NASA Contractor Report 178117

NASA-CR-178117
19860019188

MULTI-COLOR INCOMPLETE CHOLESKY CONJUGATE GRADIENT METHODS FOR VECTOR COMPUTERS

Eugene L. Poole

UNIVERSITY OF VIRGINIA
Charlottesville, Virginia

Grant NAG1-242
May 1986

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

Table of Contents

List of Figures

List of Symbols

Acknowledgements

Many individuals contributed in various ways to this research. I wish to thank Dr. James Ortega for his long hours of invaluable assistance which greatly aided in both my research and in the writing of this dissertation. I would also like to thank the rest of the faculty in the Engineering school at the University of Virginia for their contributions to my academic endeavors.

I want to express my love to my wife, Annette, for her many unselfish sacrifices during the past several years.

I want to thank the personnel at the NASA Langley Research Center who were so helpful during my times at Langley, particularly Jay Lambiotte, Geoff Tennille, and Lona Howser. Finally, I wish to thank NASA for the financial assistance they provided under grant NAG-1-242 and also Control Data Corporation for financial assistance under the Pacer fellowship program.

Abstract

In this research we are concerned with the solution on vector computers of linear systems of equations, $A\mathbf{x} = \mathbf{b}$, where A is a large, sparse symmetric positive definite matrix with non-zero elements lying only along a few diagonals of the matrix. We solve the system using the incomplete Cholesky conjugate gradient method (ICCG), an iterative method which has proven effective for a wide class of problems on scalar computers. Following the suggestion of Schrieber and Tang [1982], we apply the multi-color strategy used by Adams and Ortega [1982] to obtain p-color matrices for which an ICCG method is implemented on the Cyber 205 with $O(N/p)$ length vector operations in both the decomposition of A and, more importantly, in the forward and back solves necessary at each iteration of the method. (N is the number of unknowns and p is a small constant) A p-colored matrix is a matrix which can be partitioned into a $p \times p$ block matrix where the diagonal blocks are diagonal matrices.

The ICCG method we use is the no-fill strategy of Meijerink and van der Vorst [1977] applied to the p-colored matrices. Because of the block structure of p-color matrices we implement the ICCG(0) method in a block fashion where if the matrix is stored by diagonals the decomposition is carried out by multiplying N/p dimension blocks together using the matrix multiplication by diagonals of Madsen, et al. [1975]. Likewise the forward and back solves at each iteration are accomplished by matrix-vector multiplication by diagonals of these N/p dimension blocks. For a given problem it is necessary to find multi-color orderings which achieve the block structure of p-color matrices but we also desire long vectors within the blocks. Additionally, if the vectors across adjacent blocks line up, then some of the overhead associated with vector startups can be eliminated in the matrix vector multiplication necessary at each conjugate gradient iteration.

We discuss the natural ordering of the unknowns as an ordering that minimizes the number of diagonals in the matrix and define multi-color orderings in terms of disjoint sets of the unknowns. We give necessary and sufficient conditions to determine which multi-color orderings of the unknowns correspond to p-color matrices. We also indicate a process for choosing multi-color orderings, called the *continuous coloring rule* which is easy to apply to a wide class of problems including more difficult 3 dimensional problems and which results in p-color matrices with the desired long vector lengths within the blocks of the matrix.

A performance model is given which is used both to predict execution time for the ICCG methods and also to compare an ICCG method to conjugate gradient without preconditioning or another ICCG method. Results are given from runs on the CYBER 205 at NASA's Langley Research Center for four model problems including a three dimensional space truss, developed in conjunction with NASA engineers as a simplified model of an orbiting space platform. For all the model problems the multi-color ICCG methods we implemented ran at near the maximum possible rate on the CYBER 205. Our results showed that these methods are competitive with other vectorized ICCG methods in terms of overall speedup of execution compared to conjugate gradient.

CHAPTER 1

Introduction

We consider in this thesis the solution on vector computers of linear systems of equations, $A\mathbf{x} = \mathbf{b}$, where A is a large, sparse symmetric positive definite matrix with non zero elements lying only along a few diagonals of the matrix; such matrices arise in the solution of elliptic partial differential equations by finite difference or finite element discretizations. We solve the system using incomplete Cholesky conjugate gradient (ICCG), an iterative method that has proven effective for a wide class of problems on scalar computers. The multi-coloring strategy described by Adams and Ortega [1982] for the SOR iterative method is used to obtain a matrix structure which yields long vectors. We address the question of choosing the best coloring strategy for a given problem and present results for ICCG applied to Laplace's equation, a more general elliptic partial differential equation, and two finite element applications: plane stress in two dimensions and a three dimensional space platform model.

Our primary interest is in memory to memory vector computers for which efficient usage requires algorithms that consist mainly of operations on long vectors. The CDC CYBER 205 is a vector computer of this type and its successor, the ETA-10, will be to a somewhat lesser extent. The CYBER 205 has a clock cycle time of 20 nanoseconds (ns) and using r vector pipeline units, results of a vector operation are available every $\frac{20}{r}$ ns. Thus, a CYBER 205 with 4 pipeline units can execute floating point operations on contiguously stored operands at a maximum rate of 200 million operations per second (Mflops). For the 'linked triad' operation - vector plus

scalar times vector - this maximum rate is doubled. Half precision (32 bit) arithmetic also doubles the maximum rate. However, associated with each vector instruction is a fixed overhead cost, the *startup* cost, which adds significantly to the time required for vector operations on short vectors. For a vector add, for example, the time to add two vectors of length N on a 2 pipe CYBER 205 can be expressed as $T \approx (1000 + 10N)$ns. If N $\approx$ 100 only 50 percent of the maximum rate is achievable while vector lengths of 10,000 will result in 99 percent of the maximum rate.

The conjugate gradient method first was developed by Hestenes and Stiefel [1952] and continues to be of great interest. The computational steps at each iteration of the algorithm consist of a matrix-vector multiply, two dot products, and three linked triads and the CYBER 205 can potentially be used efficiently. For the problems we are interested in, the number of unknowns, N, is very large and while A is very sparse, the nonzero structure is such that diagonal storage of the matrix and matrix multiplication by diagonals (Madsen, et al. [1976]) can be used to achieve very high computation rates for matrix-vector multiplication. However, for most problems of interest in scientific and engineering applications, the conjugate gradient algorithm converges too slowly and so various 'preconditioning' strategies are employed to reduce the number of iterations. Preconditioned conjugate gradient methods are discussed in a number of places (see, e.g., Evans [1983]). A standard implementation of the preconditioned conjugate gradient algorithm (PCG) is shown in Figure 1.1 , where (,) denotes the usual inner product. The preconditioning is carried out at each iteration by the solution of the system $M\hat{\mathbf{r}}^{k+1} = \mathbf{r}^{k+1}$, where M is a symmetric, positive definite matrix that approximates A in some sense. If M is the identity, the algorithm reduces to the original conjugate gradient method. How well a particular choice for M performs depends on the trade off between the extra time it takes for the additional step in the algorithm and the number of conjugate

(1) Choose $\mathbf{x}^0$

(2) Set $\mathbf{r}^0 = \mathbf{b} - A\ \mathbf{x}^0$

(3) Solve $M\ \hat{\mathbf{r}}^0 = \mathbf{r}^0$

(4) Set $\mathbf{p}^0 = \mathbf{r}^0$

(5) Loop k = 0,1,.....,kmax

a) $\alpha_k = \dfrac{(\hat{\mathbf{r}}^k\ ,\mathbf{r}^k)}{(\mathbf{p}^k\ ,A\ \mathbf{p}^k)}$

b) $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k$

c) $\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha_k A\ \mathbf{p}^k$

d) if $\dfrac{||\ \mathbf{r}^{k+1}\ ||_2}{||\ \mathbf{r}^0\ ||_2} \leqslant \epsilon$ then stop

e) Solve: $M\ \hat{\mathbf{r}}^{k+1} = \mathbf{r}^{k+1}$

f) $\beta_k = \dfrac{(\hat{\mathbf{r}}^{k+1}\ ,\mathbf{r}^{k+1})}{(\hat{\mathbf{r}}^k\ ,\mathbf{r}^k)}$

g) $\mathbf{p}^{k+1} = \hat{\mathbf{r}}^{k+1} + \beta_k \mathbf{p}^k$

Preconditioned Conjugate Gradient Algorithm

Figure 1.1

gradient iterations saved. For the CYBER 205, we desire that the formation of M, if necessary, and particularly the solution of $M\hat{\mathbf{r}}^{k+1} = \mathbf{r}^{k+1}$ be accomplished with suitably long vectors.

Dubois, et al. [1979], Johnson, et al. [1983], and Adams [1983b] have considered preconditioned conjugate gradient methods on the CYBER 205 but not the ICCG method. For the ICCG method, introduced by Meijerink and van der Vorst [1977], M is chosen to be an incomplete Cholesky factorization of A, and can be represented in the so-called root-free Cholesky form, which avoids costly square root calculations, as $M = LDL^T$, where L is unit lower triangular, D is diagonal, and $R = M - A \neq 0$. If R=0, then M is just the complete Cholesky factorization of A.

Complete Cholesky factorization of the large sparse matrices we are considering is not practical, however, due to the large storage requirement from the fill that occurs and the amount of time required to do the decomposition. In addition, the long vector lengths required for efficient use of the CYBER 205 cannot be achieved.

On serial computers, ICCG has been shown to be an effective preconditioned conjugate gradient algorithm for a wide class of problems. Kershaw [1978] has shown that incomplete Cholesky is advantageous over other preconditioners for conjugate gradient for some very ill - conditioned problems. Manteuffel [1980] considered incomplete factorizations for arbitrary symmetric positive definite systems, extending the previous results of Meijerink and Van der Vorst [1977] on symmetric M-matrices. Meijerink and Van der Vorst [1981] also discuss various incomplete Cholesky conjugate gradient methods and the effectiveness of each on several model problems. Block ICCG methods have been considered by Axelsson [1984] and Concus, et al. [1985] and in some cases were more effective than point incomplete factorizations. More details on various versions of ICCG are presented in the next chapter.

There have also been several studies on the implementation of ICCG on vector and parallel computers. Kershaw [1982] gives an implementation of ICCG on the CRAY-1 vector computer by using a cyclic reduction technique applied to block tridiagonal matrices. The vector lengths used in his algorithm are too short, however, for the CYBER 205. Lichnewsky [1983] discusses parallel and vector implementations for ICCG but also mainly gives algorithms with vector lengths better suited for the CRAY computers. Meurant [1984] discusses vectorized block preconditioned conjugate gradient methods applied to block tridiagonal systems where the diagonal blocks are tridiagonal and of dimension $O(\sqrt{N})$ and gives results for both the CRAY-1 and CYBER 205. Van der Vorst [1985] also gives results for the CRAY-1 and CYBER 205 for both a non-vectorized ICCG algorithm and a vectorized

version. Schrieber and Tang [1982] suggest the multicoloring approach we have used but little is given as to how to multi-color the unknowns to achieve the best vectorization.

The multi-coloring technique was also used by Adams and Ortega [1982] to implement SOR for vector and parallel architectures. The main idea in multi-color orderings is to reorder the unknowns so that the resulting matrix is partitioned as in (1.1) into blocks where the diagonal blocks are themselves diagonal matrices. Such a matrix will be called a p-colored matrix. Here, p is the number of colors used to achieve the ordering.

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ A_{12}^T & A_{22} & & \\ \vdots & & \ddots & \vdots \\ A_{1p}^T & \cdots & & A_{pp} \end{bmatrix} \tag{1.1}$$

If $p = 2$, this is the classical red/black ordering (Young [1971]). In general the number of colors, p, will be small and the vector lengths within the A_{ij} blocks will be $O(N/p)$. By using diagonal storage of A we will see that the preconditioning step can be accomplished using vectors of length $O(N/p)$. We discuss criteria and algorithms for choosing multi-color orderings in chapter 3. We also present a procedure to assemble finite element matrices row by row using diagonal storage.

The multi-coloring techniques we have developed are applied to four model problems. The first problem is Laplace's equation

$$U_{xx} + U_{yy} = 0 \tag{1.2}$$

on the unit square with Dirichelet boundary conditions on all four sides. To obtain a more general differential equation we add a mixed derivative term and solve

$$U_{xx} + \frac{1}{2}U_{xy} + U_{yy} = 4 \tag{1.3}$$

on the same region. The usual second order finite difference discretizations are applied to both (1.2) and (1.3). The third problem is a two dimensional plane stress problem for a plate fastened to a rigid body on one side and loaded on the other side. Here, linear basis functions are used in a finite element discretization. Some preliminary results for this problem are given in Poole and Ortega [1984]. For the fourth problem a three dimensional space truss, developed in conjunction with NASA engineers as a simplified model of an orbiting space platform, is considered. Further details on these model problems will be given in subsequent chapters.

In chapter 2 we discuss in more detail ICCG methods which have been used on scalar and vector computers and then describe how multi-coloring is used to vectorize ICCG with long vectors. In Chapter 3 we treat the coloring problem in greater depth. We give necessary and sufficient conditions to obtain a p-color matrix with a multi-color ordering. We also give a strategy which is easy to use on a wide class of problems, including three dimensional problems with possibly more than one unknown per grid point, to obtain p-color matrices containing long vectors. In chapter 4 we present numerical results as well as a computational model and compare our results with those reported in the literature for similar problems. Finally, in chapter 5 we summarize the results and discuss future directions for related research.

