONTINUE
C
C TRANSFER INPUT MATRIX TO TAIL END OF ARRAY A
100 C AND MODIFY INJ TO REFLECT THE MOVE
C

        NUAL=IAJ+1
        DO 40 II=1,ND
           I=ND-II+1
105          W(I,2)=W(I,1)
           KP=IW(I,1)
           KL=KP+IK(I,1)-1
           IF (KP.GT.KL) GO TO 38
           DO 35 KK=KP,KL
40          K=KP+KK
           NUAL=NUAL-1
           A(NUAL)=A(K)
           INJ(NUAL)=INJ(K)
35          CONTINUE
115 38 IW(I,1)=NUAL-NZA
40 CONTINUE
C
C INITIALIZE COMMON MA31J AND MA31K VARIABLES
C
120        LCOL=NZ
        LROW=NZ
125        NURL=0
        NUCL=IW(1,2)
        NUAL=NUAL-NZA
125        IFLAG=0
        NPC=0
C
C CALL SECOND(TIM2)
C
130 C PERFORM THE FACTORIZATION
C
        CALL MA31C(ND,NZ,W(1,2),A(NZP1),INI,INJ(NZP1),
125          1IAI,IAJ1,IK,IW,IW(1,3),W(1,3),IFLAG,C)
C
135 45 CALL SECOND(TIM3)
C
C RESTORE INI
C
        KP=1
140        DO 56 I=1,ND
           KL=KP+IK(I,4)-1
           IF (KP.GT.KL) GO TO 56
           DO 55 K=KP,KL
55          K=KP+KK
155          NUAL=NUAL-1
           A(NUAL)=A(K)
           INJ(NUAL)=INJ(K)
55          CONTINUE
155 56 IW(I,1)=NUAL-NZA
56 CONTINUE
C
107

55 INI(K)=I
145 56 KP=KL+1
C
C OUTPUT STATISTICS ON THE FACTORIZATION
C
WRITE(LP,58)
150 58 FORMAT(25HRESULTS OF FACTORIZATION)
WRITE(LP,60) (OPTION(I),I=1,6),C
60 FORMAT(1H0,9HOPTION = 6I1,2X,4HC = ,F9.5)
WRITE(LP,65) IFLAG
65 FORMAT(9H IFLAG = ,I3)
155 C
C TPD - TIME REQUIRED TO PREPARE DATA ARRAYS
C PRIOR TO CALLING MA31C.
C TD - TIME REQUIRED BY MA31C TO PERFORM THE
C FACTORIZATION.
160 C TDT - TOTAL TIME REQUIRED BY SUBROUTINE FACTOR.
C
TDT=TIM3-TIM1
TPD=TIM2-TIM1
TD=TIM3-TIM2
165 C
WRITE(LP,70) TDT,TPD,TD
70 FORMAT(7H TDT = ,F6.3,7H TPD = ,F6.3,6H TD = ,F6.3)
C
WRITE(LP,85) NTYPE,NVERN
170 85 FORMAT(9H NTYPE = ,I2,2X,10HVERSION = ,I2)
WRITE(LP,90) LROW
90 FORMAT(21HNUM ELEMENTS IN L = ,I4)
WRITE(LP,100) ND,NZA
100 FORMAT(6H ND = ,I3,7H NZA = ,I4)
175 C
150 CONTINUE
RETURN
END
LOGICAL FUNCTION EUSE(I,J)

C EUSE1
C -----
5 C ELIMINATES ALL OFF-DIAGONAL ELEMENTS.
C USED FOR DIAGONAL SCALING
C
EUSE=.FALSE.
10 RETURN
END

LOGICAL FUNCTION EUSE(I,J)

C EUSE 2
C -----
5 C USED DURING BLOCK DIAGONAL SCALING (BDS).
C KEEPS ONLY THOSE ELEMENTS IN THE TRI-DIAGONAL
C PORTION OF THE MATRIX A.
C
EUSE=.FALSE.
10 IF (IABS(J-I).LE.1) EUSE=.TRUE.
RETURN
END

LOGICAL FUNCTION EUSE(I,J)

C EUSE3
C -----
5 C USED TO GENERATE THE LINE RED/BLACK REDUCED BLOCK FORMAT
C
EUSE=.FALSE.
ID=IABS(J-I)
10 IF ((ID.LE.1).OR.(ID.EQ.32)) EUSE=.TRUE.
RETURN
END
LOGICAL FUNCTION EUSE(I,J)

C EUSE4
C
5 C USED TO GENERATE THE POINT RED/BLACK
C REDUCED BLOCK MATRIX.
C
      EUSE=.FALSE.
      MI=(I-1/8)+1
      MJ=((J-1)/8)+1
      MD=IABS(MJ-MI)
      IF (MD.EQ.4) EUSE=.TRUE.
      RETURN
END

LOGICAL FUNCTION EUSE(I,J)

C EUSE5
C
5 C USED TO GENERATE THE 2 LINE RED/BLACK REDUCED BLOCK FORMAT
C
      EUSE=.FALSE.
      IF (IABS(J-I).LE.8) EUSE=.TRUE.
      RETURN
END

SUBROUTINE MA31C(N,NZ,D,A,INI,INJ,IAI,IAJ,IK, 
                  IP,IW,W,IFLAG,C)

C MA31C IS PART OF THE HARWELL MA31 PACKAGE.
5 C SEE ROUTINE MA31A FOR DETAILS.
C MODIFIED TO CALCULATE A WIDER VARIETY OF INCOMPLETE
C CHOLESKY FACTORIZATION.
C
C EXTERNAL FILL
C
10 REAL A(IAJ),W(N),D(N)
 INTEGER IP(N,2),OPTION(6)
 LOGICAL CHANGE,FILL
 INTEGER IK(N,3),IW(N,2),INI(IAI),INJ(IAJ)
 COMMON/MA31I/DD,LP,MP
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31K/NURL,NUCL,NUAL
COMMON/MCOMM3/OPTION
COMMON/MA31L/EPSTOL,U

C

20 C OPTION DETERMINES HOW THE FACTORIZATION WILL BE DONE
C OPTION(1) = 0 - NATURAL ORDER FACTORIZATION
C = 1 - MINIMUM DEGREE FACTORIZATION
C OPTION(2) = 0 - FUNCTION FILL USED TO CONTROL FILL-INS
C = 1 - DROP TOLERANCE C USED TO CONTROL FILL-INS
C OPTION(3) = 0 - NO DIAGONAL SCALING USED
C = 1 - DIAGONAL ELEMENTS SCALED BY 1 + ABS(C)/FLOAT(N)
C OPTION(4) = - NOT USED HERE
C OPTION(5) = 0 - DIAGONAL MODIFICATION NOT CONSIDERED
C = 1 - DIAGONAL MODIFICATION CORRESPONDING TO THE DROPPED FILL-INS IS PERFORMED
C OPTION(6) = - NOT USED HERE

C IP(I,1),IP(I,2) POINT TO THE START OF ROW/COLUMN I.
C IK(I,1),IK(I,2) HOLD THE NUMBER OF NONZEROES IN ROW/COLUMN I
C OF THE LOWER TRIANGULAR PART OF A.
C DURING THE MAIN BODY OF THIS SUBROUTINE THE VECTORS IK(*,3),IW(*,1) AND IW(*,2) ARE USED TO HOLD DOUBLY
C LINKED LISTS OF ROWS THAT HAVE NOT BEEN PIVOTAL AND HAVE EQUAL NUMBER OF NONZEROES.
C IK(I,3) HOLD FIRST ROW/COLUMN TO HAVE I NONZeros OR ZERO IF THERE ARE NONE.
C IW(I,1) HOLD ROW/COLUMN NUMBER OF ROW/COLUMN PRIOR TO ROW I IN ITS LIST OR ZERO IF NONE.
C IW(I,2) HOLD ROW/COLUMN NUMBER OF ROW/COLUMN AFTER ROW I IN ITS LIST OR ZERO IF NONE.

C DATA ZERO,ONE,CMAX/0.0,1.0,1.0E20/

C INITIALIZE IK(*,3) AND LOCAL VARIABLES.
CHANGE=.TRUE.
IF (C.LE.ZERO) CHANGE=.FALSE.
NZO=NZ
IPD=N
ALFA=1.0/0.90
B1=-.03
B2 = .03
NFILL = IAJ - NZ0 - N
MCL = LCOL
CO = 0
IF (OPTION(3).NE.0) CO = ABS(C)/FLOAT(N)
C = C * 2
DO 5 I = 1, N
D(I) = (I + CO) * D(I)
5 IK(I, 3) = 0

C Set up linked lists of rows/columns with equal number of non-zeros.

IF (OPTION(1).NE.0) GO TO 9
DO 8 I = 1, N
IW(I, 1) = I - 1
IW(I, 2) = I + 1
80 8 CONTINUE
IW(N, 2) = 0
IK(1, 3) = 1
GO TO 15

CONTINUE
DO 10 I = 1, N
NZI = IK(I, 1) + IK(I, 2) + 1
IN = IK(NZI, 3)
IK(NZI, 3) = I
IW(I, 2) = IN
IW(I, 1) = 0
10 IF (IN .NE. 0) IW(IN, 1) = I
15 CONTINUE

C Start the elimination loop
DO 180 IIP = 1, N

C Search rows with NRJP nonzeros.
DO 20 NRJP = 1, N
JP = IK(NRJP, 3)
IF (JP .GT. 0) GO TO 25
20 CONTINUE