CHAPTER 2

ICCG Methods

In this chapter we discuss several versions of incomplete Cholesky factorization which have been used as preconditioners for the conjugate gradient method. We then describe multi-color block incomplete Cholesky and give two different implementations which use long vector operations suitable for the CYBER 205.

2.1. Point ICCG Methods

An incomplete, root-free factorization of A is of the form $M = LDL^T$ where L is unit lower triangular, D is diagonal, and $R = M - A \neq 0$. For a complete Cholesky factorization, the elements of L and D must satisfy

$$a_{ij} = \sum_{k=1}^{j} l_{ik} d_k l_{jk} \quad i,j = 1,...n \tag{2.1}$$

which leads to the expressions

$$l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_k l_{jk}) d_j^{-1} \tag{2.2a}$$

$$d_i = l_{ii} d_i = a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 d_k \tag{2.2b}$$

Modifications of the equations (2.2) that limit the fill occurring in L define different versions of incomplete Cholesky factorization. One modification, given by Munksgaard [1980], is to ignore elements of L if they are numerically small compared to the diagonal elements of their respective row and column. This strategy is not well suited for our purposes since it is not in general possible to determine ahead of time which elements of L will be dropped and so the structure of L cannot be determined in advance.

The usual modification used for incomplete Cholesky factorization is to allow fill to occur during the formation of L only in pre-specified locations. The non-zero structure of L is often described using a set $J = \{ (i,j) \}$ of integer pairs where $l_{ij} = 0$ if $(i,j) \notin J$. A simple choice for J is $J = \{(i,j) \mid a_{ij} \neq 0\}$ and the ICCG(0) method of Meijerink and van der Vorst [1977] calculates l_{ij} for $(i,j) \in J$ from equation (2.2a) ; otherwise, $l_{ij} = 0$. ICCG(0) is called the 'no-fill' strategy since L will have the same non-zero structure as the lower triangular part of A. Partial fill strategies can also be described by using a suitable set J and in Meijerink and van der Vorst [1981] a class of incomplete decompositions denoted ICCG(h,k) is given where the indices h,k indicate that fill is allowed in L along h diagonals adjacent to the main diagonal and k diagonals inside the lower band of A. The problem considered in their paper arises in the solution of elliptic partial differential equations in two dimensions using the five-point discretization.

Another modification to the basic point ICCG method, called MICCG, was introduced by Gustafsson [1978] and involves using fill elements created in the factorization process to modify the diagonal matrix, D, to improve convergence rates for certain types of problems. However, Kightley and Jones [1985] note that the unmodified ICCG method has a faster convergence rate on certain problems. For further discussion of the MICCG method and the closely related 'column sum' constraint used in modifying D, see Jackson and Robinson [1981].

A major concern in incomplete decompositions is instability. As opposed to complete Cholesky decomposition of a symmetric positive definite matrix, incomplete decompositions may not always be carried out. In (2.2b) if some d_i is zero then (2.2a) cannot be computed. Likewise if any elements of D become negative, then M will not be positive definite and cannot be used as a preconditioner for conjugate gradient. Meijerink and van der Vorst [1977] showed that for symmetric M-matrices incomplete factorizations exist for any choice of J. Manteuffel [1980]

extended these results to H-matrices (A is an H-matrix if the matrix B with $b_{ii} = |a_{ii}|$ and $b_{ij} = -|a_{ij}|$ is an M-matrix.) and goes on to describe a 'shifting algorithm' for general positive definite systems whereby the diagonal elements of A are increased before factorization to ensure that the decomposition can be completed. He notes that for any A the new matrix obtained by increasing the main diagonal elements can always be made diagonally dominant, and thus an H-matrix, so that for any J an incomplete decomposition exists. It is not in general necessary to make the shifted matrix diagonally dominant and a much smaller increase of the diagonal elements may suffice.

Kershaw [1978] presents a different method to ensure that incomplete factorizations exist for symmetric positive definite systems. He notes that for incomplete factorizations defined by (2.2) and a set J as described above, the non-zero entries of the error matrix, R, lie only in those places corresponding to zero entries in A. If a non-positive element is computed for D he proposes setting that element to some positive value and continuing with the decomposition. This causes a non-zero entry in the corresponding diagonal element of R but if few of these corrections are made it is hoped that the incomplete factors will still give a good approximation of A. This method has the advantage of not changing the entire diagonal matrix D, as Mantueffel's shifting method does, but has the disadvantage of requiring a test for each element of D during the factorization. Robert [1982] defines *regular incomplete factorizations* for positive definite matrices with respect to a set J as described above, to have the property that $r_{ij} = 0$ if $(i,j) \in J, i \neq j$.

2.2. Block ICCG methods

We next consider block incomplete factorizations. Here the matrix A is partitioned as in equation (1.1), (with the diagonal blocks A_{ii} not necessarily diagonal). The block Cholesky form of (2.2) is

$$L_{ij} = [A_{ij} - \sum_{k=1}^{j-1} L_{ik} D_k L_{jk}^T] D_j^{-1} \tag{2.3a}$$

$$D_i = A_{ii} - \sum_{k=1}^{i-1} L_{ik} D_k L_{ik}^T . \tag{2.3b}$$

where now the L_{ij} and the D_i are matrices. In this form the L_{ii} are identity matrices but the D_i may be full. Hence, calculation of the L_{ij} involves solving systems with the D_j as coefficient matrices. More importantly, the forward and back solves at each iteration of ICCG become very costly on both scalar and vector computers if the D_j are dense. Incomplete block factorizations are used to deal with this problem.

For incomplete block Cholesky methods the set J can describe the non-zero structure of L in terms of its blocks, L_{ij}, or the nonzero structure of L in terms of the individual elements, l_{ij}, of L. It is clear that in the former case the L_{ij} do not in general have the same non-zero structure as the corresponding blocks of A. The incomplete block Cholesky factorization we will describe in the next section uses J for the individual elements.

In Concus, et al. [1985] and Axelsson [1984] incomplete block factorizations are discussed for the case where A is a block tridiagonal matrix. The focus of the paper by Concus, et al. is on ways to approximate the inverses required in (2.3). However, for the CYBER 205, block tridiagonal systems are not desirable since the dimensions of the blocks are usually only $O(\sqrt{N})$. Kershaw [1982] gives an algorithm for a block ICCG method for the CRAY-1 which uses cyclic reduction and vectorizes well with the shorter vector lengths but as noted above such a strategy on the CYBER 205 does not seem promising.

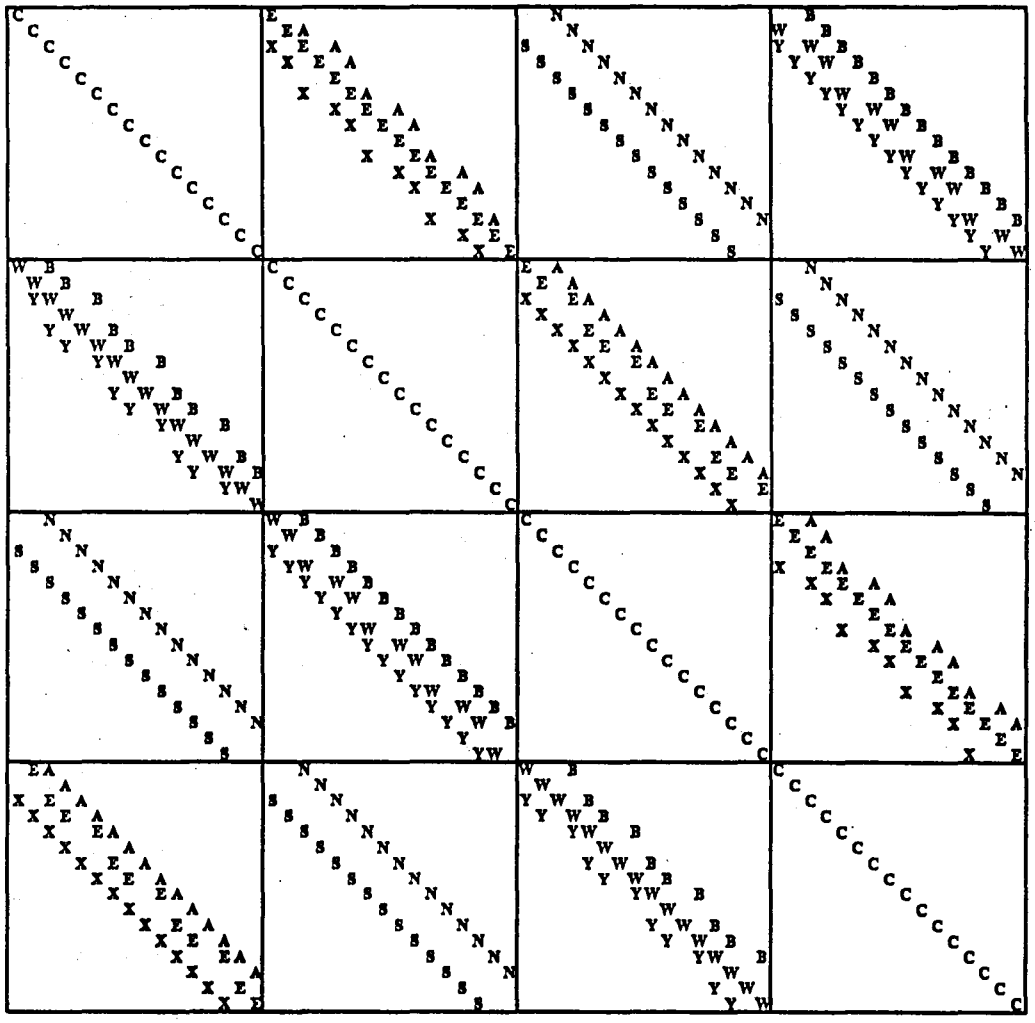
Van der Vorst [1982] discusses some block ICCG methods of a different kind for block tridiagonal matrices. One method uses a truncated Neumann series to approximate the inverses of bidiagonal matrices as part of the forward and back solves for the Laplace problem. He also discusses [1983] a diagonal ordering of the

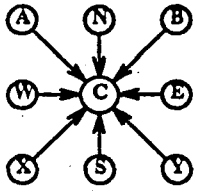
unknowns to vectorize the forward and back solves at each iteration of ICCG. Both methods use vector lengths that are $O(\sqrt{N})$ in the preconditioning step. We discuss the diagonal ordering further in chapter 3 and give results for the Laplace problem from a predicted performance model in chapter 4.

Lichnewsky [1983] also discusses block methods for vector and parallel computers using a 'subdomain approach' to reorder the unknowns to increase parallelism. For this method, the unknowns are decoupled into subdomains using orderings which contain subsets of the unknowns called *separators* that are used to break the problem into several smaller problems. These smaller problems can then be solved independently and joined together to form the final solution to the original problem. This approach may be better suited for a parallel architecture but Lichnewsky also suggests applying the multi-color orderings we describe next within the subdomains to vectorize the solution of the decoupled systems.

2.3. Multi-color Block Incomplete Cholesky

To derive a more appropriate ICCG method for the CYBER 205 we follow a suggestion of Schrieber and Tang [1982] and use multi-color orderings. If A has a block structure as in (1.1) and the A_{ii} are diagonal, we say that A is p-colored (see Adams and Ortega [1982]). To give an example of how such matrices arise, we consider the mixed derivative problem, equation (1.3). If we discretize (1.3) by the usual second-order finite difference approximations, we can order the unknowns so as to obtain the 4-colored matrix shown in Figure 2.1. In the grid below the matrix in Figure 2.1 the numbers represent four colors and the matrix is assembled by ordering the equations of color 1 left to right, bottom to top followed in like manner by colors 2,3 and 4. The grid stencil below the matrix in Figure 2.1 indicates the coupling of unknowns by the finite difference discretization. For each row in the matrix each non-zero coefficient corresponds to the unknown indicated by





Grid Stencil

1	2	3	4	1	2	3	4
.	.	.	.	.	.	.	.
1	2	3	4	1	2	3	4
3	4	1	2	3	4	1	2
1	2	3	4	1	2	3	4

Grid Multi-Color Ordering

Multicolor Ordering for Mixed Derivative Model Problem

Figure 2.1

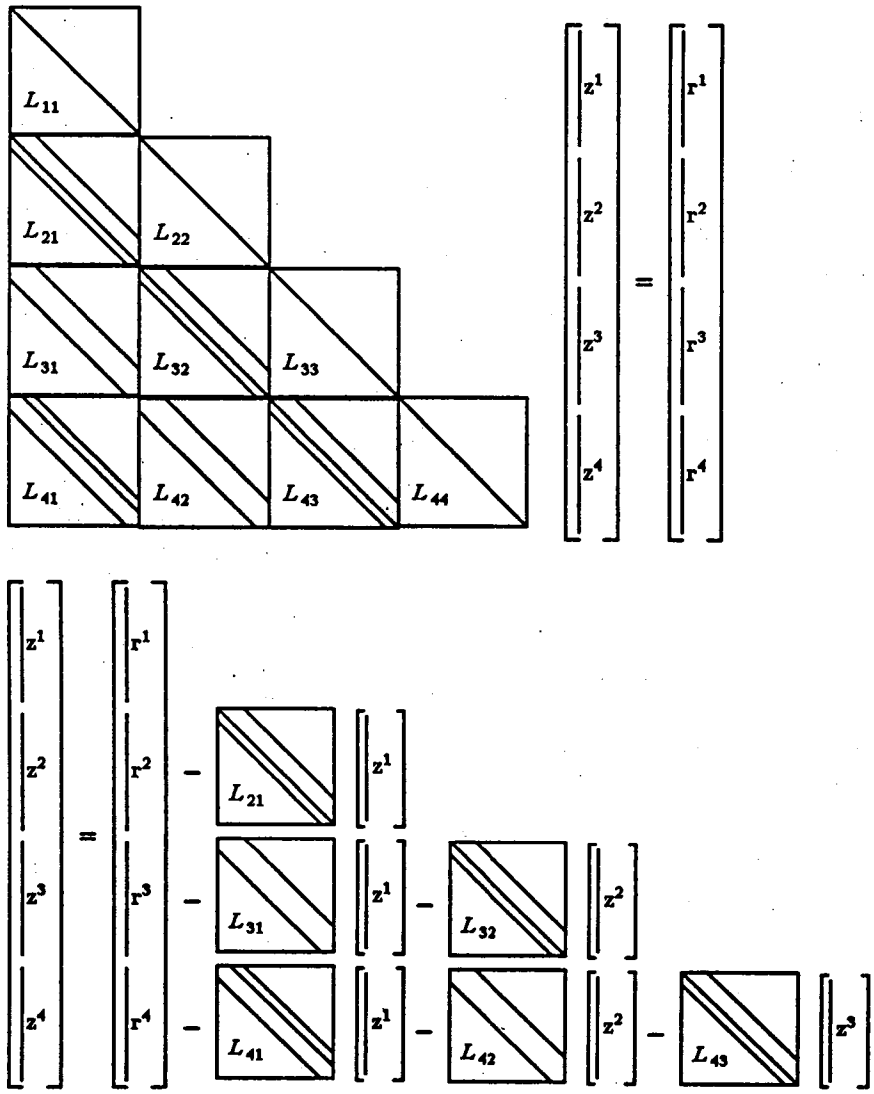
the stencil element with the same letter. For example the unknown at the first grid point corresponds to the first row in the matrix in Figure 2.1 and has non-zero

coefficients corresponding to the unknowns indicated by the C, N, E, and B points in the grid stencil so that row one has the letters C, N, E, and B in the appropriate places. In chapter three we discuss multi-coloring in more detail and we will show that the particular coloring shown in Figure 2.1 is not the best choice for this problem.

Incomplete decomposition of a p-colored matrix can be carried out using partial, or incomplete, multiplication of the blocks of A and L in the block Cholesky equations (2.3). Under a no-fill strategy, the blocks of the incomplete Cholesky factors L and D will have the same non-zero structure as the corresponding blocks of A. Partial fill strategies that allow fill along specified diagonals in the off diagonal blocks can also be implemented; in either case no fill is allowed in the D_j so that the D_j^{-1} operation in (2.3a) is just a vector divide. A is stored by diagonals, and we use matrix multiplication by diagonals as in Madsen, et al. [1976] for the matrix-matrix multiplications required in the decomposition. Details of this procedure are given in appendix A. Thus, the matrix-matrix multiplication consists of vector multiplies and adds, and we do only the vector operations which contribute to diagonals in the allowed non-zero structure. The same storage for A allows the matrix-vector multiplication at each conjugate gradient iteration to be carried out using long vectors. It is clear from (2.3) that the first block column of L equals the first block column of A. Storage requirements for L are thereby lessened and time is saved in the decomposition.

2.4. Implementation of multi-color ICCG

The first implementation we consider, ICCGC, is based on the usual PCG algorithm (see Figure 1.1). Here the preconditioning step consists of solving $M\hat{\mathbf{r}} = \mathbf{r}$ where the incomplete Cholesky factorization $M = LDL^T$ is obtained as discussed in the previous section. The solution of $LDL^T\hat{\mathbf{r}} = \mathbf{r}$ at each iteration of the ICCGC



Solution of $Lz = r$ for 4×4 Block Matrix

Figure 2.2

method is carried out in the usual 3-step process:

$$L\,\mathbf{z} = \mathbf{r} \tag{2.4a}$$

$$\hat{\mathbf{z}} = D^{-1}\mathbf{z} \tag{2.4b}$$

$$L^T \hat{\mathbf{r}} = \hat{\mathbf{z}} \tag{2.4c}$$

Because of the structure of L and D described above, and with L stored by diagonals, the entire process can be carried out with $O(N/p)$ length vectors in (2.4a) and (2.4c) and one vector divide of length N in (2.4b). Hence, the cost is essentially the same as the multiplication by A in the conjugate gradient iteration. The forward solve (2.4a) for a 4-color matrix stored by diagonals is illustrated in Figure 2.2. Details of this process will also be given in appendix A. The effectiveness of this multi-coloring approach in achieving an ICCG method which vectorizes well on the CYBER 205 is dependent upon finding orderings which achieve p-colored matrices and which yield a structure that minimizes the number of diagonals within each block of L. We discuss this question more fully in chapter three.

2.5. ICCGE Implementation

A more efficient implementation for block ICCG can be derived following Eisenstat [1981]. The algorithm, which we call ICCGE, is shown in Figure 2.3.

The idea behind this implementation is the following. As before, $M = LDL^T$ is the incomplete Cholesky factorization. The ICCGE algorithm shown in Figure 2.3 is equivalent to a preconditioned conjugate gradient method with a preconditioning matrix $\hat{M} = D$ applied to the system

$$\hat{A}\hat{\mathbf{x}} = \hat{\mathbf{b}} \;\; ; \;\; \hat{A} = L^{-1}AL^{-T} \;\; , \;\; \hat{\mathbf{x}} = L^T\mathbf{x} \;\; , \;\; \hat{\mathbf{b}} = L^{-1}\mathbf{b} \tag{2.5}$$

A key consideration now is the efficient evaluation of $\hat{A}\hat{\mathbf{p}}$ in terms of the original matrix A. If we set $K = L + L^T - A$, then

(1) Choose $\mathbf{x}^0$

(2) Set $\hat{\mathbf{r}}^0 = L^{-1}(\mathbf{b} - A\mathbf{x}^0)$

(3) Set $\hat{\mathbf{p}}^0 = \mathbf{q}^0 = D^{-1}\hat{\mathbf{r}}^0$

(4) Loop k = 0,1,.....,kmax

a) $\hat{\alpha}_k = \dfrac{(\hat{\mathbf{r}}^k, \mathbf{q}^k)}{(\hat{\mathbf{p}}^k, \hat{A}\,\hat{\mathbf{p}}^k)}$

b) $\mathbf{x}^{k+1} = \mathbf{x}^k + \hat{\alpha}_k L^{-T}\hat{\mathbf{p}}^k$

c) $\hat{\mathbf{r}}^{k+1} = \hat{\mathbf{r}}^k - \hat{\alpha}^k \hat{A}\,\hat{\mathbf{p}}^k$

d) if $\dfrac{||\hat{\mathbf{r}}^{k+1}||}{||\hat{\mathbf{r}}^0||} \leqslant \epsilon$
then stop

e) Solve: $D\,\mathbf{q}^{k+1} = \hat{\mathbf{r}}^{k+1}$

f) $\hat{\beta}_k = \dfrac{(\hat{\mathbf{r}}^{k+1}, \mathbf{q}^{k+1})}{(\hat{\mathbf{r}}^k, \mathbf{q}^k)}$

g) $\hat{\mathbf{p}}^{k+1} = \mathbf{q}^{k+1} + \hat{\beta}_k\hat{\mathbf{p}}^k$

ICCGE algorithm

Figure 2.3

$$\hat{A}\,\hat{\mathbf{p}} = L^{-1}(L + L^T - K)L^{-T}\hat{\mathbf{p}} = L^{-T}\hat{\mathbf{p}} + L^{-1}(\hat{\mathbf{p}} - KL^{-T}\hat{\mathbf{p}}) = \mathbf{t} + L^{-1}(\hat{\mathbf{p}} - K\mathbf{t}) \quad (2.6)$$

where $\mathbf{t} = L^{-T}\hat{\mathbf{p}}$. Thus, the evaluation of $\hat{A}\hat{\mathbf{p}}$ requires a forward and back solve plus the multiplication $K\mathbf{t}$. The forward and back solves are carried out as in Figure 2.2. Note that since $L^{-T}\mathbf{p}$ is required in (2.6), it can be used to update $\mathbf{x}^k$, as shown in Figure 2.3, so that we maintain the original $\mathbf{x}$ variables.

We now show how to make the matrix vector multiplication, $K\mathbf{t}$, less costly than the corresponding $A\mathbf{p}^k$ in the first implementation. If we prescale A so that its main diagonal is the identity, then the first Cholesky block is $D_1 = I$ and the first block column of L is $L_{i1} = A_{i1}$, i=1,....,p, so that $K_{i1} = 0$, $i = 1, \cdots, p$. For a p-colored matrix A, the evaluation of $A\mathbf{p}$ requires $p^2 - p$ block matrix-vector

multiplications while for Kt, p^2-3p+2 block matrix-vector multiplications are required. Thus $2p-2$ block matrix-vector multiplications are saved, although the actual savings will depend on the number of non-zero elements in each of the A_{i1} blocks. We also note that this implementation of PCG for vector computers is not attractive for any of the natural order based factorizations since one trades the matrix multiplication by A during each iteration for the forward and back solves in (2.6). It is the vectorization of the forward and back solves by multi-color ordering that makes this implementation more efficient.

For matrices that can be 2-colored the ICCGE implementation is particularly well suited as we now show. We first write A as

$$A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} \begin{bmatrix} L_{11} & L_{21}^T \\ & L_{22} \end{bmatrix} \tag{2.7}$$

where $L_{11} = A_{11}$, $L_{21} = A_{21}$, $L_{22} = A_{22} - A_{21}A_{11}^{-1}A_{21}^T$, $D_1 = L_{11}^{-1}$, and $D_2 = L_{22}^{-1}$. Note that L_{11} and L_{22} are symmetric. Note also that for this form of block Cholesky the L_{ij} are not multiplied by D_j^{-1} as in (2.3). Now assume that we prescale A so that its main diagonal is the identity. Since A is 2-colored, the matrices A_{11} and A_{22} of (2.7) are diagonal and thus the scaled matrix has the form

$$\hat{A} = \begin{bmatrix} I & \hat{A}_{21}^T \\ \hat{A}_{21} & I \end{bmatrix} \tag{2.8}$$

The decomposition (2.7) applied to $\hat{A}$ gives $\hat{L}_{11} = I$, $\hat{L}_{21} = \hat{A}_{21}$, $\hat{L}_{21} = \hat{A}_{21}$, $\hat{L}_{22} = I - \hat{A}_{21}A_{21}^T$, and $D = I$. Now both K in (2.6) and D in step d) of Figure 2.3 are identity matrices so that the ICCGE algorithm for 2-colored matrices costs essentially the same as standard conjugate gradient.

Both block ICCG implementations described above for p-colored matrices are in fact equivalent to the point ICCG(0) method of Meijerink and van der Vorst [1977] applied to p-color matrices. In chapter 4 we examine the effect of multi-color

orderings on both the convergence rate of the ICCG method and the total execution time.

2.6. m-step ICCG

Incomplete factorization may be viewed as a splitting, $A = LDL^T - R$, and we are led to consider the iterative method defined by this splitting. Following Adams, [1983b] we take m steps of the iterative method

$$LDL^T \hat{r}_i^k = R\,\hat{r}_{i-1}^k + r^k \quad , \quad \hat{r}_0^k = 0 \ , \ i=1, \cdots, m \tag{2.9}$$

as the preconditioning step at the kth conjugate gradient iteration and call this an m-step ICCG method. With m=1, (2.9) reduces to ICCG as previously described. An improvement of the m-step ICCG preconditioning can be achieved using polynomial preconditioning as suggested by Johnson, Micchelli, and Paul [1983] and used by Adams [1983b].

Adams [1985] discusses m-step preconditioned conjugate gradient methods and gives necessary and sufficient conditions for the preconditioning matrix M to be symmetric positive definite when M is given in terms of an m-step linear stationary method defined by any splitting, $A = P - Q$, where P is a symmetric nonsingular matrix. If $K = P^{-1}Q$ is the iteration matrix, M^{-1} is given explicitly by

$$M^{-1} = (I + K + \cdots + K^{m-1})P^{-1} \tag{2.10}$$

For incomplete factorization, $P = LDL^T$ and Adam's result can be stated as: For odd m, M is positive definite if and only if LDL^T is positive definite while for even m, M is positive definite if and only if $LDL^T + R$ is positive definite. Assuming that the incomplete factorization of A ensures that D is positive definite, then for odd m, M will be positive definite. Since R is not explicitly known, the case of m even is not so easily ascertained. One way to meet the conditions for M to be positive definite for even m is to ensure that R is positive semi-definite, and Robert [1982] gives an incomplete factorization with this property. The main drawback of

Robert's factorization method for the CYBER 205 is that the decomposition is recursively defined. It also requires the formation of R which increases the storage requirements significantly. Robert's results show that the shifting method of Manteuffel is an alternate way of ensuring that R is positive semidefinite.