C Row JP is used as pivot.
105 C
C Remove rows/columns involved in elimination from ordering vectors.
C
25 DO 45 L = 1, 2
KPP = IP(JP, L)
DO 40 K = KPP, KLL
IF (L.EQ.2) GO TO 27
J = INJ(K)
GO TO 28
27 J = INI(K)
28 IL = IW(J, 1)
IN = IW(J, 2)
IW(J, 2) = -1
IF (OPTION(1).EQ.0) GO TO 40
IF (IN.LT.0) GO TO 40
IF (IL.EQ.0) GO TO 30
IW(IL, 2) = IN
GO TO 35
30 NZ = IK(J, 1) + IK(J, 2) + 1
IK(NZ, 3) = IN
35 IF (IN.GT.0) IW(IN, 1) = IL
40 CONTINUE
45 CONTINUE
C C REMOVE JP FROM ORDERING VECTORS
IL = IW(JP, 1)
IN = IW(JP, 2)
IW(JP, 2) = -10
IF (OPTION(1).NE.0) GO TO 54
IK(1, 3) = JP + 1
GO TO 55
54 CONTINUE
IF (IN.LT.0) GO TO 55
NZ = IK(JP, 1) + IK(JP, 2) + 1
IK(NZ, 3) = IN
IF (IN.GT.0) IW(IN, 1) = IL
55 CONTINUE
C C STORE PIVOT.
IW(JP, 1) = IP
C COMPRESS ROW FILE IF NECESSARY.
C
IF (LROW + IK(JP, 1) + IK(JP, 2) .GT. IAJ - N) C = CMAX
IF (NURL + IK(JP, 1) + IK(JP, 2) .LT. NUAL) GO TO 60
CALL MA31D(A, INJ, IAJ, N, IK, IP, .TRUE.)
60 KP = IP(JP, 1)
KL = IK(JP, 1) + KP - 1
IP(JP, 1) = NURL + 1
IF (KP.GT.KL) GO TO 90
C C REMOVE JP FROM COLUMNS CONTAINED IN THE PIVOT ROW.
DO 85 K=KP,KL
  160 J=INJ(K)
  KPC=IP(J,2)
  NZ=IK(J,2)-1
  IK(J,2)=NZ
  KLC=KPC+NZ
  IF (KLC.GT.KPC) GO TO 65
  INI(KPC)=0
  GO TO 80
65 DO 70 KC=KPC,KLC
  IF (JP.EQ.INI(KC)) GO TO 75
  170 70 CONTINUE
  75 INI(KC)=INI(KLC)
   INI(KLC)=0
  80 LCOL=LCOL-1
       NURL=NURL+1
  175 INJ(NURL)=J
       A(NURL)=A(K)
  85 INJ(K)=0
C
C TRANSFORM COLUMN PART OF PIVOT ROW TO THE ROW FILE.
180  90 KP2=IP(JP,2)
    KL2=IK(JP,2)+KP2-1
    IF (KP2.GT.KL2) GO TO 100
    DO 95 K=KP2,KL2
    NURL=NURL+1
  185 LCOL=LCOL-1
    I=INI(K)
    KPR=IP(I,1)
    KLR=KPR+IK(I,1)-1
    DO 92 KR=KPR,KLR
    IF (JP.EQ.INJ(KR)) GO TO 93
  190  92 CONTINUE
  93 INJ(KR)=INJ(KLR)
    A(NURL)=A(KR)
    A(KR)=A(KLR)
  195 INJ(KLR)=0
    IK(I,1)=IK(I,1)-1
    INJ(NURL)=I
  95 INI(K)=0
  100 NZC=IK(JP,1)+IK(JP,2)
  200 INJ(JP,1)=NZC
       IK(JP,2)=NZC
C
C UNPACK PIVOT ROW AND CONTROL DIAGONAL VALUE.
  KP=IP(JP,1)
  205 KL=KP+NZC-1
    CO=EPSSTOL*U
IF (KP.GT.KL) GO TO 102
DO 101 K=KP, KL
AA=A(K)
CO=AMAX1(CO,ABS(AA))
J=INJ(K)
W(J)=AA
101 CONTINUE
102 DJP=D(JP)
215 IF (DJP.GT.CO/U) GO TO 103
IFLAG=2
IF (MP.GT.0) WRITE(MP,250) JP
250 FORMAT(/44H+ WARNING MODIFICATION OF ZERO OR NEGATIVE,
148H DIAGONAL ENTRY HAS BEEN PERFORMED IN LOCATION,17)
220 D(JP)=CO
IF (CO.EQ.EPSTOL*U) D(JP)=ONE
103 IF (KP.GT.KL) GO TO 179
C PERFORM ROW OPERATIONS.
225 DO 170 NC=1,NZC
KC=IP(JP,1)+NC-1
IR=INJ(KC)
AL=A(KC)/D(JP)
C COMPRESS ROW FILE IF NECESSARY.
IF (LROW+IK(IR,1)+IK(JP,1).GT.IAJ-N) C=CMAX
IF (NURL+IK(IR,1)+IK(JP,1).LT.NUAL) GO TO 105
CALL MA31D(A,INJ,IAJN,IK,IP,.TRUE.)
105 KR=IP(IR,1)
235 KRL=KR+IK(IR,1)-1
IF (KR.GT.KRL) GO TO 120
C SCAN THE OTHER ROW AND CHANGE SIGN IN IW FOR EACH COMMON COLUMN NUMBER.
240 DO 115 KS=KR,KRL
J=INJ(KS)
IF (IW(J,2).NE.-1) GO TO 115
IW(J,2)=1
A(KS)=A(KS)-AL*W(J)
245 115 CONTINUE
C SCAN PIVOT ROW FOR FILLS.
120 DO 165 KS=KP, KL
J=INJ(KS)
250 C ONLY ENTRIES IN THE UPPER TRIANGULAR PART ARE CONSIDERED.
IF (J.LT.IR) GO TO 165
IF(IW(J,2).EQ.1) GO TO 165
AA=-AL*W(J)
255 IF(IR.NE.J) GO TO 122
   D(IR)=D(IR)+AA
   GO TO 165
122 IF (OPTION(2).NE.0) GO TO 123
   IF (FILL(IR,J)) GO TO 124
260 IF (OPTION(5).EQ.0) GO TO 165
   D(J)=D(J)+AA
   D(IR)=D(IR)+AA
   GO TO 165
123 IF (AA*AA.GT.C*ABS(D(IR)*D(J))) GO TO 124
265 IF (OPTION(5).EQ.0) GO TO 165
   D(J)=D(J)+AA
   D(IR)=D(IR)+AA
   GO TO 165
124 LROW=LROW+1
270 IK(IR,1)=IK(IR,1)+1
   C IF POSSIBLE PLACE THE NEW ELEMENT NEXT TO THE PRESENT ENTRY.
   C
   C TRY IF THERE IS ROOM AT THE END OF THE ENTRY.
275 IF (KR.GT.KRL) GO TO 130
   IF (KRL.EQ.IAJ) GO TO 125
   IF (INJ(KRL+1).NE.0) GO TO 125
   KRL=KRL+1
   INJ(KRL)=J
280 A(KRL)=AA
   GO TO 133
   C
   C TRY IF THERE IS ROOM AHEAD OF PRESENT ENTRY.
125 IF (KR.NE.NUAL) GO TO 126
285 NUAL=NUAL-1
   GO TO 127
126 IF (INJ(KR-1).NE.0) GO TO 128
127 KR=KR-1
   IP(IR,1)=KR
290 INJ(KR)=J
   A(KR)=AA
   GO TO 133
   C
   C NEW ENTRY HAS TO BE CREATED.
295 DO 129 KK=KR,KRL
   NUAL=NUAL-1
   INJ(NUAL)=INJ(KK)
   A(NUAL)=A(KK)
   129 INJ(KK)=0
300 C
   C ADD THE NEW ELEMENT.
130 NUAL=NUAL-1
INJ(NUAL)=J
A(NUAL)=AA
IP(IR,1)=NUAL
KR=NUAL
KRI=KR+IK(IR,1)-1
C
C CREATE FILL IN COLUMN FILE.
310 133 NZ=IK(J,2)
K=IP(J,2)
KLI=K+NZ-1
LCOL=LCOL+1
C
C IF POSSIBLE PLACE NEW ELEMENT AT THE END OF PRESENT ENTRY.
315 IF (NZ.EQ.0) GO TO 140
IF (KLI.EQ.KAI) GO TO 137
IF (INI(KLI+1).NE.0) GO TO 137
INI(KLI+1)=IR
GO TO 160
C
C IF POSSIBLE PLACE ELEMENT AHEAD OF PRESENT ENTRY.
137 IF (K.NE.NUCL) GO TO 138
IF (NUCL.EQ.1) GO TO 140
320 NUCL=NUCL-1
GO TO 139
138 IF (INI(K-1).NE.0) GO TO 140
139 K=K-1
INI(K)=IR
IP(J,2)=K
GO TO 160
C
C NEW ENTRY HAS TO BE CREATED.
140 IF (NZ+1.LT.NUCL) GO TO 145
C
C COMPRESS COLUMN FILE IF THERE IS NOT ROOM FOR NEW ENTRY.
335 C
C COMpress COLUMN FILE IF THERE IS NOT ROOM FOR NEW ENTRY.
335 IF (LCOL+NZ+2.GE.IAI) C=CMAX
CALL MA31D(A,INI,IAI,N,IK(1,2),IP(1,2),.FALSE.)
K=IP(J,2)
340 KLI=K+NZ-1
C
C TRANSFER OLD ENTRY INTO NEW.
145 IF (K.GT.KLI) GO TO 155
DO 150 KK=K,KLI
345 NUCL=NUCL-1
INI(NUCL)=INI(KK)
150 INI(KK)=0
C
C ADD THE NEW ELEMENT.
350 155 NUCL=NUCL-1
INI(NUCL)=IR
IP(J,2)=NUCL
160 IK(J,2)=NZ+1
165 IW(J,2)=1
355 170 CONTINUE

C UPDATE ORDERING ARRAYS.
DO 172 K=KP,KL
   J=INJ(K)
   W(J)=0.
   A(K)=A(K)/D(JP)
   IF (OPTION(1).EQ.O) GO TO 171
   NZ=IK(J,1)+IK(J,2)+1
   IN=IK(NZ,3)
365 IW(J,2)=IN
   IW(J,1)=0
   IK(NZ,3)=J
   IF (IN.NE.0) IW(IN,1)=J
   GO TO 172
370 171 IW(J,2)=J+1
   IW(J,1)=J-1
172 CONTINUE
   IF (OPTION(1).EQ.O) IW(N,2)=0
   MCL=MAX0(MCL,LCOL)
375 PIVT=FLOAT(IIP)/FLOAT(N)

C GIVE WARNING IF AVAILABLE SPACE IS USED TOO EARLY.
   IF (C.NE.CMAX) GO TO 175
   IF (IPD.LT.IIP) GO TO 179
380 IPD=IIP
   IF (PIVT.GT.9) GO TO 179
   IFLAG=4
   IF (MP.GT.0) WRITE(MP,260) IIP
   GO TO 179
385 260 FORMAT(/'WARNING AVAILABLE SPACE USED AT PIVOT STEP',I7)

C CHANGE C IF NECESSARY.
175 IF (.NOT. CHANGE) GO TO 179
   PFILL=FLOAT(LROW-NZ0)/FLOAT(NFILL)
   IF (PIVT.GT.0.9) GO TO 179
   IF (PFILL.LT.ALFA*PIVT+B1) GO TO 176
   IF (PFILL.LT.ALFA*PIVT+B2) GO TO 179
   C=2.25*C
176 ALFA=(1.0-PFILL)/(0.9-PIVT)
390 B1=PFILL-PIVT*ALFA-0.03
   B2=B1+0.06

C IF THE MATRIX IS FULL THEN STOP THE SPARSE ANALYZE.
179 NR=N-IIP
400    LFULL=NR*(NR-1)/2
        LFULLD=IFIX(DD*FLOAT(LFULL))
        IF (LCOL.GE.LFULDD.AND.NURL+LFULL.LT.IAJ) GO TO 185
180 CONTINUE
C
405 C
C ELIMINATION LOOP TERMINATES
C AFTER DEVIATION WE FACTORIZE THE REMAINING FULL MATRIX.
185 IPD=IIP
        C=SQR(T(C)
410    LCOL=MCL
        IF (.NOT. CHANGE) C=-C
C
C THE ORDER OF THE FULL MATRIX IS NR.
C LOOP THROUGH ROWS IN THE ACTIVE MATRIX AND STORE
415 C
C ROW NUMBERS IN INI.
    KK=0
    DO 197 I=1,NR
        JP=IK(I,3)
    194 IF (JP)196,196,195
    195 KK=KK+1
        INI(KK)=JP
        JP=IW(JP,2)
        GO TO 194
    196 IF (KK.EQ.NR) GO TO 198
197 CONTINUE
C
C MAKE A SORT OF ROWNUMBERS IN INI.
198 IF (NR.EQ.1) GO TO 200
    NRMl=NR-1
430    DO 199 I=1,NRMl
    198 IF (NR.EQ.1) GO TO 200
        J1=I+1
        DO 199 J=J1,NR
            IF (INI(J).GT.INI(I)) GO TO 199
    199 JJ=INI(I)
35      INI(I)=INI(J)
36      INI(J)=JJ
35      INI(I)=INI(J)
            INI(J)=JJ
345 CONTINUE
200 DO 201 I=1,NR
201 IW(I,1)=(IPD+I)
C
C MAKE AN ORDERED LIST OF THE PIVOTS.
    DO 202 I=1,N
        IR=IW(I,1)
    201 IK(IR,2)=I
C
C MOVE FULL MATRIX TO THE FRONT AND ORDER.