The major drawback in using (2.9) is that R is not explicitly formed during the decomposition and if it is formed, it may take more storage than L. We can compute the vector $R\,\hat{r}_{i-1}^{k}$ by using $R = LDL^T - A$ but this requires a costly matrix multiply, $A\,\hat{r}_{i-1}^{k}$, for $m \geqslant 2$. Since the cost of solving LDL^T is nearly the same as a matrix-vector multiply, the cost of m-step incomplete Cholesky preconditioning will be approximately $2m-1$ times the cost of a matrix-vector multiply so that even though the number of iterations may decrease for m-step ICCGC, the execution time may not. It remains an open question as to how to carry out an m-step ICCG method in an efficient way.

Summary

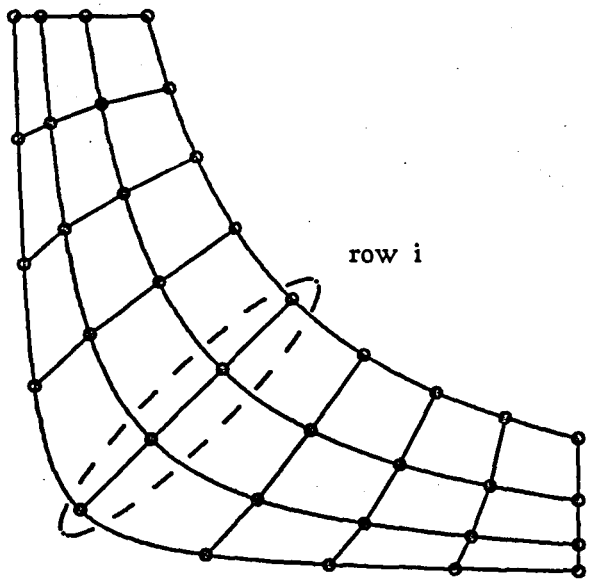
In this chapter we have discussed point and block incomplete Cholesky factorizations. We have shown how multi-color orderings are used to derive a block incomplete Cholesky method suitable for the CYBER 205. An implementation was given, based on a generalization of Eisenstat [1981], which is superior to the usual PCG form given in chapter 1. We also discussed m-step methods but noted that a major drawback of these methods is the necessity of additional matrix-vector multiplies, which make the process too costly.

CHAPTER 3

Multi-Coloring to Vectorize ICCG

In this chapter we discuss the multi-coloring technique used to vectorize the ICCG method. The multi-color orderings we describe can be applied to a wide class of problems but for simplicity and clarity we shall restrict ourselves to a class of problems we call class $\mathcal{R}$ which arise from the solution of partial differential equations using finite element or finite difference methods. We assume the domain of a class $\mathcal{R}$ problem to be rectangular in 2 or 3 dimensions with Dirichlet boundary conditions imposed along each side of the domain. To obtain a numerical approximation to the exact solution of the partial differential equation, we discretize the domain so that there are r points per row in l planes, each containing c rows, leading to the solution of a linear system of equations, $A\mathbf{x} = \mathbf{b}$. The vector $\mathbf{x}$ contains the unknowns which are the approximate solutions to the partial differential equation at the grid points. At each grid point there will be $k > 1$ unknowns if we are solving a system of partial differential equations. The size of the linear system for a class $\mathcal{R}$ problem is given by $N = r \times c \times l \times k$.

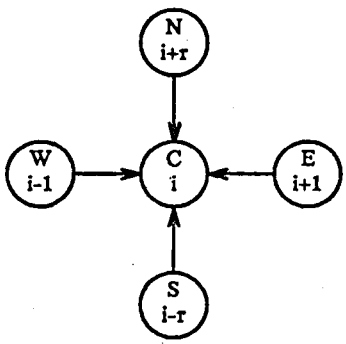
Although class $\mathcal{R}$ is limited to rectangular and rectangular parallelepiped domains and Dirichlet boundary conditions, the results given in this chapter extend to more general 2 and 3 dimensional problems. Figure 3.1 is an example of a non-rectangular region which has been discretized so that there are "rows" of equal numbers of grid points which can be ordered by the multi-color orderings we describe in this chapter. Moreover, it is possible to handle Neumann or other boundary conditions, at least in certain cases.



'Rectangular - like' Discretization

Figure 3.1

Each row, i, in the matrix A contains the non-zero coefficients for the equation for the unknown x_i. For class $\mathcal{R}$ problems we will assume that the number and location of non-zero coefficients is described by a uniform grid stencil; that is, the same grid stencil is applied at each grid point. Note that at grid points near the boundary of the domain, any elements of the stencil associated with known boundary conditions do not give rise to non-zero elements in the matrix A. For example, the Laplace problem has a 5 point grid stencil shown in Figure 3.2, indicating that the equation for the unknown at the *ith* interior grid point has non-zero coefficients for the unknowns to the east, west, north, and south in the grid and of course, for the *ith* point itself. Equations for the unknowns associated with the bottom row of interior grid points, however, will have no south grid point coefficients.



Diagonals stencil	Stored κ's
i	0
i + 1	1
i - 1	-1
i + r	r
i - r	-r

Grid Stencil and Connectivity Set for Laplace's Equation

Figure 3.2

The coefficients for the N equations are assembled into an $N \times N$ matrix. For problems with one unknown per grid point, the structure of the assembled matrix is determined by an ordering of the grid points. However, for problems with more than one unknown per grid point there is more than one equation associated with each grid point and so we need to define an ordering in terms of the unknowns, rather than just the grid points, to determine the structure of the assembled matrix. We shall hereafter refer to orderings in terms of the unknowns rather than the grid points. The process of determining the structure of the assembled matrix given the ordering of unknowns is described in Appendix B. Our interest in this chapter is in choosing orderings which lead to efficient vectorization of both the matrix-vector multiplication in the conjugate gradient iteration and the forward and back solves in the preconditioning step of the ICCG algorithm. We will give neccessary and sufficient conditions to determine which p-color orderings of the unknowns correspond to p-color matrices, as described in chapter 2. We also indicate a process for choosing multi-color orderings which will maximize vector lengths within blocks of the p-colored matrices for class $\mathcal{R}$ problems and which is easy to apply for even the more difficult 3 dimensional problems. Examples are given from the model

problems to indicate how the multi-coloring techniques are applied. We also discuss the diagonal ordering which has been used to vectorize ICCG, showing why it is not as effective for computers like the CYBER 205.

3.1. The Natural Ordering

For two dimensional problems, the natural ordering is usually described as a left to right, bottom to top ordering of the grid points although several variants are equivalent. For example, orderings from bottom to top, right to left or right to left, top to bottom can also be considered natural orderings. The natural ordering we use will be the left to right, bottom to top ordering. For a three dimensional problem the planes of grid points are ordered from bottom to top with the unknowns in each plane ordered as described above. For problems with more than one unknown per grid point the unknowns at each grid point may be ordered consecutively or alternating. That is, we can order all the unknowns of one type consecutively following the natural ordering and then order the unknowns of another type again by the natural ordering and so on, or we can order the grid points by the natural ordering and alternate the various unknowns at each grid point. If there are two unknowns, u and v, at each grid point the two methods are :

$$\text{consecutive:}\quad u,u,u,u,\cdots,u,v,v,v,v,\cdots,v \tag{3.1}$$

$$\text{alternating:}\quad u,v,u,v,u,v,u,v,\cdots,u,v \tag{3.2}$$

Using (3.1) or (3.2) in conjunction with the natural ordering of the grid points, we number the unknowns and will hereafter refer to these unknowns as the w_i's. The main purpose for defining the unknowns, w_i, in this way is to describe the position of a given unknown in the grid independent of the order in which these unknowns appear in the linear system $A\mathbf{x}=\mathbf{b}$. For example,for a $r \times c$ grid with 3 unknowns per grid point, the second unknown associated with the fifth grid point in the third row is w_i, where $i = rc+2r+5$ when (3.1) is used, and

$i = 2(3r)+(4)(3)+2$ when (3.2) is used. The w_i will be the components of the vector $\mathbf{x}$ but not necessarily in the same order in which we have numbered the w_i. This is true, in particular, for the multicolor orderings to be discussed later.

For a given ordering of the unknowns w_i, as discussed above, we now define a connectivity set, Υ, of integers, κ, which indicate couplings between the w_i. Let P be any interior grid point and let w_i be one of the unknowns at P. Through the differential equations and their discretizations, there is an equation in terms of w_i and the remaining unknowns which represents the approximate partial differential equation at the grid point P. We say that w_i is coupled to w_j if there is a non-zero coefficient of w_j in the equation for w_i. We define the set Υ_P by

$$\Upsilon_P = \{ \kappa \mid w_{i+\kappa} \text{ is coupled to } w_i \text{ for all } w_i \text{ at } P\}$$

Then we define the connectivity set Υ by

$$\Upsilon = \bigcup_P \Upsilon_P$$

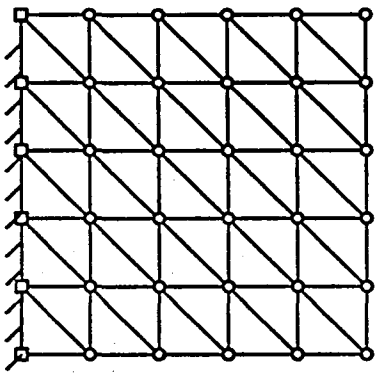
where the union is taken over all interior grid points. We observe that, because of the uniform grid stencil, $\Upsilon = \Upsilon_P$ for any grid point P_0 at which the grid stencil does not include a boundary point. Thus, we can obtain Υ by calculating Υ_{P_0} for any suitable P_0. Υ depends on the dimensions of the grid, the uniform stencil, the number of unknowns at each grid point, and the choice (3.1) or (3.2) of numbering these unknowns.

We now give an example for a class $\mathcal{R}$ problem having k unknowns per grid point on a $r \times c$ grid with a uniform stencil containing s points; note that there are kr unknowns in each row of the grid. If we use (3.2) in conjunction with the natural ordering of the grid points to number the unknowns, the north neighbors of the leftmost, or first, unknown, which we denote by w_i, at some grid point are the k unknowns w_{i+kr}, $w_{i+kr+1}, \cdots, w_{i+kr+k-1}$; the north neighbors of the second unknown, denoted by w_j, at the same grid point are w_{j+kr-1}, $w_{j+kr}, \cdots, w_{j+kr+(k-2)}$,

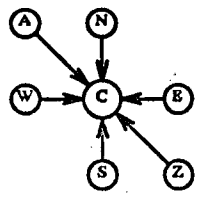
and the north neighbors of the kth unknown, w_h, at the same grid point are $w_{h+kr-k+1}, w_{h+kr-k+2}, \cdots, w_{h+kr}$. Thus, the set Υ will contain the $2k-1$ integers $kr-k+1, kr-k+2, \cdots, kr, kr+1, \cdots, kr+k-1$ corresponding to all of the north coefficients for the unknowns at this grid point. In general, for each other grid stencil point, for example, south, east, etc., Υ also will contain $2k-1$ κ's. However, the κ's arising from the s different grid stencil points need not all be different so that the set Υ, in general, will not contain $s(2k-1)$ values. If (3.1) is used in numbering the w_i there are still $2k-1$ κ's for each grid point but, for example, the values of κ for the north neighbors are $r-(k-1)rc$, $r-(k-2)rc$, $\cdots$, r, $r+rc$, $\cdots$, $r+(k-1)rc$.

The above discussion has indicated how to obtain Υ in general and we now give two complete examples to illustrate how the set Υ is obtained for a problem with one unknown per grid point and one with two unknowns per grid point. We consider first the Laplace model problem. Figure 3.2 shows the grid stencil for Laplace's equation and lists the κ's corresponding to each non-zero coefficient in the equation for w_i. The grid stencil is derived from the finite difference equations used to approximate the solution of the differential equation and the κ's in Υ are then determined using the number of points per row in the discretized domain. For instance, the north neighbor of grid point i in a grid with r points per row is the $(i+r)$th grid point so that the κ value for the north neighbor is r. Note that the κ for the north neighbor depends on the dimension, r, of the grid.

For an example of a system of partial differential equations, we consider a plane stress problem described in Adams [1983a] in which a plate is fastened to a rigid body along one side and a load is applied on the opposite side. A detailed description of the finite element solution for this problem is also given in Becker, et al. [1981]. The rectangular domain is discretized using triangular finite elements on which linear basis functions are defined (see Figure 3.3). At each grid point, there



Triangular Finite Elements



Grid Stencil

κ's for Plane Stress Problem

unknown	$j=2i-1$ u	$j=2i$ v
C_u	0	1
C_v	1	0
N_u	2r	2r-1
N_v	2r+1	2r
S_u	-2r	-2r-1
S_v	-2r+1	-2r
E_u	2	1
E_v	3	2
W_u	-2	-3
W_v	-1	-2
A_u	2r-2	2r-3
A_v	2r-1	2r-2
Z_u	-2r+2	-2r+1
Z_v	-2r+3	-2r+2

Plane Stress Problem

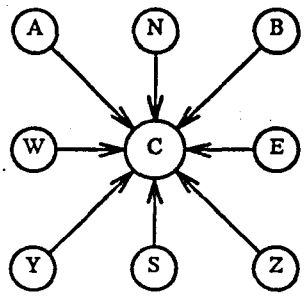
Figure 3.3

is a pair of equations for the displacements, u and v, in the x and y directions. We order the grid points according to the natural ordering and use the alternating pattern (3.2) for the u and v unknowns at each point. For this ordering, the u unknowns are odd numbered and the v unknowns are even numbered. Since both the u and v equations at each grid point are coupled to the u and v unknowns at the grid points indicated by the grid stencil, Υ contains values for both the u and v unknowns. For example, the u unknown at grid point i is w_j, where $j=2i-1$. It is coupled to the north u and north v unknowns as shown by the grid stencil in Figure 3.3. The north u unknown is the $(j+2r)$th unknown and the north v unknown is the $(j+2r+1)$st unknown. Similarily, the v unknown at grid point i is w_j, where $j=2i$. Note, however, that the north u and v unknowns are now the $(j+2r-1)$st and $(j+2r)$th unknowns. Thus, the set Υ contains the values $2r-1$,

$2r$, and $2r+1$ for the coupling of the north and center points. Figure 3.3 lists all the κ's for the plane stress problem, giving those associated with the u unknowns in one column and those associated with the v unknowns in another. Note that there are duplicate κ's in each column but some values appear in only one of the columns, such as the 3 for the E_v unknown in the u column and the -3 for the W_u unknown in the v column. Combining the two columns, we see that there are 17 different values of κ for the 7 point stencil shown in Figure 3.3. Although there are 17 different values in the set Υ, no more than 14 different values are associated with any one unknown. This is expected since there are only 14 unknowns corresponding to the grid stencil at any grid point and thus the equation for any w_i will contain at most 14 non-zero coefficients.

If, in the plane stress problem, we use the consecutive pattern (3.1) with the natural ordering of the grid points to number the w_i, the set Υ will contain different values and a greater number of distinct values. For example, the north u and v unknowns for the u unknown at grid point i, w_i, are now given by w_{i+r} and w_{i+rc+r} while the north u and v unknowns for the v unknown at grid point i, w_j, $j = i+rc$, are w_{j-rc+r} and w_{j+r}. The number of distinct values in Υ for this ordering is 21.

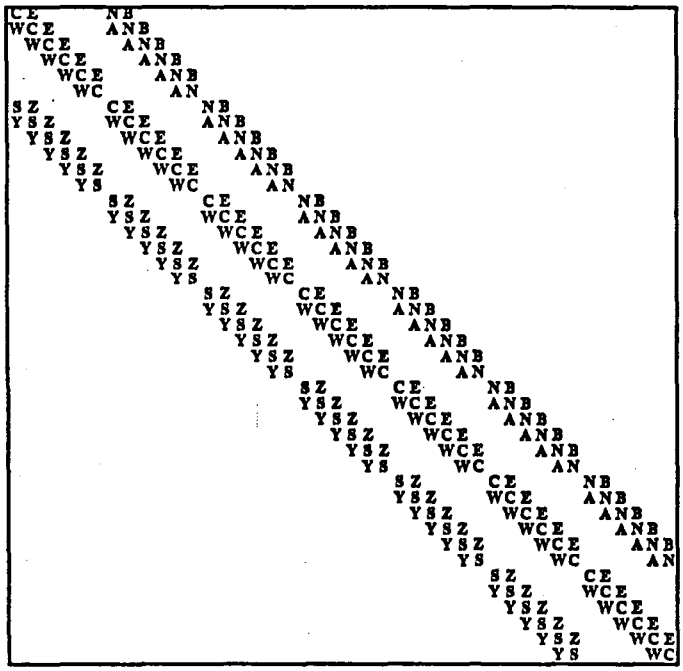
We now discuss, in general, the formation of the matrix A from the N equations in the unknowns w_i. We number the unknowns as already discussed but then the vector $\mathbf{x}$ in the linear system $A\mathbf{x} = \mathbf{b}$ is formed by assigning each w_i to a particular x_j in a one-to-one mapping. Having chosen a mapping of the w_i into $\mathbf{x}$ the structure of A is determined as follows. Each row i in A corresponds to the equation for the w_j that is assigned to x_i. Row i in the matrix will contain the non-zero coefficients for the equation for that w_j. For each $w_{j+\kappa}$ that has a non-zero coefficient in the equation for w_j the position of $w_{j+\kappa}$ in $\mathbf{x}$, denoted x_k, is determined. Then the entry in row i column k, $a_{i,k}$, is the coefficient for unknown



stencil	Diagonals Stored κ's	constraints
C	0	$i \neq qp$
E	1	$i+1 \neq qp$
A	r-1	$i+r-1 \neq qp$
N	r	$i+r \neq qp$
B	r+1	$i+r+1 \neq qp$
W	-1	
Z	-r+1	
S	-r	
Y	-r-1	

Grid Stencil and κ's for Mixed Derivative Problem

Figure 3.4



Mixed Derivative Problem - 6 × 7 Grid

Natural Ordering of the Unknowns

Figure 3.5

$w_{i+\kappa}$. In the next section we discuss assignments of the w_l's to $\mathbf{x}$ which are called

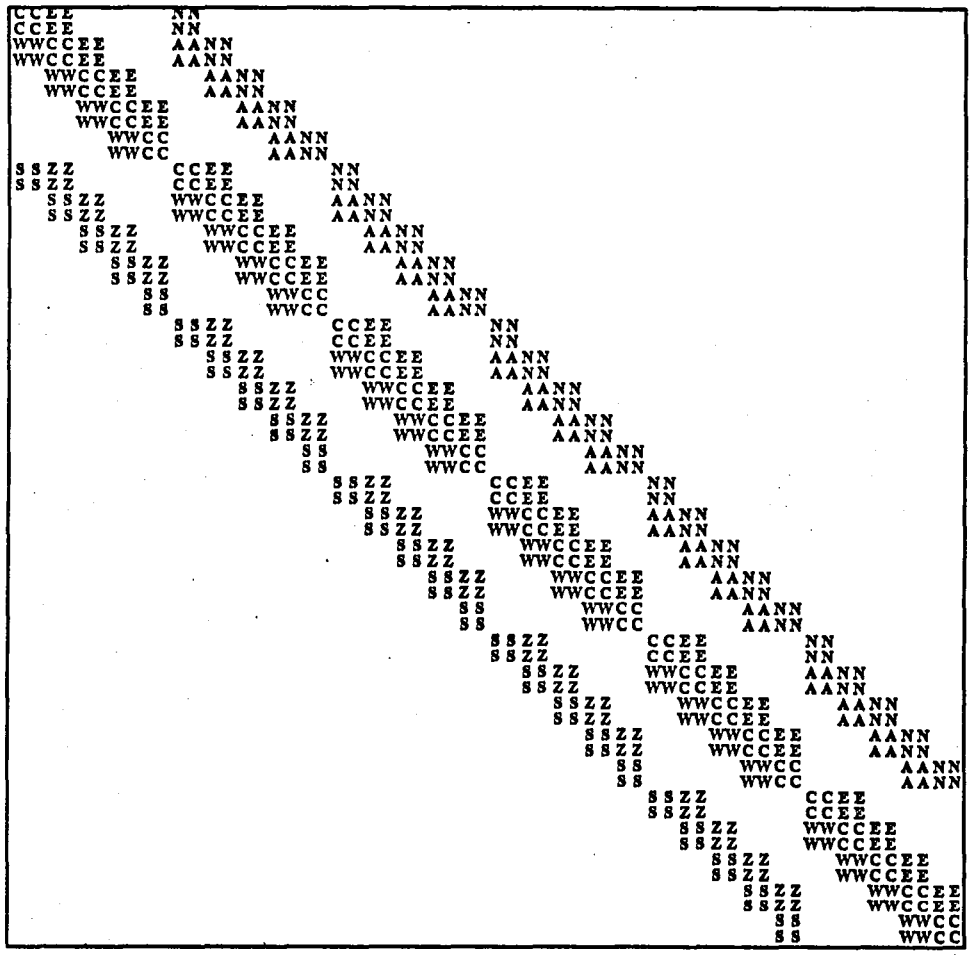
multi-color orderings of the unknowns. In this section we define the ordering of $\mathbf{x}$ by

$$x_i = w_i \ , \ 1 \leqslant i \leqslant N \tag{3.3}$$

If there is one unknown per grid point (3.3) defines the natural ordering of the unknowns which leads to the natural ordered matrix for the given problem. We illustrate the natural ordering for a problem with one unknown per grid point with an example from the mixed derivative problem. The grid stencil for the mixed derivative equation (1.3) is the 9 point star shown in Figure 3.4. The κ's in Υ are given for a grid with dimensions $r \times c$ ordered by the natural ordering as discussed in section 3.1. The constraints refered to in Figure 3.4 are discussed in a later next section. The 9 diagonal matrix shown in Figure 3.5 is the natural ordered matrix for the mixed derivative problem. Because of symmetry the matrix can be stored in 5 diagonals of length $O(N)$.

Figures 3.6 and 3.7 show the matrices for the plane stress problem where $\mathbf{x}$ has been ordered by (3.3) and the unknowns, w_i at each grid point have been numbered by the natural ordering of the grid points and (3.2) and (3.1), respectively. Recall that the set Υ contained 21 elements for the (3.1) numbering of the unknowns and 17 elements for the (3.2) numbering. Likewise, the matrix in Figure 3.6 contains 21 diagonals and the matrix in Figure 3.7 contains 17 diagonals.

Let us now consider vectorization of the matrix-vector multiplication. For this operation we store the matrix by diagonals and use matrix multiplication by diagonals, as previously stated. We want to choose an ordering of the unknowns, w_i, in $\mathbf{x}$ which will result in the minimum number of diagonals to store the matrix, thereby maximizing vector lengths. We will show that for class $\mathcal{R}$ problems the assignment (3.3) for problems with one unknown per grid point yields a matrix with the minimum number of diagonals and for problems with more than one unknown the number of diagonals in A is equal to the number of elements in the



51 52 53 54 55 56 57 58 59 60

.. ..

11 12 13 14 15 16 17 18 19 20

1 2 3 4 5 6 7 8 9 10

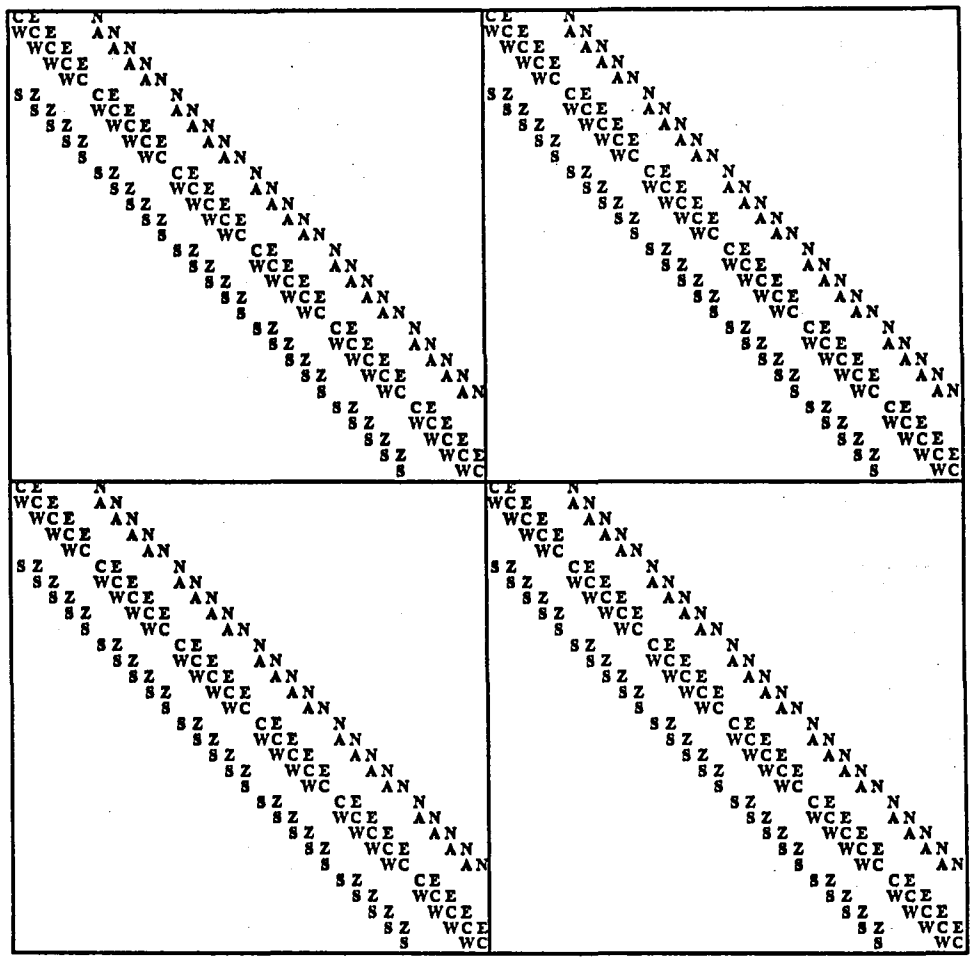
Plane Stress Problem - 5 × 6 Grid

Natural Ordering with (3.2)

Figure 3.6

set Υ.

For a class $\mathcal{R}$ problem with only one unknown per grid point and a uniform grid stencil with s points, the equation for any w_i for which the unknowns coupled by the grid stencil are not boundary points will contain s non-zero coefficients and



26 56 27 57 28 58 29 59 30 60

.. ..