IPDP1 = IPD + 1
NM1 = N - 1

450 IF (IPDP1 GT NM1) GO TO 245
DO 215 IIP = IPDP1, NM1
JP = IK(IIP, 2)
KP = IP(JP, 1)
KL = KP + IK(JP, 1) - 1

455 C MOVE ROW JP TO W.
IF (KP GT KL) GO TO 204
DO 203 K = KP, KL
J = INJ(K)

460 INJ(K) = 0
203 W(J) = A(K)
C
C COMPRESS FILE IF NECESSARY.

204 IF (NURL + N - IIP LT NUAL) GO TO 205

465 CALL MA31D(A, INJ, IAJ, N, IK, IP, .TRUE.)

205 IP(JP, 1) = NURL + 1
IK(JP, 1) = N - IIP
C
C MOVE ROWS AND COLUMN INDICES INTO PIVOTAL ORDER.

470 IIP1 = IIP + 1
DO 210 I = IIP1, N
J = IK(I, 2)
NURL = NURL + 1
A(NURL) = W(J)

475 INJ(NURL) = J
210 W(J) = ZERO
215 CONTINUE
LROW = NURL
C
C FACTORIZE THE FULL MATRIX.

480 C DO 240 IIP = IPDP1, NM1
JP = IK(IIP, 2)
KPI = IP(JP, 1)
IPI = IIP + 1

485 IF (IPI EQ N) GO TO 235
C
C LOOP THROUGH THE OTHER ROW
DO 230 J = IPI, NM1
JJ = IK(J, 2)

490 KPJ = IP(JJ, 1)
KLJ = KPJ + IK(JJ, 1) - 1
AL = A(KPI) / D(JP)
D(JJ) = D(JJ) - AL * A(KPI)
KK = KPI + 1
495 DO 220 K=KPJ,KLJ
  A(K)=A(K)-AL*A(KK)
220 KK=KK+1
C C STORE FACTOR AND PROCEED TO NEXT ROW.
500 A(KPI)=AL
  KPI=KPI+1
230 CONTINUE
C C MODIFY LAST DIAGONAL ENTRY
505 235 JJ=IK(N,2)
  AL=A(KPI)/D(JP)
  D(JJ)=D(JJ)-AL*A(KPI)
  A(KPI)=AL
240 CONTINUE
510 245 CONTINUE
RETURN
END

LOGICAL FUNCTION FILL(I,J)
C C FILL1
C -----
5 C C USED WHEN NO FILL-INS ARE TO BE KEPT
C FILL=.FALSE.
RETURN
10 END

LOGICAL FUNCTION FILL(I,J)
C C FILL2
C -----
5 C C ALLOWS ONE DIAGONAL OF FILL-INS TO BE KEPT
C ADJACENT TO THE OUTER DIAGONAL.
C FILL=.FALSE.
IF (IABS(J-I).GE.7) FILL=.TRUE.
RETURN
END
LOGICAL FUNCTION FILL(I,J)

C FILL3
C-----

C ALLOWS THREE DIAGONAL OF FILL-INS TO BE KEPT.
C ONE ADJACENT TO THE INNER DIAGONAL AND TWO
C ADJACENT TO THE OUTER DIAGONAL.
C

FILL=.FALSE.
ID=IABS(J-I)
IF (ID.LE.2).OR.(ID.GE.6)) FILL=.TRUE.
RETURN
END

SUBROUTINE SOLVE(NN,NZ,A,INJ,IAJ,IAI,W,IK,B,W1)

C SUBROUTINE WHICH SOLVES
C THE
C LINEAR SYSTEM USING
C HARWELL'S PRECONDITIONED CONJUGATE GRADIENT
C ROUTINE MA31F.

C INPUT PARAMETERS
C----------------

C NN   - ORDER OF MATRIX A.
C NZ   - NUMBER OF NON-ZERO ELEMENTS IN THE UPPER
C        TRIANGULAT PORTION OF MATRIX A.
C A    - ARRAY OF LENGTH IAJ CONTAINING THE NON-ZERO
C        OFF-DIAGONAL ELEMENTS OF THE UPPER TRIANGULAR
C        PORTION OF MATRIX A IN THE FIRST NZ LOCATIONS
C        IN ROW ORDER. LOCATIONS NZ+1,...,NZ+LROW
C INJ   - ARRAY OF LENGTH IAJ CONTAINING THE COLUMN
C        INDICES OF THE CORRESPONDING ENTRY IN ARRAY A.
C        (IE. INJ(K) CONTAINS THE COLUMN INDICE FOR
C        ENTRY A(K), K=1,...,NZ+LROW).
C INI   - ARRAY OF LENGTH NZ CONTAINING THE ROW INDICES
C        OF THE CORRESPONDING ENTRY IN ARRAY A.
C        (IE. INI(K) CONTAINS THE ROW INDICE FOR
C        ENTRY A(K), K=1,...,NZ).
C IAJ   - SIZE OF ARRAYS INJ AND A.
C B     - CONTAINS THE RIGHHAND SIDE OF THE SYSTEM.
C W     - ARRAY OF LENGTH 3*NN IN WHICH LOCATIONS
1,...,NN contain the diagonal elements of
matrix \(\mathbf{A}\) and locations \(\text{NN}+1,\ldots,2\times\text{NN}\) contain
the inverse of the diagonal elements of matrix \(\mathbf{C}\).
The remaining \(\text{NN}\) locations are work space.

\(w_l\) — array of length \(3\times\text{NN}\) used as work space.

Output parameters

\(b\) — the solution vector

Common block parameters

\(\text{lcol}, ncp, ipd, dd\) — not used
\(\text{lrow}\) — number of non-zero elements in upper
triangular portion of the preconditioning
matrix \(\mathbf{C}\).
\(\text{nd}\) — order of matrix \(\mathbf{A}\) and \(\mathbf{C}\).
\(\text{lp}\) — output file device number.
\(\text{mp}\) — message file device number.
\(\text{mits}\) — maximum number of iterations to be attempted.
\(\text{eps1}\) — desired accuracy of \(\mathbf{M}\).

Internal variables

\(\text{niter}\) — on entry to \text{MA31F} it contains the maximum
number of iterations to be attempted. On
return from \text{MA31F} it contains the number
of iterations performed.
\(\text{eps}\) — on entry to \text{MA31F}, \text{eps}(1) contains the
desired accuracy for \(\mathbf{M}\). On return,
\text{eps}(i) contains the value of \(\mathbf{M}\) after
iteration \(i-1\).

\(\text{real}\) \(a(iaj), w(nn, 3), w_l(nn, 3), \text{eps}(150), b(nn)\)
\(\text{integer}\) \(\text{ini}(iai), \text{inj}(iaj), \text{ik}(nn, 2)\)

\(\text{common/ma31n/mits, eps1}\)
\(\text{common/ma31i/dd, lp, mp}\)
\(\text{common/ma31j/lrow, lcol, ncp, nd, ipd}\)

\(\text{write(mp, 5)}\)

\(\text{format(12h start solve)}\)
\(\text{iaj1=iaj-nz}\)
\(\text{nz1=nz+1}\)
WRITE(LP,292)
80 292 FORMAT(3HORESULTS OF PRECONDITIONED CG ROUTINE)
   IFLAG=0
   NITER=MITS
   EPS(1)=EPS1
   CALL SECOND(STRTIM)
   CALL MA31F(ND,NZ,A,W,INI,INJ,IAJ1,A(NZ1),W(1,2),
   1INJ(NZ1),IK,B,W(1,3),W1,W1(1,2),W1(1,3),
   2NITER,EPS)
   CALL SECOND(STPTIM)
   NITER1=NITER+1
   IF (EPS(NITER1).LE.EPS1) GO TO 300
   WRITE(LP,295) NITER
   295 FORMAT(20H0--WARNING MORE THAN,17,2X,
   *47HITERATIONS REQUIRED TO OBTAIN DESIRED
   ACCURACY.)
   IFLAG=3
   CALL SECOND(STPTIM)
   NITER1=NITER+1
   IF (EPS(NITER1).LE.EPS1) GO TO 300
   WRITE(LP,295) NITER
   295 FORMAT(20H0--WARNING MORE THAN,17,2X,
   *47HITERATIONS REQUIRED TO OBTAIN DESIRED
   ACCURACY.)
   IFLAG=3
   WRITE(LP,301) IFLAG
   301 FORMAT(20H0--AFTER MA31F IFLAG = ,I2)
   WRITE(LP,305) NITER,EPS(NITER1)
   305 FORMAT(18H0NUM ITERATIONS = ,I3,2X,
   *19HNORM OF RESIDUAL = ,E13.5)
   RTIME=STPTIM-STRTIM
   WRITE(LP,310) RTIME
   310 FORMAT(12HORUN TIME = ,F7.3,4H SEC)
   WRITE(LP,330) NITER1
   330 FORMAT(38H0NORM OF RESIDUAL AFTER EACH
   ITERATION)
110 DO 340 I=1,NITER1
   WRITE(LP,335) (I-1),EPS(I)
   335 FORMAT(1H ,I3,2X,E13.5)
115 500 CONTINUE
RETURN
END
SUBROUTINE MA3IF(N,NZ,A,D,INI,INJ,IAF,DF,INJF,IK,B,R,
 1 E,F,G,KMAX,EPS)

C
C MA3IF IS PART OF THE HARWELL MA31 PACKAGE.
5 C IT HAS BEEN MODIFIED TO:
C 1) HANDLE ADDED PARAMETER TO MA31H CALLING SEQUENCE
C 2) SAVE THE RESULTING RESIDUAL EACH ITERATION
C 3) USE A RANDOM STARTING VECTOR.
C SEE ROUTINE MA31A FOR DETAILS.
10 C