6 12 7 14 8 16 9 18 10 20

1 31 2 32 3 33 4 34 5 35

Plane Stress Problem - 5×6 Grid

Natural Ordering with (3.1)

Figure 3.7

the number of elements in Υ, ν, is also equal to s. Hence, there are s coefficients in every row in A corresponding to one of these w_i and so there will be at least s diagonals in A. It is clear that this minimum number of diagonals, s, will not be achieved by arbitrary orderings of the w_i into x since the s non-zero coefficients in

succesive rows of A may not all lie along common diagonals. For class $\mathcal{R}$ problems with $k > 1$ unknowns per grid point, the minimum number of diagonals is not so easily asertained. We have already seen that each grid stencil point corresponds to $2k-1$ κ's in Υ so that $\nu \leqslant s(2k-1)$. Thus an upper bound on the number of diagonals in A which contain the non-zero coefficients for the equations for the k unknowns at any particular grid point is $s(2k-1)$. The actual number of non-zero coefficients in A for the k rows corresponding to any grid point may be much less than this bound for two reasons. First of all, in many problems several of the coefficients indicated by the grid stencil may be zero at every grid point. In the space truss problem which we consider in a later section, the element matrices for the truss elements that make up the finite element model for the 3 dimensional truss are themselves sparse and so many of the coefficients indicated by the stencil are zero. Secondly, it is generally the case that the κ's at each grid stencil point are not all different from the κ's at other grid stencil points, as we have already discussed in defining the set Υ. In any case, we seek orderings of the unknowns for which the number of diagonals in A is equal to the number of diagonals necessary to store all of the non-zero coefficients for the k equations of the unknowns at any *one* interior grid point. Thus, the number of diagonals in A would be given by ν, the number of elements in Υ. The following theorem states this result for the ordering of $\mathbf{x}$ by (3.3).

Theorem 3.1 :

Given a (2 or 3 dimensional) class $\mathcal{R}$ problem with a uniform grid stencil containing s points, the associated conectivity set Υ containing ν elements, and $\mathbf{x}$ ordered by (3.3), then

a) If there is one unknown per grid point and the unknowns w_i are numbered by the natural ordering of the grid points, the number of non-zero diagonals

in the matrix A is ν, and is equal to s. Moreover, ν is the minimum number of diagonals possible.

b) If there are $k > 1$ unknowns per grid point and (3.1) is used in conjunction with the natural ordering to number the unknowns, then the number of diagonals in A is given by ν, and is less than or equal to

$$(2k-1)s \tag{3.4}$$

c) If there are $k > 1$ unknowns per grid point and (3.2) is used in conjunction with the natural ordering to number the unknowns, then the number of diagonals in the matrix A is given by ν and is less than or equal to

$$k(s+1)+2(k-1)s_R + 1 \tag{3.5}$$

where s_R is the number of points in the grid stencil which come after the center point in the stencil as defined by the natural ordering and which have no left neighbor in the stencil.

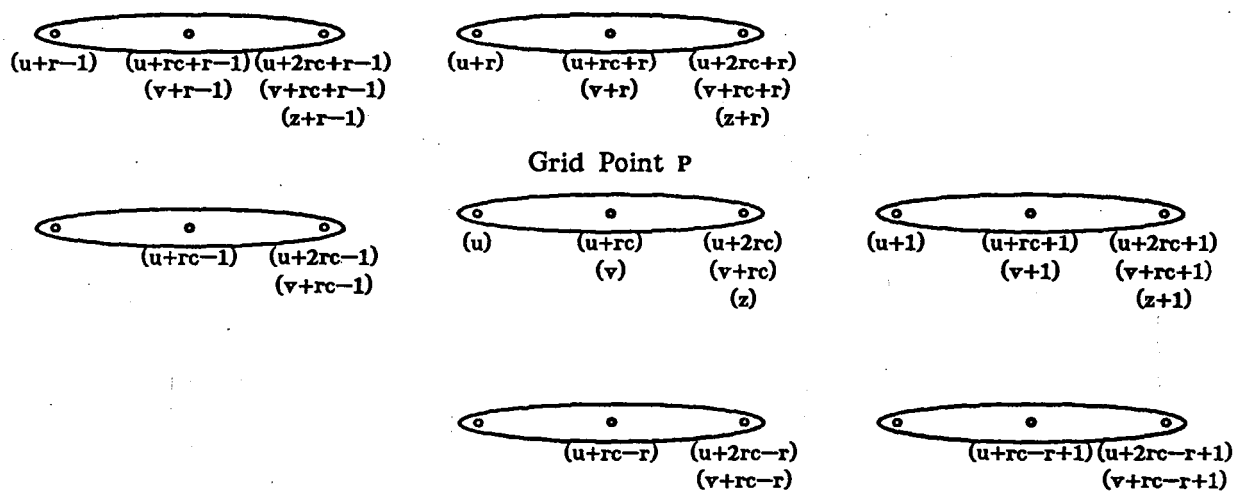
Proof :

a) By (3.3),the unknown w_i corresponds to row i in the matrix and each point in the grid stencil described by a $\kappa \in \Upsilon$ corresponds to the unknown $x_{i+\kappa}$ at grid point $i+\kappa$. Therefore, each κ gives rise to a diagonal in A of non-zero coefficients of the form $a_{i,i+\kappa}$, $1 \leqslant i, i+\kappa \leqslant N$. Since each grid stencil point corresponds to one and only one $\kappa \in \Upsilon$, the number of elements in Υ is s and there will be s diagonals in A, the minimum number possible.

b) Once again, by (3.3), $x_i = w_i$ and each κ in Υ corresponds to a diagonal in A with coefficients $a_{i,i+\kappa}$ so that the number of diagonals in A is ν. To derive the bound (3.4) on the number of diagonals in A it is sufficient to consider only diagonals in A which lie above the main diagonal since A is symmetric. We will show that the number of diagonals above the main diagonal in A is

$$(k-1)s + \lfloor s/2 \rfloor \tag{3.6}$$

where $\lfloor\ \rfloor$ and $\lceil\ \rceil$ denote the integer rounding functions that round down or up respectively if the quotient has any remainder. Since diagonals above the main diagonal are of the form $a_{i,i+\kappa}$, $\kappa>0$, we determine the number of κ's in Υ which are positive. Figure 3.8 represents 7 grid points coupled by a 7-point grid stencil in a two dimensional $r \times c$ grid where there are 3 unknowns per grid point numbered by the natural ordering in conjunction with (3.1). Each ellipse represents a grid point and the smaller circles within each ellipse are the three unknowns at each grid point. The center point, P, contains the 3 unknowns u, v, and z and the expressions below each unknown describe all the unknowns which are coupled to P and are numbered after the particular unknown at P. Notice that since all of the middle and rightmost unknowns at the grid points are numbered after all of



7 Point Stencil for a Problem with k=3 Using (3.1)

Figure 3.8

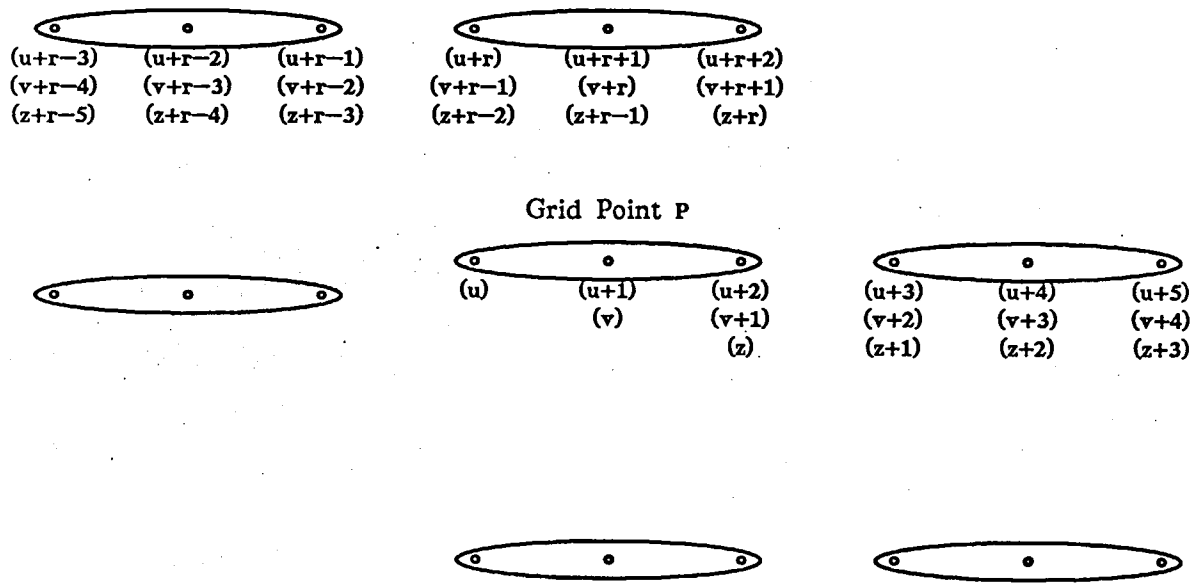
the leftmost unknowns, the κ's corresponding to the coupling between point P and all of the middle and right unknowns are positive and the coefficients $a_{i,i+\kappa}$ all lie above the main diagonal. In general, for k unknowns per grid point and s stencil points the number of non-zero coefficients corresponding to u for any grid point P is given by $s(k-1)$. In addition, the leftmost unknowns at the grid points to the right and above u also are numbered after u and the corresponding coefficients lie above the main diagonal in A. In general there are $\lfloor s/2 \rfloor$ stencil points to the right and above the center grid point for a grid stencil that corresponds to a symmetric matrix. Therefore, the total number of diagonals above the main diagonal in A is bounded by (3.6). Now since A is symmetric the grid stencil will have an odd number of points so that $2\lfloor s/2 \rfloor = s-1$ and the total number of diagonals in A is bounded by

$$2(k-1)s + 2\lfloor s/2 \rfloor + 1 = (2k-1)s$$

c) Again, by (3.3), each $\kappa \in \Upsilon$ corresponds to a diagonal in A with coefficients $a_{i,i+\kappa}$ so there are ν diagonals in A. As before, we count the number of diagonals above the main diagonal by counting the number of positive κ's in Υ for a point P in the grid. Figure 3.9 again represents 7 grid points coupled by a 7-point grid stencil, as above, but where the unknowns w_i are numbered by (3.2). Now all κ's for unknowns to the right and above u are positive. In general, the number of unknowns to the right and above the leftmost unknown at point P is

$$k\lfloor s/2 \rfloor - 1 \qquad (3.7)$$

Notice, however, that in Figure 3.9 the expressions for the unknown v at P include a κ that was not among the κ's for u, namely the κ for the leftmost unknown in the top left grid point in Figure 3.9, $\nu+r-4$. The coefficient indicated for this coupling lies in diagonal $a_{i,i+r-4}$, a diagonal not



7 Point Stencil for a Problem with k=3 Using (3.2)

Figure 3.9

counted in (3.7). In general, if there are k unknowns at each grid point, then for every grid stencil element to the right and above the center grid stencil point that does not have a left neighbor in the stencil, there are $k-1$ additional positive κ's in Υ. These are the s_R points in (3.5) and we note that $s_R \leqslant \lfloor s/2 \rfloor$. This completes the proof.

For class $\mathcal{R}$ problems with only one unknown per grid point, Theorem 3.1 states that we can do no better than the natural ordering for diagonal storage of the matrix. For more than one unknown per grid point, the number of diagonals in A is given by ν, the number of elements in Υ. Note that since $s_R \leqslant \lfloor s/2 \rfloor$, (3.5) is less than or equal to (3.4). Therefore, (3.2) in conjunction with the natural ordering will always require no more diagonals than (3.1) with the natural ordering. However, the minimum number of diagonals is not always the best all around

choice; another consideration is the amount of storage required for each ordering. Note that by (3.4) the number of diagonals in the plane stress matrix is less than or equal to 21 while (3.5) indicates that using (3.2) with the natural ordering will require no more than 17 diagonals. In Figures 3.6 and 3.7 the matrices shown do contain 17 and 21 diagonals, respectively. However, in Figure 3.6 storing A in 17 diagonals requires that $O(N/2)$ zeros be stored for many of the diagonals, for example, the lower S diagonal. Thus, a total of $O(9N)$ memory locations is required if we store only the upper or lower triangular part of A. Alternatively, one may chose not to store the alternating zeros, in which case execution time for a matrix vector multiply is slowed considerably since costly gather-scatter operations must be performed each time such a diagonal is used in the matrix vector multiply. The matrix in Figure 3.7, on the other hand, shows that the (3.1) ordering requires 21 diagonals but only $O(7.5N)$ memory locations.

3.2. The Coloring Problem

We now consider the more difficult problem of obtaining orderings of the w_i which result in p-color matrices as well as maximize vector lengths. A p-color ordering can be described as a partitioning of the unknowns, w_i, into p disjoint sets, S_τ , $1 \leqslant \tau \leqslant p$. The unknowns in each S_τ are assigned to $\mathbf{x}$ consecutively beginning with the w_i's in S_1, followed by S_2 and so on. If the p-coloring has the property that no elements in S_τ are neighbors, as defined by the set Y, the corresponding matrix will be a p-color matrix, that is, it will have the form (1.1) with the diagonal blocks themselves diagonal matrices. Determining the coloring of the unknowns that corresponds to a p-color matrix for an arbitrary grid stencil using the smallest possible number, p, of colors is a graph coloring problem which, in general, is NP-complete. (See, e.g., Horowitz and Sahni [1978]). For many problems of interest in scientific and engineering applications, however, colorings have

been given which result in the desired p-colored matrices. (See, e.g., Adams [1983a]). In Chapter 2 we used a particular multi-color ordering to obtain a p-colored matrix which was used to implement a block incomplete Cholesky preconditioning with the desired long vector lengths. However, for a given problem with its associated grid stencil, there may be several different coloring schemes that result in p-colored matrices, and we wish to determine those orderings which maximize vector lengths. Schreiber and Tang [1982] claim that the ordering chosen should achieve the p-color matrix form with the smallest p possible. We will show that additional factors must be considered and in fact that using more than the minimum p colors may result in more efficient vectorization for some problems.

We now state and prove the following theorem, which gives necessary and sufficient conditions to ensure that the matrix corresponding to a p-color ordering will be a p-color matrix.

Theorem 3.2 : The P-Coloring Theorem

Given a (2 or 3 dimensional) class $\mathcal{R}$ problem with connectivity set Υ associated with a given uniform grid stencil, and a p-color ordering of the unknowns, w_i, (one or more per grid point) into p disjoint sets, S_τ, $1 \leqslant \tau \leqslant p$, the corresponding matrix, A, is a p-color matrix if and only if the following condition is true for every unknown w_i:

if $w_i \in S_\tau$ *then* $w_{i+\kappa} \notin S_\tau$ *for every* $\kappa \in \Upsilon$, $\kappa \neq 0$ *such that* κ *corresponds to an unknown* $w_{i+\kappa}$ *coupled to* w_i.

Proof :

The matrix A is p-colored if and only if each diagonal block, $A_{j,j}$, is diagonal. If $w_{i+\kappa} \in S_\tau$ for some $w_i \in S_\tau$, and $\kappa \neq 0$ then the row in $A_{\tau,\tau}$ corresponding to i will have a non-zero coefficient in an off diagonal position so the matrix is not p-colored. Conversely, if the above condition is satisfied

for all i, then there will be no off diagonal entries in any of the $A_{j,j}$, $1 \leqslant j \leqslant p$, and the matrix will be a p-color matrix.

In the next section we give an example of a matrix which arises from a multi-color ordering but is not a p-color matrix and we will see that it violates the condition of Theorem 3.2.

We now consider the other important characteristic of p-color matrices that makes the long vector operations in the forward and back solves in the preconditioning step of ICCG possible, namely $O(N/p)$ length vectors within the blocks of A. Given a coloring which results in a p-color matrix, if we order the w_i's within the S_τ's randomly, we will almost certainly not obtain $O(N/p)$ length vectors in A. On the other hand, the ordering of the w_i in S_1 may be arbitrary as long as the remaining w_i's in the S_τ's, $2 \leqslant \tau \leqslant p$, correspond to the w_i's in S_1 in the manner described in Figure 3.10.

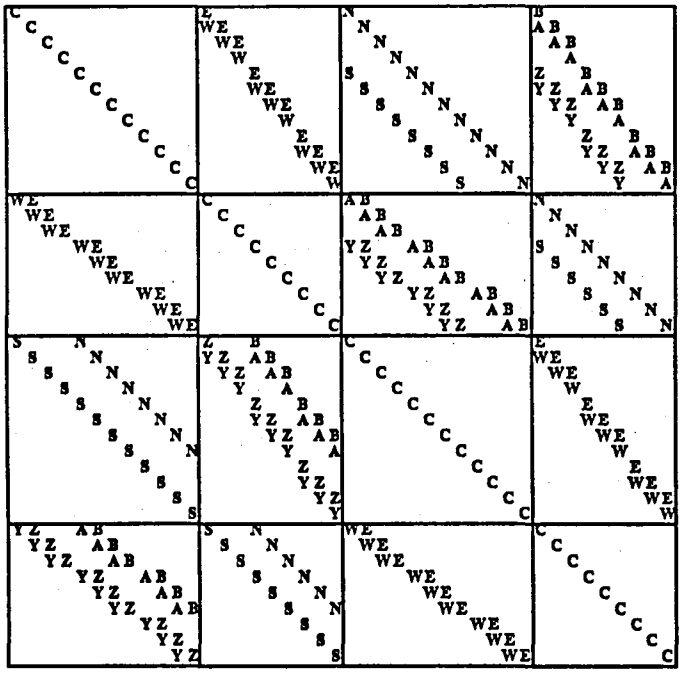
Let us now consider an example of Theorem 3.2 and the condition in Figure 3.10. For the mixed derivative problem, we can write the S_i explicitly as shown in Figure 3.11 for a 4-color ordering of a 7×6 grid The first 12 rows in the matrix in Figure 3.11 correspond to the 12 unknowns in S_1, in the order shown. Likewise, the next 9 rows correspond to the unknowns of S_2, and so on. For the 7×6 grid the set Υ is $\{0,1,6,7,8,-1,-6,-7,-8\}$.

For each κ in Υ, all of the unknowns, $w_{i+\kappa}$, coupled to w_i's in S_1 are in one and only one of the S_τ , $\tau \neq 1$, which we denote by S_K. Furthermore, the unknowns in S_K denoted by $w_{k1}, w_{k2}, \cdots$, which are coupled by a particular κ to unknowns in S_1, which we will denote as $w_{s1}, w_{s2}, \cdots$, are coupled in the following way:

$$w_{ki} \text{ is coupled } w_{si+h} \text{ , } h \text{ is a constant.}$$

Multi-Color Ordering Constraint

Figure 3.10



.	.	.	.	.	.	.
.	.	.	.	.	.	.
3	4	3	4	3	4	3
1	2	1	2	1	2	1
3	4	3	4	3	4	3
1	2	1	2	1	2	1

Mixed Derivative Problem - 7 × 6 Grid

$S_1 = \{1,3,5,7,15,17,19,21,29,31,33,35\}$, $S_2 = \{2,4,6,16,18,20,30,32,34\}$

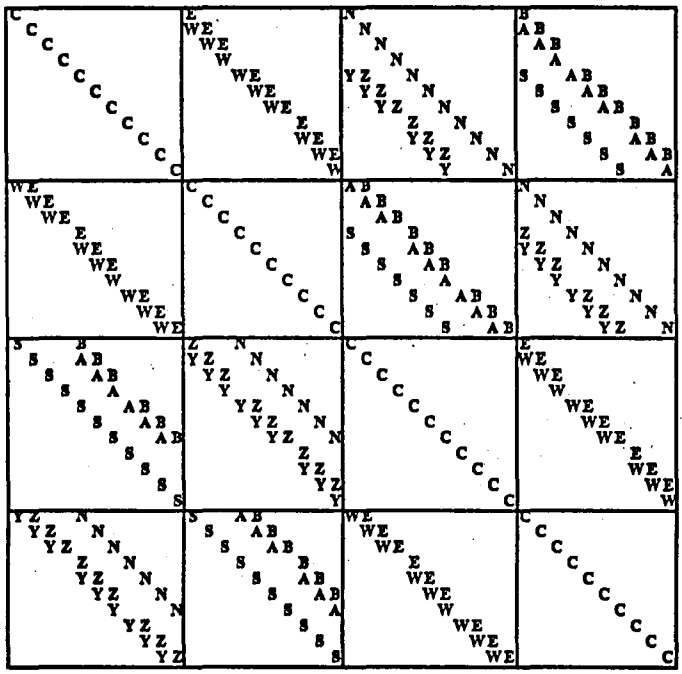
$S_3 = \{8,10,12,14,22,24,26,28,36,38,40,42\}$, $S_4 = \{9,11,13,23,25,27,37,39,41\}$

4 Color Ordering - *O(r)* Length Vectors

Figure 3.11

The fact that Theorem 3.2 is satisfied by the p-color ordering in Figure 3.11 is evident from the picture of the grid. At any grid point, P, the surrounding neighbors in the 9 point stencil are different colors and hence are not in the S_i containing w_i. We can also apply the condition of Theorem 3.2 directly for each unknown. For example, the equation for the first unknown has non-zero coefficients corresponding to the N, B, and E grid stencil elements. The κ's from the set Υ for the N, B, and E grid stencil elements are the integers 7, 8, and 1. To satisfy the condition in Theorem 3.2, unknowns 8, 9, and 2 cannot be in the set S_1. Note that at a grid point which has boundary points as neighbors, only those κ's in Υ which correspond to non-boundary points are used in the test. For example, if we calculate the Z coefficient of grid point 7 for which $\kappa = 8$ we get w_{15}. This unknown is in set S_1 which would violate the condition in Theorem 3.2. However since grid point 7 does not have an Z coefficient due to the boundary of the domain, we do not test point 7 with $\kappa = 8$. For problems with more than one unknown per grid point, we have already seen that some of the κ's apply only to one of the unknowns at each grid point while some apply to all of the unknowns.

Although the diagonals corresponding to the N and S grid stencil elements line up within each block of the matrix in Figure 3.11, the remaining diagonals for each grid stencil element do not. For the unknowns in S_1, the coefficients corresponding to the north grid stencil are all in S_3 and each element in S_3 is associated with the elements in S_1 consecutively. Note that, in the notation described in Figure 3.10, w_{si} is coupled to w_{ki} where $k=3$ and, here, $h=0$. The coefficients corresponding to the east grid stencil element are all in S_2 but the elements in S_2 do not correspond to consecutive elements in S_1. Note that the first 3 elements in S_2 correspond to the first 3 elements in S_1 ($h=0$) and the first E diagonal in block 1,2 of the matrix in Figure 3.11 has length 3. However, the 4th thru 7th elements in S_2 correspond to the 5th thru 8th elements in S_1 ($h=1$) and the corresponding E diagonal in block



.	.	.	.	.	.	.
.	.	.	.	.	.	.
4	3	4	3	4	3	4
2	1	2	1	2	1	2
3	4	3	4	3	4	3
1	2	1	2	1	2	1

Mixed Derivative Problem - 7 × 6 Grid

$S_1 = \{1,3,5,7,16,18,20,29,31,33,35\}$, $S_2 = \{2,4,6,15,17,19,21,30,32,34\}$

$S_3 = \{8,10,12,14,23,25,27,36,38,40,42\}$, $S_4 = \{9,11,13,24,26,28,37,39,41\}$

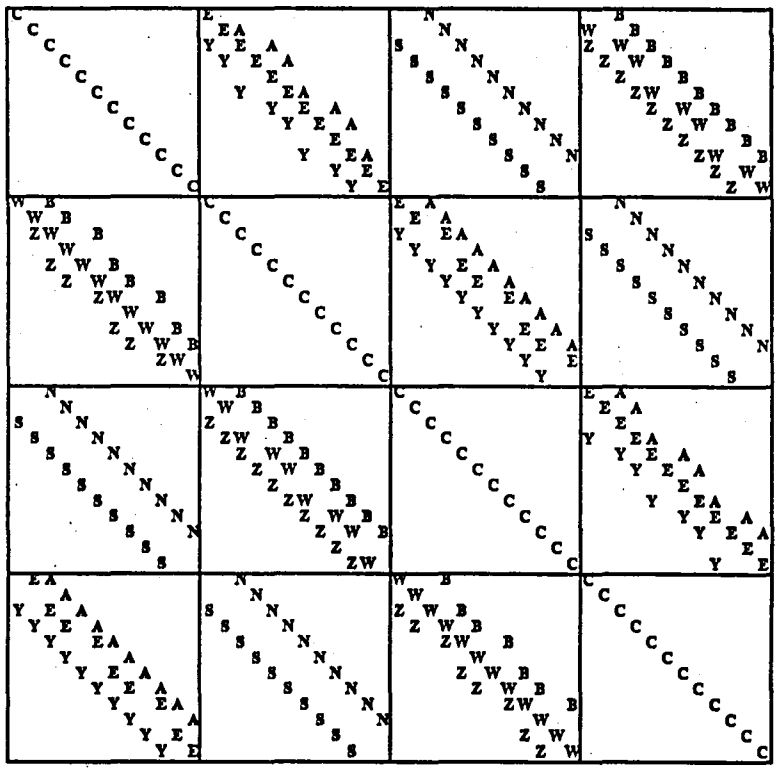
4 Color Ordering - *O(N/4)* Length Vectors

Figure 3.12

1,2 lies below the first E diagonal. This shift of the E diagonals occurs because in each row containing points colored 1 and 2 there is one more color 1 point than color 2 points. The vector lengths of the E diagonals are $O(r/4)$ rather than the desired $O(N/4)$.

One solution to this problem is to change the coloring pattern as shown in Figure 3.12. Row 3 now begins with color 2 and row 4 begins with color 4. This 4 row pattern is maintained throughout the grid. One can verify for each S_i shown in Figure 3.12 that the condition in Theorem 3.2 is satisfied by this p-coloring. Note that the E coefficients for the unknowns in S_1 are again in S_2 and the first 3 unknowns in S_2 correspond to the first 3 unknowns in S_1. The 4th unknown in S_2 does not correspond to the 4th unknown in S_1 but the 4th unknown in S_1 is on the right border of the grid and has no E coefficient. This extra element in S_2 allows the 5th unknown in S_2 to correspond to the 5th unknown in S_1 and the condition of Figure 3.10 is satisfied. As a result, the E diagonal in the first block row of A is of length $O(N/4)$, as desired. The same result is true for all of the diagonals in Figure 3.12 and the vector lengths for this p-color matrix are all $O(N/4)$.