REAL AF(IAF),DF(N),A(NZ),B(N),R(N),E(N),F(N),G(N),L,D(N)
REAL EPS(KMAX)
INTEGER INJF(IAF),INI(NZ),INJ(NZ),IK(N,2)
DATA ZERO/0.0/

C THIS SUBROUTINE PERFORMS THE ITERATIVE PROCEDURE.
C THE PRECONDITIONED CONJUGATE GRADIENT METHOD IS USED.
DO-ZERO
EPSI=EPS(I)**2
20 C
C COMPUTE THE INITIAL SOLUTION.
DO 10 I=1,N
10 E(I)=RANF(I)*2.0
CALL MA31G(N,AF,INJF,IAF,DF,IK,E)

C COMPUTE THE RESIDUALS AND INSERT THE INITIAL SOLUTION IN B.
CALL MA31H(A,D,INI,INJ,NZ,N,E,R)
R1=ZERO
DO 20 I=1,N
20 R(I)=B(I)-E(I)
R1=R1+R(I)**2
G(I)=B(I)
20 B(I)=E(I)
K1R=0
EPSL=SQRT(R1)
IF (R1.LT.EPS1) GO TO 75

C INITIALIZE E AND G.
CALL MA31G(N,AF,INJF,IAF,DF,IK,G)
DO 30 I=1,N
30 E(I)=-G(I)

C START ITERATION LOOP
40 K1R=K1R+1
CALL MA31H(A,D,INI,INJ,NZ,N,E,F)
L=ZERO
DO 40 I=1,N

SUBROUTINE MA31H(A,D,INI,INJ,NZ,N,B,Z)

MA31H IS PART OF THE HARWELL MA31 PACKAGE.

REAL A(NZ),D(N),B(N),Z(N)
INTEGER INI(NZ),INJ(NZ)

THIS SUBROUTINE CALCULATES THE INNER PRODUCT OF A MATRIX A AND A VECTOR B AND THE RESULT IS RETURNED IN VECTOR Z.

THE DIAGONAL ENTRIES OF MATRIX A ARE CONTAINED IN D.

INITIALIZE A.

DO 10 I=1,N
DO 10 I=1,N
10 Z(I)=B(I)*D(I)

LOOP OVER THE NON-ZEROES IN A.

IF (NZ.LE.0) GO TO 100
DO 90 K=1,NZ
   I=INI(K)
   J=INJ(K)
   Z(I)=Z(I)+A(K)*B(J)
90     Z(J)=Z(J)+A(K)*B(I)
100 RETURN
END

PROGRAM PROG2(INPUT,OUTPUT,DATA,TAPE4=INPUT,TAPE6=DATA,
*TAPE5=OUTPUT)

C PROGRAM TO CALCULATE THE EIGENVALUES OF OUR
5 SYMMETRICALLY PRECONDITIONED COEFFICIENT MATRIX
C USING THE HARWELL EA14A LANCZOS ALGORITHM.
C
C SUBROUTINE GENA AND FACTOR
C--------------------------
10 C SEE PROG1 FOR DESCRIPTION
C
C SUBROUTINE GETEIG
C------------------
C SUBROUTINE WHICH CALLS SUBROUTINE EA14A TO CALCULATE
15 C THE DESIRED EIGENVALUES.
C MITE - MAXIMUM NUMBER OF ITERATIONS TO BE ATTEMPTED
C ACC - DESIRED ACCURACY OF RESULTING EIGENVALUES
C EL,ER - SEARCH INTERVAL
C
20 C SEE INDIVIDUAL SUBROUTINES FOR MORE DETAILS.
C
REAL A(650),B(64),W(64,3)
INTEGER INI(200),INJ(650),IK(64,4),IW(64,4),OPTION(6)
C
25 COMMON/MA31I/DD,LP,MP
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31K/NURL,NUCL,NUAL
COMMON/MCOMM3/OPTION
COMMON/MA31N/MITE,ACC,EL,ER
COMMON/MA31L/EPS10L,UL
COMMON/MA31M/NI,NJ,NVERS,NTYPE
EXTERNAL FILL, EUSE

DATA DD, LP, MP/1.0, 6, 5/
DATA EPS25, U/2.0E-6, 1.0E2/
DATA IAIA, IAJ, NN/200, 650, 64/
DATA MITE, ACC, EL, ER/600, 1.0E-4, 1.0, 0.0/
DATA NI, NJ/8, 8/

GET PARAMETERS DETAILING TYPE OF
PRECONDITIONING METHOD TO USE.

READ(4, *) NTYPE, NVERSN
READ(4, *) (OPTION(I), I=1, 6)
READ(4, *) C

CALL GENA(NN, NZ, A, INI, INJ, IAIA, IAJ, W, B, IK, IW)

IF (OPTION(6).EQ.1) GO TO 5

PERFORM THE DESIRED FACTORIZATION

CALL FACTOR(NN, NZ, A, INI, INJ, IAIA, IAJ, IK, IW, W, C, FILL, EUSE)
GO TO 15

NO PRECONDITIONING REQUESTED
GENERATE IDENTITY MATRIX

LROW = 0
DO 10 I = 1, NN
IK(I, 1) = 0
IK(I, 2) = I
W(I, 2) = 1.0
10 CONTINUE

CALCULATE THE EIGENVALUES OF THE PRECONDITIONED MATRIX

CALL GETEIG(NN, NZ, A, INI, INJ, IAIA, IAJ, W, IK, B)

END
SUBROUTINE GETEIG(NN,NZ,A,INI,INJ,IAI,IAJ,W,IK)
C
C SUBROUTINE TO CALCULATE ALL THE EIGENVALUES OF
C OUR SYMMETRICALLY PRECONDITIONED INPUT MATRIX
5 C USING THE HARWELL EAI4A LANCZOS ALGORITHM.
C SEE SUBROUTINE SOLVE FOR DESCRIPTION OF INPUT PARAMETERS.
C
REAL A(IAJ),W(NN,3)
REAL EIG(1024),U(1024),V(1024),T1(1024),T2(1024)
REAL X(3000),DEL(3000),ALFA(5000),BETA(5000)
INTEGER INI(IAI),INJ(IAJ),IK(NN,4)
INTEGER NU(3000)
C
COMMON/EA14BD/PRVT(4),IPRVT(6)
15 COMMON/MA31I/DD,LP,MP
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31N/MITE,ACC,EL,ER
C
DATA LEIG,LX,LALFA/1024, 3000, 5000/
20 C
NZ1=NZ+1
IAJ1=IAJ-NZ
IFLAG=-1
C
A MAXIMUM OF MITE ITERATIONS ARE ATTEMPTED TO
ACQUIRE ALL EIGENVALUES IN THE INTERVAL EL TO ER
TO AN ACCURACY OF ACC.
C
DO 30 ITER=1,MITE
C
CALL EA14AD(NN,EL,ER,ACC,LEIG,LX,LALFA,LP,IFLAG,
*U,V,EIG,NEIG,X,DEL,NU,ALFA,BETA)
C
IF (IFLAG.EQ.0) GO TO 200
35 IF (IFLAG.GT.1) GO TO 100
C
CALCULATES VECTOR U = VECTOR U + MATRIX A' TIMES VECTOR V,
WHERE MATRIX A' IS THE RESULT OF SYMMETRICALLY
PRECONDITIONING MATRIX A BY MATRIX C.
C
40 CALL MA31G2(NN,A(NZ1),INJ(NZ1),IAJ1,W(1,2),IK,V,T1)
CALL MA31H(A,W,INI,INJ,NZ,NN,T1,T2)
CALL MA31G1(NN,A(NZ1),INJ(NZ1),IAJ1,W(1,2),IK,T2)
C
DO 20 I=1,NN
U(I)=U(I) + T2(I)
20 CONTINUE
30 CONTINUE
   GO TO 180
C
C EA14AD IS SIGNALING FAILURE
C
100 WRITE(LP,110) IFLAG
   WRITE(MP,110) IFLAG
110 FORMAT(26H0EA14AD HAS FAILED. IFLAG=,I2)
   GO TO 290
C
C EA14A COULDN'T FINISH IN THE REQUESTED
C NUMBER OF ITERATIONS.
C
180 WRITE(LP,185) MITE
   WRITE(MP,185) MITE
185 FORMAT(39H0--WARNING ALL EIGENVALUES NOT FOUND IN,
   *I3,2X,10HITERATIONS)
   ITER=MITE
C
C OUTPUT DATA ON THE CALCULATED EIGENVALUES
C
70 200 CONTINUE
   WRITE(LP,205) PRVT(I)
205 FORMAT(19H0SPECTRAL RADIUS = ,E14.7)
   WRITE(LP,215)
215 FORMAT(30HODATA ON RESULTING EIGENVALUES)
75 220 FORMAT(8H0ITER = ,I3,2X,6HACC = ,E13.5)
   WRITE(LP,230) NEIG
230 FORMAT(28H0NUM DISTINCT EIGENVALUES = ,I3)
C
80 235 DO 235 I=1,NEIG
235 EIG(I)=EIG(I)-1.0
C
   WRITE(LP,240)
240 FORMAT(25H0STATISTICS ON EIG(I)-1.0)
85 290 CONTINUE
   RETURN
END
SUBROUTINE MA31G1(N,A,INJ,IAJ,D,IK,B)
C
C SUBROUTINE TO SOLVE A SYSTEM OF EQUATIONS
C (L TRANSPOSE) (SQRT D) T = B
C BY BACKWARD SUBSTITUTION. RESULT IS RETURNED
C IN VECTOR B. BASED ON HARWELL ROUTINE MA31G.
C REMINDER, ARRAY A CONTAINS L TRANSPOSE.
C SEE MA31G FOR DESCRIPTION OF VARIABLES.
C
10 INTEGER INJ(IAJ),IK(N,2)
REAL A(IAJ),D(N),B(N)
C
KP=1
C
15 DO 25 II=1,N
IC=IK(II,2)
KL=KP+IK(IC,1)-1
BIC=B(IC)
IF (KP.GT.KL) GO TO 20
20 DO 15 K=KP,KL
IR=INJ(K)
15 B(IR)=B(IR)-A(K)*BIC
20 KP=KP+1
25 CONTINUE
C
DO 30 I=1,N
B(I)=B(I)/SQRT(D(I))
30 CONTINUE
C
30 RETURN
END

SUBROUTINE MA31G2(N,A,INJ,IAJ,D,IK,B,T)
C
C SUBROUTINE TO SOLVE A SYSTEM OF EQUATIONS
C (SQRT D)(L) T = B
C BY FORWARD SUBSTITUTION. BASED ON THE HARWELL
C ROUTINE MA31G. SEE MA31G FOR DESCRIPTION OF
C VARIABLES.
C REMINDER, ARRAY A CONTAINS L TRANSPOSE.
C
10 INTEGER INJ(IAJ),IK(N,2)
REAL A(IAJ),D(N),B(N),T(N)
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
C
KL=LROW

15  C
   DO 10 I=1,N
10  T(I)=B(I)/SQRT(D(I))
   C
   DO 30 IPI=1,N
20  IIP=N+1-IPI
   IR=IK(IIP,2)
   BIR=0.0
   KP=KL-TK(IR,1)+1
   IF (KP,GT,KL) GO TO 25
25  DO 20 K=KP,KL
   IC=INJ(K)
20  BIR=BIR-A(K)*T(IC)
25  T(IR)=T(IR)+BIR
   KL=KP-1
30  30 CONTINUE
   C
   RETURN
   END
PROGRAM PROGIA(OUTPUT,DATA,TAPE5=OUTPUT,TAPE6=DATA)

C DRIVER PROGRAM USED DURING PHASE III.
C DESIGNED TO SOLVE THE TEST PROBLEMS USING:
C A) POINT RED/BLACK GRID POINT ORDERING SCHEME
C (NTYPE=2)
C B) INCOMPLETE CHOLESKY FACTORIZATION W/ 0 DIAGONALS
C ADDED AS PERFORMED BY SUBROUTINE ICCGO.
C THE FOLLOWING PROGRAM CHANGES ARE REQUIRED BY THE
C VARIOUS TEST PROBLEMS:

1) TEST PROBLEM 1 (GENA1)
   NVERSN = 1    NI = 32    NJ = 32
   USE DIMENSIONS
   B(1024), W(1024,3), W1(1024,3), IK(1024,2)
   AND IW(1024)

2) TEST PROBLEM 2 (GENA2)
   NVERSN = 2    NI = 32    NJ = 31
   ND = 992
   USE DIMENSIONS
   B(992), W(992,3), W1(992,3), IK(992,2)
   AND IW(992)

3) TEST PROBLEM 3 (GENA3)
   NVERSN = 3
   EVERYTHING ELSE AS PER PROBLEM 2
   SEE PROGRAMS PROGI, GENA, ICCGO AND SOLVE FOR MORE
   DETAILS.

REAL A(5000),B(1024),W(1024,3),W1(1024,3)
INTEGER INI(2000),INJ(5000),IK(1024,2),IW(1024)
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31N/MITS,EPSI
COMMON/MA31L/EPSTOL,U
COMMON/MA31I/DD,LP,MP
COMMON/MA31M/NI,NJ,NVERSN,NTYPE

DATA U,EPSTOL/1.0E2,2.0E-6/
DATA MITS,EPSI/100,1.0E-6/
DATA IAI,IAJ,ND/2000,5000,1024/
DATA DD,LP,MP/1.0,6,5/
DATA NI,NJ/32,32/
DATA NVERSN,NTYPE/1,2/

CALL GENA(ND,NZ,A,INI,INJ,IAI,IAJ,W,B,IK,IW)
NZ1=NZ+1
CALL ICCGO(ND,NZ,A,INI,INJ,A(NZ1),INJ(NZ1),IK,
*IW,W(1,1),W(1,2))
IAJ2 = NZ + LROW

50 CALL SOLVE(ND, NZ, A, INI, INJ, NZ, IAJ2, W, IK, B, W)

C END

PROGRAM PROG1B(OUTPUT, DATA, TAPE5=OUTPUT, TAPE6=DATA)
C
C DRIVER PROGRAM USED DURING PHASE III.
C DESIGNED TO SOLVE THE TEST PROBLEMS USING
5 C A) LINE RED/BLACK GRID POINT ORDERING SCHEME
C (NTYPE = 1)
C B) CHOLESKY FACTORIZATION OF THE BLOCK TRIDIAGONAL
C PORTION OF MATRIX A, AS PERFORMED BY SUBROUTINE BDIAG.
C SEE PROGIA FOR PROGRAM SET-UP ASSOCIATED WITH EACH OF
10 C THE TEST PROBLEMS.
C SEE PROG1, GENA, BDIAG AND SOLVE FOR MORE DETAILS.
C
REAL A(5000), B(992), W(992, 3), W1(992, 3)
INTEGER INI(2000), INJ(5000), IK(992, 2), IW(992)

15 C COMMON/MA31J/LROW, LCOL, NCP, ND, IPD
COMMON/MA31N/MITS, EPS1
COMMON/MA31L/EPSTOL, U
COMMON/MA31D/DD, LP, MP
20 COMMON/MA3/NII, NJ, NVERSNN, NTYPE

C DATA U, EPSTOL/1.0E2, 2.0E-6/
DATA MITS, EPS1/150, 1.0E-6/
DATA IAI, IAJ, ND/2000, 5000, 992/
25 DATA DD, LP, MP/1.0, 6, 5/
DATA NII, NJ/32, 31/
DATA NVERSNN, NTYPE/2, 1/

C
30 CALL GENA(ND, NZ, A, INI, INJ, IAI, IAJ, W, B, IK, IW)
NZ1 = NZ + 1
CALL BDIAG(ND, NZ, A, INI, INJ, A(NZ1), INJ(NZ1), IK, 
*IW, W(1, 1), W(1, 2))
IAJ2 = NZ + LROW
CALL SOLVE(ND, NZ, A, INI, INJ, NZ, IAJ2, W, IK, B, W)
35 C END
PROGRAM PROGIC (OUTPUT, DATA, TAPE5=OUTPUT, TAPE6=DATA)

C DRIVER PROGRAM USED DURING PHASE III.
C DESIGNED TO SOLVE THE TEST PROBLEMS USING
5 C A) 2 LINE RED/BLACK GRID POINT ORDERING SCHEME
C (NTYPE = 3)
C B) REDUCED BLOCK INCOMPLETE CHOLESKY FACTORIZATION
C WITH 0 DIAGONALS ADDED AS PERFORMED BY RBICO.
C SEE PROG1A FOR PROGRAM SET-UP ASSOCIATED WITH EACH OF
10 C THE TEST PROBLEMS.
C SEE PROG1, GENA, RBICO AND SOLVE FOR MORE DETAILS.
C
REAL A(5000), B(1024), W(1024, 3), W1(1024, 3)
INTEGER INI(2000), INJ(5000), IK(1024, 2), IW(1024)
15 COMMON/MA31/LROW, LCOL, NCP, ND, IPD
COMMON/MA31N/MITS, EPS1
COMMON/MA31L/EPSTOL, U
COMMON/MA311/DD, LP, MP
20 COMMON/MA31M/NI, NJ, NVERSN, NTYPE
C
DATA U, EPS1, EPS1/100, 1.0E-6/
DATA MITS, EPS1/100, 1.0E-6/
DATA IAI, IAJ, ND/2000, 5000, 1024/
25 DATA DD, LP, MP/1.0, 6, 5/
DATA NI, NJ/32, 32/
DATA NVERSN, NTYPE/1, 3/
C
CALL GENA (NDNZ, A, INI, INJ, IAI, IAJ, W, B, IK, IW)
30 NZ = NZ + 1
NCP = NJ
CALL RBICO (ND, NZ, A, INI, INJ, A(NZ1), INJ(NZ1), IK,
* IW, W(1, 1), W(1, 2))
IAJ2 = NZ + LROW
35 CALL SOLVE (ND, NZ, A, INI, INJ, NZ, IAJ2, W, IK, B, WI)
C
END
PROGRAM PROG1D(OUTPUT,DATA,TAPE5=OUTPUT,TAPE6=DATA)

C DRIVER PROGRAM USED DURING PHASE III.
C DESIGNED TO SOLVE THE TEST PROBLEMS USING
5 C A) NATURAL GRID POINT ORDERING SCHEME (NTYPE = 0)
C B) NO PRECONDITIONING
C SEE PROGIA FOR PROGRAM SET-UP ASSOCIATED WITH EACH OF
C THE TEST PROBLEMS.
C SEE PROG1, GENA AND SOLVE FOR MORE DETAILS.

10 C REAL A(5000),B(1024),W(1024,3),W1(1024,3)
INTEGER INI(2000),INJ(5000),IK(1024,2),IW(1024)
C
15 COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31N/MITS,EPSTOL,U
COMMON/MA31I/DD,LP,MP
COMMON/MA31M/NI,NJ,NVERS,N,NTYPE

20 DATA U,EPSTOL/1..0E2,2.0E-6/
DATA MITS,EPSTOL/100,1.OE-6/
DATA IAI,IAJ,ND/2000,5000,1024/
DATA DD,LP,MP/1.0,6,5/
DATA NI,NJ/32,32/
25 DATA NVERS,NTYPE/1,0/

C
C CALL GENA(ND,NZ,A,INI,INJ,IAI,IAJ,W,B,IK,IW)
LROW=0
DO 10 I=1,ND
30 IK(I,1)=0
IK(I,2)=I
W(I,2)=1.0
10 CONTINUE
C
35 WRITE(LP,15)
15 FORMAT(19H NO PRECONDITIONING)
IAJ2=NZ+LROW
CALL SOLVE(ND,NZ,A,INI,INJ,NZ,IAJ2,W,IK,B,WI)
C
40 END
SUBROUTINE GENA(NN,NZ,A,INI,INJ,IAI,IAJ,D,B,IK,IW)

C
C************************************************************************
C
5 C GENA2
C
--
C
C PERFORMS THE DISCRETIZATION OF MODEL PROBLEM 2.
C SEE GENA1 FOR DESCRIPTION OF VARIABLES.

10 C
C************************************************************************
C
REAL A(IAJ),B(NN),D(NN),ATYPE(4)
INTEGER INI (IAI),INJ(IAJ),IK(NN,2),IW(NN)

15 C
COMMON/MA31/LRROW,LCOL,NCP,ND,IPD
COMMON/MA311/DD,LP,MP
COMMON/MA31M/NI,NJ,NVERS,NTYPE

20 DATA ATYPE/7HNATURAL,7HLINE RB,8HPOINT RB,8H2LINE RB/
WRITE(MP,2)
2 FORMAT(11H GENA START)

C
DO 5 I=1,ND
IK(I,1)=0
IK(I,2)=0
IW(I)=0
5 CONTINUE

30 CALL TIME(AT)
CALL DATE(AD)
CALL SECOND(TIM1)

C
NNAT=0
NZ=0

35 C
DO 90 I=1,NI
DO 90 J=1,NJ
NNAT=NNAT+1
N=NORDER(NTYPE,I,J,NNAT)
D(N)=4.0
B(N)=0.0
IF (I.EQ.1) D(N)=D(N)/2.0
IF (J.EQ.NJ) D(N)=D(N)/2.0
IF (I.EQ.NI) D(N)=D(N)/2.0
IF (J.NE.1) GO TO 10
B(N)=1.0
IF ((I.EQ.1).OR.(I.EQ.NI)) B(N)=0.5
10 CONTINUE

50 C CALCULATE INNER DIAGONAL
C
IF (J.EQ.NJ) GO TO 20
NZ=NZ+1
55 A(NZ)=-1.0
IF ((I.EQ.1).OR.(I.EQ.NI)) A(NZ)=-0.5
NT=NORDER(NTYPE,I,J+1,NNAT+1)
CALL ISTORE(N,NT,INI,INJ,IAI,IK,ND,NZ)
20 CONTINUE

60 C CALCULATE OUTER DIAGONAL
C
IF (I.EQ.NI) GO TO 90
NZ=NZ+1
65 A(NZ)=-1.0
IF (J.EQ.NJ) A(NZ)=-0.5
NT=NORDER(NTYPE,I+1,J,NNAT+NJ)
CALL ISTORE(N,NT,INI,INJ,IAI,IK,ND,NZ)
90 CONTINUE

70 100 CONTINUE
C
C INITIALIZE IW(I) TO POINT JUST BEYOND WHERE THE
C LAST COMPONENT OF ROW I WILL BE STORED
C
75 KI=1
DO 200 I=1,ND
KI=KI+IK(I,1)
200 IW(I)=KI
C
80 C REORDER BY ROWS USING IN-PLACE SORT ALGORITHM
C
CALL MA31E(INI,INJ,NZ,IW,ND,A)
C
85 C REINITIALIZE IW(I) TO POINT TO THE BEGINNING OF ROW I
C
85 KI=1
DO 210 IR=1,ND
IW(IR)=KI
210 KK=KK+IK(IR,1)
90 DO 220 I=1,NZ
220 INI(I)=IABS(INI(I))
C
CALL SECOND(TIM2)
TIMD=TIM2-TIM1
95 C WRITE(LP,250) TIMD
SUBROUTINE GENA(NN,NZ,A,INI,INJ,IAI,IAJ,D,B,IK,IW)
C
C*********************************************************************************
C
5 C GENA3
C C ------
C C PERFORMS THE DISCRETIZATION OF MODEL PROBLEM 3.
C C SEE GENA1 FOR DESCRIPTION OF VARIABLES.
10 C*********************************************************************************
C
REAL A(IAJ),B(NN),D(NN),ATYPE(4)
INTEGER INI(IAI),INJ(IAJ),IK(NN,2),IW(NN)
15 C
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/ADATA/NT,NV,AD,AT
COMMON/MA31I/DD,LP,MP
COMMON/MA31M/NI,NJ,NVERS,NTYPE
20 C
DATA ATYPE/7RNATURAL,7HLINE RB,8HPOINT RB,8H2LINE RB/
WRITE(MP,2)
2 FORMAT(11H GENA START)
C
25 DO 5 I=1,ND
IK(I,1)=0
IK(I,2)=0
IW(I)=0
5 CONTINUE
CALL TIME(AT)
CALL DATE(AD)
CALL SECOND(TIM1)

C
35        NNAT=0
NZ=0
H=1.0/31.0
HD2=H/2.0
H2=2.0*H
HSQ=H*H
40        C
X=-H
XP1=HD2
XP1SQ=XP1*XP1

45        C
DO 95 I=1,NI
C
XS1=XP1
XS1SQ=XP1SQ
XP1=XS1+H
XP1SQ=XP1*XP1
X=X+H
XSQ=X*X

50        C
Y=0.0
YP1=HD2
CYP=EXP(X*YP1)

C
DO 95 J=1,NJ
C
Y=Y+H
YS1=YP1
YP1=YS1+H
YSQ=Y*Y

60        C
NNAT=NNAT+1
N=NORDER(NTYPE,I,J,NNAT)
C
AXS=XS1SQ+YSQ+1.0
AXP=XP1SQ+YSQ+1.0
CYS=CYP
CYP=EXP(X*YP1)

C
GXY=4.0*YSQ*(XSQ+YSQ+1.0)+6.0*Y
GXY=XSQ*GXY+2.0*Y*(YSQ+1.0)
GXY=1.0-GXY-XSQ*YSQ*(XSQ+X)*EXP(X*Y)
GXY=EXP(XSQ*Y)*GXY
D(N) = AXS + AXP + CYS + CYP + HSQ
B(N) = HSQ * GXY

C
IF (I.EQ.1) GO TO 25
IF (I.EQ.NI) GO TO 50
C
IF (J.EQ.1) GO TO 10
IF (J.NE.NJ) GO TO 15
C
D(N) = (D(N) + H2 * CYP) / 2.0
B(N) = (B(N) + H2 * CYP * EXP(XSQ) * (1.0 + XSQ)) / 2.0
NZ = NZ + 1
A(NZ) = -AXP / 2.0
GO TO 20
C
NZ = NZ + 1
B(N) = B(N) + CYS
C
NZ = NZ + 1
A(NZ) = -CYP
NT = NORDER(NTYPE, I, J+1, NNAT+1)
CALL ISTORE(N, NT, INI, INJ, IAI, IK, ND, NZ)
NZ = NZ + 1
A(NZ) = -AXP
NT = NORDER(NTYPE, I+1, J, NNAT+NJ)
CALL ISTORE(N, NT, INI, INJ, IAI, IK, ND, NZ)
GO TO 95
C
D(N) = D(N) / 2.0
B(N) = B(N) / 2.0
IF (J.NE.1) GO TO 30
B(N) = B(N) + CYS / 2.0
GO TO 35
C
IF (J.NE.NJ) GO TO 35
T1 = H2 * CYP / 4.0
D(N) = D(N) / 2.0 + T1
B(N) = B(N) / 2.0 + T1
NZ = NZ + 1
A(NZ) = -AXP / 4.0
GO TO 40
C
NZ = NZ + 1
A(NZ) = -CYP / 2.0
NT = NORDER(NTYPE, I, J+1, NNAT+1)
CALL ISTORE(N, NT, INI, INJ, IAI, IK, ND, NZ)
NZ = NZ + 1
A(NZ) = -AXP / 2.0
NT = NORDER(NTYPE, I+1, J, NNAT+NJ)
CALL ISTORE(N, NT, INI, INJ, IAI, IK, ND, NZ)
GO TO 95

50 IF (J.EQ.NJ) GO TO 55
D(N)=D(N)/2.0
B(N)=B(N)+4.0*H*Y*EXP(Y)*AXP
IF (J.EQ.1) B(N)=B(N)+CYS
B(N)=B(N)/2.0
NZ=NZ+1
A(NZ)=-CYP/2.0
NT=NORDER(NTYPE,I,J+1,NNAT+1)
CALL ISTORE(N,NT,INI,INJ,IAI,IK,ND,NZ)
GO TO 95

140 55 D(N)=(D(N)+H2*CYP)/4.0
B(N)=B(N)/4.0+H*EXP(1.0)*(AXP+CYP)
C

95 CONTINUE
C

145 C INITIALIZE IW(I) TO POINT JUST BEYOND WHERE THE
C LAST COMPONENT OF ROW I WILL BE STORED
C
KI=1
DO 200 I=1,ND
KI=KI+IK(I,1)
200 IW(I)=KI
C
C REORDER BY ROWS USING IN-PLACE SORT ALGORITHM
C
CALL MA31E(INI,INJ,NZ,IW,ND,A)
C
C REINITIALIZE IW(I) TO POINT TO THE BEGINNING OF ROW I
C
KK=1
DO 210 IR=1,ND
IW(IR)=KK
KK=KK+IK(IR,1)
DO 220 I=1,NZ
220 INI(I)=IABS(INI(I))
210
C
CALL SECOND(TIM2)
TIMD=TIM2-TIMI
C
WRITE(LP,250) TIMD
250 FORMAT(13H GENA TIME = ,F6.3,4H SEC)
WRITE(LP,260) NVERSION
260 FORMAT(1H VERSION = ,I2)
WRITE(LP,265) ATYPE(NTYPE+1)
265 FORMAT(14H MATRIX A HAS ,A10,9H ORDERING)
SUBROUTINE ICCGO(NN,NZA,A,INI,INJ,C,INJC,IK,IW,DA,DC)
C
C SUBROUTINE TO CALCULATE THE INCOMPLETE CHOLESY
C FACTORIZATION WITH ZERO FILL-IN OF THE INPUT
C
INPUT PARAMETERS
---------------
NN - ORDER OF MATRIX A
NZA - NUMBER OF NON-ZERO ELEMENTS IN THE UPPER
TRIANGULAR PORTION OF MATRIX A
A - ARRAY CONTAINING THE NON-ZERO ELEMENTS IN THE
UPPER TRIANGULAR PORTION OF MATRIX A IN ROW ORDER
INI/INJ - ARRAYS CONTAINING THE ROW/COLUMN INDICES
OF THE CORRESPONDING ENTRY IN ARRAY A.
(IE. INI(I) AND INJ(I) CONTAIN THE ROW AND COLUMN INDICE FOR ENTRY A(I))
IK(I,1) - CONTAINS THE NUMBER OF ELEMENTS IN
ARRAY A BELONGING TO ROW I
IW(I) - POINTS TO THE START OF ROW I IN ARRAY A
DA - ARRAY CONTAINING THE DIAGONAL ELEMENTS OF
MATRIX A

OUTPUT PARAMETERS
---------------
C - ARRAY CONTAINING THE NON-ZERO ELEMENTS IN THE
UPPER TRIANGULAR PORTION OF THE INCOMPLETE
CHOLESKY FACTORIZATION
INJC - ARRAY CONTAINING THE COLUMN INDEX OF THE
CORRESPONDING ENTRY IN ARRAY C
DC - ARRAY CONTAINING THE DIAGONAL ELEMENTS IN
THE INCOMPLETE CHOLESKY FACTORIZATION
C IK(I,1) - NUMBER OF NON-ZERO ELEMENTS IN ROW I
C OF THE INCOMPLETE FACTORIZATION
C IK(I,2) - USED BY OTHER HARWELL ROUTINES TO
C IDENTIFY THE ORDER IN WHICH THE ROWS
C WERE PROCESSED. IN THIS CASE, ROWS
C PROCESSED IN NATURAL ORDER AND
C IK(I,2) = I
C
COMMON BLOCK PARAMETERS

C DD,LCOL,NCP,IPD - NOT USED
C LP - OUTPUT FILE UNIT NUMBER
C MP - MESSAGE FILE UNIT NUMBER
C LROW - NUMBER OF NON-ZERO ELEMENTS IN THE UPPER
C TRIANGULAR PORTION OF THE INCOMPLETE
C FACTORIZATION
C ND - ORDER OF MATRIX A
C EPSTOL - MINIMUM SIZE FOR DIAGONAL ELEMENT
C U - PARAMETER USED TO DETERMINE WHEN A DIAGONAL
C ELEMENT MUST BE MODIFIED TO INSURE POSITIVE
C DEFINITENESS
C
INTEGER IK(NN,2),IW(NN),INI(NZA),INJ(NZA),INJC(NZA)
REAL A(NZA),DA(NN),DC(NN),C(NZA)
COMMON/MA311/DDLP,MP
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31L/EPSTOL,U

CALL SECOND(TIM1)
C
WRITE(MP,2)
2 FORMAT(12H START ICCGO)
WRITE(LP,3)
3 FORMAT(26H PRECONDITIONING = ICCG(0))
C
IDC=0
CT=EPSTOL*U
IRC=0
C
DO 5 K=1,ND
5 DC(K)=DA(K)
C
DO 100 IROW=1,ND
IRS=IW(IROW)
IRE=IRS+IK(IROW,1)-1
IK(IROW,1)=0
IK(IROW,2) = IROW

C DETERMINE IF DIAGONAL ELEMENT MUST BE MODIFIED
C TO PRESERVE POSITIVE DEFINITENESS
C
CO = CT
90 IF (IRS.GT.IRE) GO TO 20
   DO 10 K = IRS, IRE
   10 CO = AMAX1(CO, ABS(A(K)))
20 IF (DC(IROW).GT.(CO/U)) GO TO 30
   IDC = IDC + 1
   DC(IROW) = CO
   IF (CO.EQ.CT) DC(IROW) = 1.0
30 CONTINUE
C
C PROCESS ELEMENTS IN CURRENT ROW
100 IF (IRS.GT.IRE) GO TO 100
   DO 90 IR = IRS, IRE
      I = INI(IR)
      J = INJ(IR)
      105 IRC = IRC + 1
      T = A(IR)
      C(IRC) = T/DC(IROW)
      INJC(IRC) = J
      DC(J) = DC(J) - T*C(IRC)
      110 IK(IROW,1) = IK(IROW,1) + 1
90 CONTINUE
C
100 CONTINUE
C
115 LROW = IRC
   CALL SECOND(TIM2)
   TMD = TIM2 - TIM1
C
C OUTPUT STATISTICS
120 WRITE(LP, 110) TMD
110 FORMAT(14H ICCGO TIME = ,F6.3, 5H SECS)
   WRITE(LP, 120) LROW
120 FORMAT(8H LROW = ,I4)
130 IF (IDC.NE.0) WRITE(LP, 130) IDC
130 FORMAT(4H ** I4, 19H DIAGONALS MODIFIED)
   WRITE(MP, 140)
140 FORMAT(10H ICCGO END)
C
RETURN
END
SUBROUTINE BDIAG(NN, NZA, A, INI, INJ, C, INJC, IK, IW, DA, DC)

C SUBROUTINE TO CALCULATE THE CHOLESKY FACTORIZATION
C OF THE TRI-DIAGONAL PORTION OF THE INPUT MATRIX A.

C SEE SUBROUTINE ICCGO FOR DESCRIPTION OF PARAMETERS.

REAL A(NZA), DA(NN), DC(NN), C(NZA)
INTEGER IK(NN, 2), IW(NN), INI(NZA), INJ(NZA), INJC(NZA)

COMMON/MA31/DD, LP, MP
COMMON/MA31J/LROW, LCOL, NCP, ND, IPD
COMMON/MA31L/EPSTOL, U

CALL SECOND(TIML)
WRITE(MP, 2)
WRITE(LP, 3)

DO 5 K=1, ND
  DC(K)=DA(K)

DO 100 IROW=1, ND
  IRS=IW(IROW)
  IRE=IRS+IK(IROW,1)-1
  IK(IROW,1)=0
  IK(IROW,2)=IROW

C DETERMINE IF DIAGONAL ELEMENT MUST BE MODIFIED
C TO PRESERVE POSITIVE DEFINITENESS

CO=CT
  IF (IRS.GT.IRE) GO TO 20
  DO 10 K=IRS, IRE
    CO=AMAX1(CO, ABS(A(K)))
  20 IF (DC(IROW).GT.(CO/U)) GO TO 30
    IDC=IDC+1
    DC(IROW)=CO
  30 IF (CO.EQ.CT) DC(IROW)=1.0

C PROCESS ELEMENTS IN THE CURRENT ROW
50 IF (IRS.GT.IRE) GO TO 100
DO 90 IR=IRS,IRE
I=INJ(IR)
J=INJ(IR)
IF ((J-I).GT.1) GO TO 90
55 IRC=IRC+1
T=A(IR)
C(IRC)=T/DC(IRW)
INJC(IRC)=J
DC(J)=DC(J)-T*C(IRC)
60 IK(IRW,1)=IK(IRW,1)+1
90 CONTINUE
100 CONTINUE
C
LROW=IRC
65 CALL SECOND(TIM2)
TIMD=TIM2-TIM1
C
C OUTPUT STATISTICS
C
70 WRITE(LP,110) TIMD
110 FORMAT(14H BDIAG TIME = ,F6.3,5H SECS)
WRITE(LP,120) LROW
120 FORMAT(8H LROW = ,I4)
IF (IDC.NE.0) WRITE(LP,130) IDC
75 130 FORMAT(4H ** ,I4,19H DIAGONALS MODIFIED)
WRITE(MP, 140)
140 FORMAT(10H BDIAG END)
C
RETURN
80 END

SUBROUTINE RBICO(NN,NZA,A,INI,INJ,C,INJC,IK,DA,DC)
C
C SUBROUTINE TO CALCULATE THE INCOMPLETE CHOLESKY
C FACTORIZATION OF THE QUINT-DIAGONAL PORTION OF
C THE INPUT MATRIX A. IT IS ASSUMED THAT THE
C 2 LINE RED/BLACK ORDERING OF GRID POINTS WAS
C USED IN GENERATING MATRIX A.
C
C SEE SUBROUTINE ICCGO FOR DESCRIPTION OF PARAMETERS
C
10 C
C NCP - DISTANCE FROM MAIN DIAGONAL TO OUTER MOST
C DIAGONAL TO BE INCLUDED IN THE INCOMPLETE
FACTORIZATION.

REAL A(NZA),DA(NN),DC(NN),C(NZA)
INTEGER IK(NN,2),IWN(NN),INI(NZA),INJ(NZA),INJC(NZA)

COMMON/MA31I/DD,LP,MP
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31L/EPSTOL,U

CALL SECOND(TIM1)
WRITE(MP,2)
WRITE(LP,3)

IDC=0
CT=EPSTOL*U
IRC=0

DO 5 K=1,ND
DC(K)=DA(K)

DO 100 IROW=1,ND
IRS=IW(IROW)
IRE=IRS+IK(IROW,1)-1
IK(IROW,1)=0
IK(IROW,2)=IROW

DETERMINE IF DIAGONAL ELEMENT MUST BE MODIFIED
TO PRESERVE POSITIVE DEFINITENESS

CO=CT
IF (IRS.GT.IRE) GO TO 20
DO 10 K=IRS,IRE
10 CO=AMAX1(CO,ABS(A(K)))
IF (DC(IROW).GT.(CO/U)) GO TO 30

IDC=IDC+1
DC(IROW)=CO
IF (CO.EQ.CT) DC(IROW)=1.0
CONTINUE

PROCESS ELEMENTS IN CURRENT ROW

IF (IRS.GT.IRE) GO TO 100
DO 90 IR=IRS,IRE
I=INI(IR)
90 J=INJ(IR)
IF ((J-I) .GT. NCP) GO TO 90
IRC = IRC+1
T = A(IR)
C(IRC) = T/DC(IR)
INJC(IRC) = J
DC(J) = DC(J) - T*C(IRC)
IK(IRON, 1) = IK(IRON, 1) + 1
90 CONTINUE
100 CONTINUE
70 C
LROW = IRC
CALL SECOND(TIM2)
TIMD = TIM2 - TIM1
C
75 C OUTPUT STATISTICS
C
WRITE(LP, 110) TIMD
110 FORMAT(14H RBICO TIME = , F6.3, 5H SECS)
WRITE(LP, 120) LROW
120 FORMAT(8H LROW = , I4)
IF (IDC .NE. 0) WRITE(LP, 130) IDC
130 FORMAT(4H **, I4, 19H DIAGONALS MODIFIED)
WRITE(NP, 140)
140 FORMAT(10H RBICO END)
85 C
RETURN
END
PROGRAM PROG2A(OUTPUT, DATA, TAPE5=OUTPUT, TAPE6=DATA)
C
DRIVER PROGRAM USED DURING PHASE III.
C DESIGNED TO FIND THE EXTREME EIGENVALUES OF THE
5 SYMMETRICALLY PRECONDITIONED TEST MATRICES.
C IT USES:
C A) POINT RED/BLACK GRID POINT ORDERING SCHEME (NTYPE = 2)
C B) INCOMPLETE CHOLESKY FACTORIZATION WITH 0 DIAGONALS
C ADDED, AS PERFORMED BY SUBROUTINE ICCGO.
10 SEE PROG1A FOR PROGRAM SET-UP ASSOCIATED WITH EACH OF
C THE TEST PROBLEMS.
C SEE PROG2, GENA, ICCGO AND GETEG2 FOR MORE DETAILS.
C
REAL A(5000), B(1024), W(1024, 3), W1(1024, 3)
15 INTEGER N1(2000), INJ(5000), IK(1024, 2), IW(1024)
C
COMMON/MA3I/LROW, LCOL, NCP, ND, IPD
COMMON/MA3I/MITE, ACC, EL, ER
COMMON/MA3I/EPSTOL, U
20 COMMON/MA3II/DD, LP, MP
COMMON/MA3II/NI, NJ, NVERS, NTYPE
C
DATA U, EPSTOL/1.0E2, 2.0E-6/
DATA MITE, ACC, EL, ER/750, 1.0E-2, 0.0, 1.2/
25 DATA IAI, IAJ, ND/2000, 5000, 1024/
DATA DD, LP, MP/1.0, 6, 5/
DATA NI, NJ/32, 32/
DATA NVERS, NTYPE/1, 2/
C
30 CALL GENA(ND, NZ, A, INI, INJ, IAI, IAJ, W, B, IK, IW)
NZ1=NZ+1
CALL ICCGO(ND, NZ, A, INI, INJ, A(NZ1), INJ(NZ1), IK, IW,
*W(1, 1), W(1, 2))
IAJ2=NZ+LROW
35 CALL GETEG2(ND, NZ, A, INI, INJ, NZ, IAJ2, W, IK)
C
END
PROGRAM PROG2B(OUTPUT, DATA, TAPE5=OUTPUT, TAPE6=DATA)
C
C DRIVER PROGRAM USED DURING PHASE III.
C DESIGNED TO FIND THE EXTREME EIGENVALUES OF THE
5 C SYMMETRICALLY PRECONDITIONED TEST MATRICES.
C IT USES:
C A) LINE RED/BLACK GRID POINT ORDERING SCHEME (NTYPE = 1)
C B) CHOLESKY FACTORIZATION OF THE BLOCK TRI-DIAGONAL PORTION
C OF MATRIX A.
10 C SEE PROG1A FOR PROGRAM SET-UP ASSOCIATED WITH EACH OF
C THE TEST PROBLEMS.
C SEE PROG2, GENA, BDIAG AND GETEG2 FOR MORE DETAILS.
C
REAL A(5000), B(1024), W(1024, 3), W1(1024, 3)
INTEGER INI(2000), INJ(5000), IK(1024, 2), IW(1024)
C
COMMON/MA3 J/LROW, LCOL, NCP, ND, IPD
COMMON/MA3 IN/MITE, ACC, EL, ER
COMMON/MA3 LP/EPSTOL, U
20 COMMON/MA3 L/ACC, EL, ER/750, 1.0E-2, 0.0, 1.2/
COMMON/MA3 M/ND, NJ, NVERSN, NTYPE
COMMON/MA3 N/IAI, IAJ, ND/2000, 5000, 1024/
DATA U, EPSTOL/1.0E2, 2.0E-6/
DATA MITE, ACC, EL, ER/750, 1.0E-2, 0.0, 1.2/
DATA DD, LP, MP/1.0, 6, 5/
DATA NI, NJ/32, 32/
DATA NVERSN, NTYPE/1, 1/
C
30 CALL GENA(ND, NZ, A, INI, INJ, IAI, IAJ, W, B, IK, IW)
NZI=NZ+1
CALL BDIAG(ND, NZ, A, INI, INJ, A(NZ1), INJ(NZ1), IK, IW,
*W(1, 1), W(1, 2))
IAJ2=NZ+LROW
35 CALL GETEG2(ND, NZ, A, INI, INJ, NZ, IAJ2, W, IK)
C
END
PROGRAM PROG2C(OUTPUT,DATA,TAPE5=OUTPUT,TAPE6=DATA)

C DRIVER PROGRAM USED DURING PHASE III.
C DESIGNED TO FIND THE EXTREME EIGENVALUES OF THE
C SYMMETRICALLY PRECONDITIONED TEST MATRICES.
C IT USES:
C A) 2 LINE RED/BLACK GRID POINT ORDERING SCHEME (NTYPE = 3)
C B) REDUCED BLOCK INCOMPLETE CHOLESKY FACTORIZATION WITH
C 0 DIAGONALS ADDED AS PERFORMED BY SUBROUTINE RBICO.
10 C SEE PROGIA FOR PROGRAM SET-UP ASSOCIATED WITH EACH OF THE
C TEST PROBLEMS.
C SEE PROG2, GENA, RBICO AND GETEG2 FOR MORE DETAILS.
C
REAL A(5000),B(1024),W(1024,3),W1(1024,3)
INTEGER INI(2000),INJ(5000),IK(1024,2),IW(1024)
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31N/MITE,ACC,EL,ER
COMMON/MA31L/EPSTOL,U
20 COMMON/MA31I/DD,LP,MP
COMMON/MA31M/NJ,NJ,NVRSN,NTYPE
C
DATA U,EPSTOL/1.0E2,2.0E-6/
DATA MITE,ACC,EL,ER/1500,1.0E-2,1.0,0.0/
25 DATA IAI,IAJ,ND/2000,5000,1024/
DATA DD,LP,MP/1.0,6,5/
DATA NJ,NJ/32,32/
DATA NVRSN,NTYPE/1,3/
C
30 CALL GENA(ND,NZ,A,INI,INJ,IAI,IAJ,W,B,IK,IW)
NZ1=NZ+1
NCP=NJ
CALL RBICO(ND,NZ,A,INI,INJ,A(NZ1),INJ(NZ1),IK,IW,
35 *W(1,1),W(1,2))
IAJ2=NZ+LROW
CALL GETEG2(ND,NZ,A,INI,INJ,NZ,IAJ2,W,IK)
C
END
PROGRAM PROG2D(OUTPUT,DATA,TAPE5=OUTPUT,TAPE6=DATA)

C
C DRIVER PROGRAM USED DURING PHASE III.
C DESIGNED TO FIND THE EXTREME EIGENVALUES OF THE
5 C SYMMETRICALLY PRECONDITIONED TEST MATRICES.
C IT USES:
C A) NATURAL GRID POINT ORDERING SCHEME (NTYPE = 0)
C B) NO PRECONDITIONING
C SEE PROG1A FOR PROGRAM SET-UP ASSOCIATED WITH EACH OF
10 C THE TEST PROBLEMS.
C SEE PROG2, GENA AND GETEG2 FOR MORE DETAILS.
C
REAL A(5000),B(1024),W(1024,3),WI(1024,3)
INTEGER INI(2000),INJ(5000),I(104,2),IW(1024)
15 C COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31N/MITE,ACC,EL,ER
COMMON/MA31L/EPSTOL,U
COMMON/MA31I/DD,LP,MP
20 COMMON/MA31M/NI,NJ,NVERSN,NTYPE

C DATA U,EPSTOL/1.0E2,2.0E-6/
DATA MITE,ACC,EL,ER/750,1.0E-2,0.0,1.2/
DATA IAI,IAJ,ND/2000,5000,1024/
25 DATA DD,LP,MP/1.0,6,5/
DATA NI,NJ/32,32/
DATA NVERSN,NTYPE/1,0/

C CALL GENA(ND,NZ,A,INI,INJ,IAI,IAJ,W,B,IK,IW)
30 LROW=0
DO 10 I=1,ND
IK(I,1)=0
IK(I,2)=I
W(I,2)=1.0
35 10 CONTINUE
C
WRITE(LP,15)
15 FORMAT(19H NO PRECONDITIONING)
IAJ2=NZ+LROW
40 CALL GETEG2(ND,NZ,A,INI,INJ,NZ,IAJ2,W,IK)
C
END
SUBROUTINE GETEG2(NN,NZ,A,INI,INJ,IAI,IAJ,W,IK)
C
C SUBROUTINE TO CALCULATE THE HIGH AND LOW ORDER
C EIGENVALUES OF OUR SYMMETRICALLY PRECONDITIONED
C INPUT MATRIX USING THE HARWELL EAI4A LANCZOS
C ALGORITHM. SEE SOLVE FOR DESCRIPTION OF INPUT
C PARAMETERS.
C
REAL A(IAJ),W(NN,3)
REAL EIG(1024),U(1024),V(1024),T1(1024),T2(1024)
REAL X(3000),DEL(3000),ALFA(5000),BETA(5000)
INTEGER INI(IAI),INJ(IAJ),IK(NN,4)
INTEGER NU(3000)

COMMON/EA14BD/PRVT(4),IPRVT(6)
COMMON/MA31I/DD,LP,MP
COMMON/MA31J/LROW,LCOL,NCP,ND,IPD
COMMON/MA31N/MITE,ACC,EL,ER

DATA LEIG,LX,LALFA/1024,3000,5000/

NZ1=NZ+1
IAJ1=IAJ-NZ
IFLAG=-1

CALL SECOND(TIM1)
C PASS 1 CALCULATES THE EIGENVALUES IN THE INTERVAL
C EL TO ER AS SPECIFIED BY THE CALLING ROUTINE.
C PASS 2 CALCULATES THE HIGH ORDER EIGENVALUES USING
C ESTIMATED NORM OF THE MATRIX PRODUCED BY ROUTINE
C EAI4A TO DEFINE THE INTERVAL TO BE EXAMINED.
C
DO 290 IPASS=1,2
C
WRITE(MP,10) IPASS
10 FORMAT(6H PASS ,I1,6H START)
C
CALL SECOND(TIM2)
C
A MAXIMUM OF MITE ITERATIONS ARE ATTEMPTED TO
C ACQUIRE ALL EIGENVALUES IN THE INTERVAL EL TO ER
C TO AN ACCURACY OF ACC.
C
DO 30 ITER=1,MITE
C
CALL EAI4AD(NN,EL,ER,ACC,LEIG,LX,LALFA,LP,IFLAG,
*U,V,EIG,NEIG,X,DEL,NU,ALFA,BETA)
154  

IF (IFLAG.EQ.0) GO TO 200  
IF (IFLAG.GT.1) GO TO 100  

C  
C CALCULATES VECTOR U = VECTOR U + MATRIX A' TIMES VECTOR V,  
C WHERE MATRIX A' IS THE RESULT OF SYMMETRICALLY  
C PRECONDITIONING MATRIX A BY MATRIX C.  

CALL MA31G2(NN,A(NZ1),INJ(NZ1),IAJ1,W(1,2),IK,V,T1)  
CALL MA31H(A,W,INI,INJ,NZ,NN,T1,T2)  
CALL MA31G1(NN,A(NZ1),INJ(NZ1),IAJ1,W(1,2),IK,T2)  

DO 20 I=1,NN  
U(I)=U(I) + T2(I)  
20 CONTINUE  

GO TO 180  

C  
C EA14AD IS SIGNALING FAILURE  
C  
100 WRITE(LP,110) IFLAG  
WRITE(MP,110) IFLAG  
110 FORMAT(26HOEA14AD HAS FAILED. IFLAG-,I2)  
GO TO 290  

C  
C EA14AD COULDN'T FINISH IN THE REQUESTED  
C NUMBER OF ITERATIONS  

180 WRITE(LP,185) MITE  
WRITE(MP,185) MITE  
185 FORMAT(39HO--WARNING ALL EIGENVALUES NOT FOUND IN,  
*I4,2X,10HITERATIONS)  
ITER=MITE  

C  
C OUTPUT DATA ON THE CALCULATED EIGENVALUES  
C  
85 200 CONTINUE  
CALL SECOND(TIM3)  
TRUN=TIM3-TIM2  
WRITE(LP,202) IPASS,TRUN  
202 FORMAT(6H PASS ,I1,12H RUN TIME = ,F10.3,5H SECS)  
WRITE(LP,205) PRVT(1)  
205 FORMAT(19HOSPECTRAL RADIUS = ,E14.7)  
WRITE(LP,210) EL,ER  
210 FORMAT(19H INTERVAL EXAMINED ,E13.5,3H - ,E13.5)  
WRITE(LP,215)  
215 FORMAT(39HODATA ON RESULTING EIGENVALUES)  
WRITE(LP,220) ITER,ACC  
95
220 FORMAT(8H ITER = ,I3,2X,6H ACC = ,E13.5)
WRITE(LP,230) NEIG
230 FORMAT(28H NUM DISTINCT EIGENVALUES = ,I3)

100 C
WRITE(LP,235)
235 FORMAT(28H ORDERED LIST OF EIGENVALUES)
WRITE(LP,240) (EIG(I),I=1,NEIG)
240 FORMAT(1X,10E13.5)

105 WRITE(MP,245) IPASS
245 FORMAT(6H PASS ,I1,5H DONE)

C PERPARE TO EXAMINE EIGENVALUES AT THE END OF THE SPECTRUM.

110 C
EL=PRVT(1)*0.8
ER=PRVT(1)*1.1

C
290 CONTINUE

115 C
CALL SECOND(TIM4)
TT=TIM4-TIM1
WRITE(LP,295) TT
295 FORMAT(18H TOTAL RUN TIME = ,F10.3,5H SECS)

120 C
RETURN
END
References


[Vors81] van der Vorst, H., private communication.