As a final example, we give in Figure 3.13 another multi-color ordering which does not meet the conditions given in Figure 3.10 for the minimum number of diagonals. Notice that here we color each row continuously but the pattern is not continued from one row to the next. The alternating pattern in some of the diagonals in the 4-color matrix in Figure 3.13 are $r/4$ length diagonals which can be stored contiguously if $r/4$ zeros are added between each $r/4$ vector. For instance, the A, Y, and W diagonals in block row 1 can be stored in 6 vectors of length $O(N/4)$ but $O(N/4)$ zeros must also be stored for each vector, introducing undesired overhead in storage and computations. Clearly, this matrix does not have the minimum number of diagonals within each block.



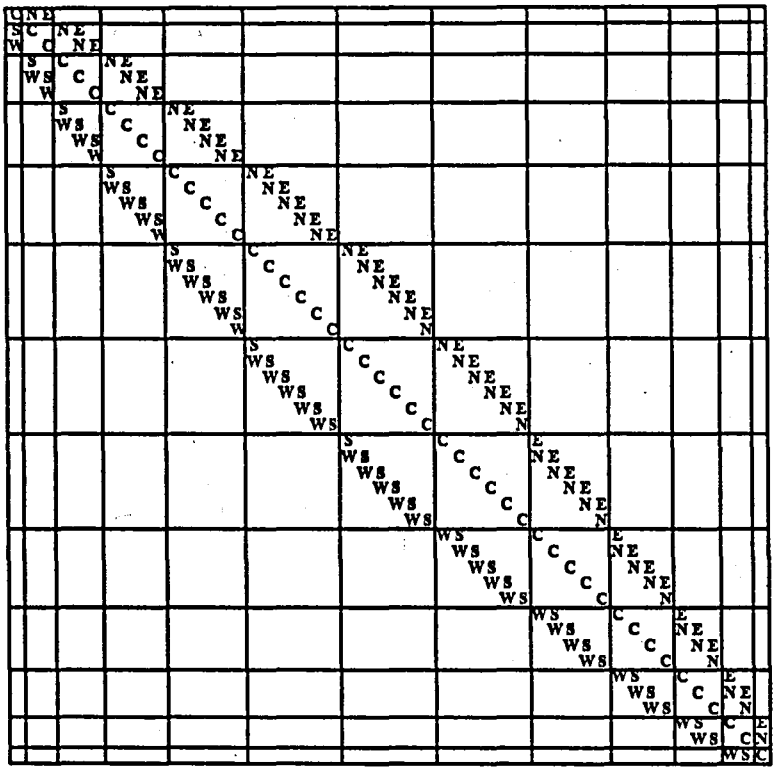
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
3	4	1	2	3	4	1	2
1	2	3	4	1	2	3	4
3	4	1	2	3	4	1	2
1	2	3	4	1	2	3	4

Mixed Derivative Problem - 8 × 6 Grid

4 Color Ordering - $O(r/4)$ Vector Lengths

Figure 3.13

We now consider another type of ordering, called the diagonal ordering, which can be considered a type of multi-coloring where p is no longer a constant independent of the grid dimensions but, rather, is given by $p = r + c - 1$. This



8 9 10 11 12 13

. . . . . .

3 4 5 6 7 8

2 3 4 5 6 7

1 2 3 4 5 6

Laplace Model Problem - 6 × 8 Grid

Diagonal Ordering

Figure 3.14

ordering, shown in Figure 3.14 for a 6 × 8 grid for the Laplace problem, has been used to vectorize ICCG by several authors, for example, van der Vorst [1983] and Schreiber and Tang [1982]. There are 13 colors and the 13 sets are formed by

assigning unknowns along diagonals in the grid to distinct sets. For example, $S_1 = \{w_1\}$, $S_2 = \{w_7, w_2\}$, $S_3 = \{w_{13}, w_8, w_3\}$, and so on. Since the grid stencil for Laplace's equation does not couple any grid points along diagonals, it is obvious that the matrix corresponding to the diagonal ordering will be a p-color matrix. This fact can also be deduced from the condition in Theorem 3.2. One can also verify that the condition in Figure 3.10 is satisfied by this ordering although in this case the maximum vector lengths are only $O(r)$. The main drawback of this ordering for vector computers such as the CYBER 205 is that the vector lengths in the matrix in Figure 3.14 are of average length $r/2$ and the longest diagonal within any block row is length $min(r,c)$. For rectangular regions this length is particularily bad but even on square regions the vectors are not long enough for effecient vectorization unless the problem size is very large. Another drawback of the diagonal ordering is that it does not achieve p-color matrices for even slightly more complex stencils and more complicated 'diagonal-like' orderings are needed. For threee dimensional problems diagonal-like orderings are even more difficult to construct.

3.3. The Continuous Coloring Rule

Theorem 3.2 gives necessary and sufficient conditions for a multi-color ordering to yield a p-color matrix. We also discussed in the previous section a criteria (Figure 3.10) to obtain $O(N/p)$ length vectors within the block rows of the p-colored matrix. But these criteria do not suggest a strategy to follow to obtain this result in general. We next describe a process we call the *continuous coloring rule* which is easy to use on any class $\mathcal{R}$ problem in order to obtain p-color matrices with $O(N/p)$ length vectors in the blocks of A. In the continuous coloring rule for p colors, we go through the grid points by the natural ordering assigning the p colors to the unknowns as : $1,2,3,\cdots,p,1,2,3,\cdots,$. This is equivalent to forming

p disjoint sets

$$S_\tau = \{\tau, \tau+p, \tau+2p, \tau+3p, \cdots\}, \quad \tau = 1, 2, \cdots, p$$

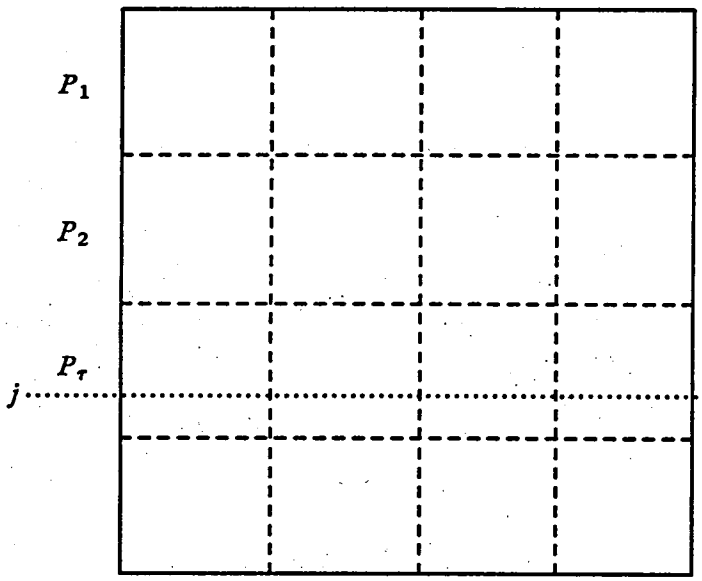
where the grid points are ordered by the natural ordering with (3.2) used when there is more than one unknown per grid point. If $i = qp+\tau$, with $1 \leqslant \tau \leqslant p$, then unknown i is in set S_τ and its position in S_τ is $q+1$. The east neighbor of point i is in S_γ where $\gamma = (i-1) \, mod \, p + 1$, the north neighbor is in S_γ, $\gamma = (i+r-1) \, mod \, p+1$, and so on. Since the unknowns are partitioned by assigning every pth unknown to the same set, the number of points in each set is N/p if p divides N evenly. Otherwise the first $N \, mod \, p$ sets will contain $\lfloor N/p \rfloor$ unknowns and the remaining sets will contain $\lfloor N/p \rfloor$ points. We can also express the number of points in the set S_τ as $P_\tau = \lfloor (N-\tau+p)/p \rfloor$.

To determine the matrix row, j, corresponding to some unknown $i = qp+\tau$, we sum the number of elements in each of the sets S_k , $k < \tau$: $\sum_{k=1}^{\tau-1} P_k$. The position of unknown i within S_τ is $q+1$. Therefore, the unknown $i = qp+\tau$ corresponds to row $j = \sum_{k=1}^{\tau-1} P_k + q + 1$ in the multi-colored matrix. This is illustrated in Figure 3.15.

We will refer to multi-color orderings derived using the continuous coloring rule as continuous color orderings. We now state and prove esssentially a corollary of Theorem 3.2 which applies to continuous p-color orderings and gives necessary and sufficient conditions for the matrix corresponding to a continuous color ordering to be p-colored.

Theorem 3.3 :

Given a class R problem with connectivity set Υ obtained by the continuous coloring rule using p colors, the matrix corresponding to a continuous color ordering of the unknowns is a p-colored matrix if and only if for $\kappa \neq 0$ in Υ,



P-Colored Matrix

Figure 3.15

$$\kappa \bmod p \neq 0 \tag{3.8}$$

Proof :

If $\kappa \bmod p = 0$ for some $\kappa \in \Upsilon$, then $i+\kappa$ is in the same S_τ as i so by Theorem 3.2 the the matrix is not p-colored. Conversely if for each $\kappa \in \Upsilon$, $\kappa \neq 0$, (3.8) holds, then $(i+\kappa) \bmod p \neq i \bmod p$ for any i. Therefore i and $i+\kappa$ are not in the same set and by Theorem 3.2 the matrix is p-colored.

We also observe that the condition in Figure 3.10, given in the previous section for obtaining $O(N/p)$ length vectors in A, is satisfied by any continuous color ordering. This is easily seen when we observe that every unknown that is colored τ (i.e. is in set S_τ) has neighboring unknowns that are determined by the set Υ as $(\tau+\kappa-1) \bmod p + 1$ so that all the unknowns for a particular κ that are coupled to the w_i in set S_τ are in some S_K , $K \neq \tau$. Furthermore the sets S_τ are formed with the same sequencing of the unknowns within each set so the consecutive

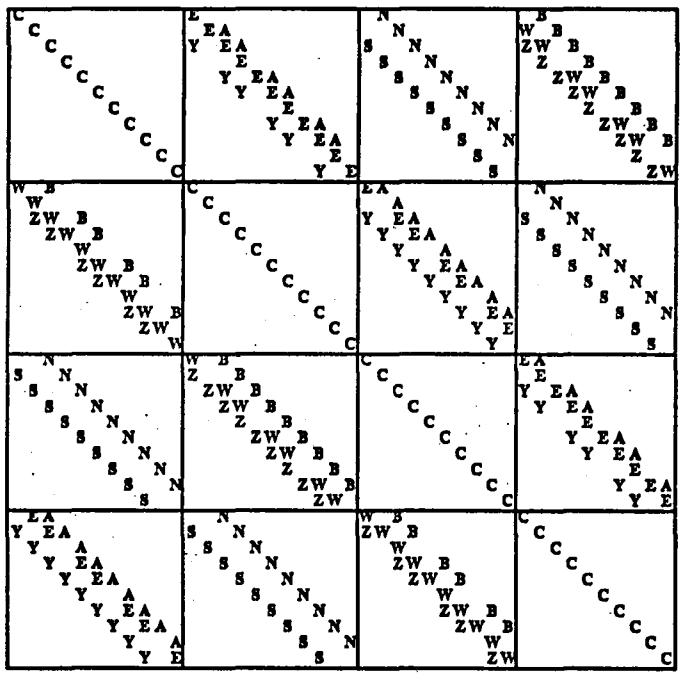
correspondence of coupled unknowns for each κ is also satisfied.

For many problems, applying the constraint (3.8) makes the minimum p necessary to achieve a multi-color ordering immediately obvious. For more complicated stencils, and especially for 3 dimensional problems, an easy and effective strategy to follow is to estimate a value for p and then apply (3.8) until a conflict is noted. If a conflict occurs one can increase p and repeat the procedure. In our experience the minimum p is found quickly after a few trials.

3.4. Multi-Coloring Examples from the Mixed Derivative Problem

We next give examples from the mixed derivative problem to illustrate how to apply Theorem 3.3 to obtain a p-color matrix. We will discuss several strategies to use when the given dimensions of the grid do not fit the constraints given in (3.8) of Theorem 3.3.

If we color the grid using p colors and the continuous coloring rule, the constraints (3.8) of Theorem 3.3 are as shown in Figure 3.4. Note that for symmetric matrices $\kappa \in \Upsilon$ implies that $-\kappa \in \Upsilon$ so that we need only consider positive κ's to obtain a complete set of constraints. The 3 constraints on r in Figure 3.4 require that $p > 3$ since for any integer q, one of the integers q, q+1, or q+2 is always divisible by 3. If we chose $p = 4$ we have $r \neq 4q+1$, $r \neq 4q$, and $r \neq 4q+3$ so by Theorem 3.3 a continuous 4-coloring exists for the mixed derivative problem if and only if $r = 4q+2$. Figure 3.16 is a 4-color matrix for the mixed derivative problem for a 6×7 grid. Note that within each block of the matrix, diagonals corresponding to each κ in the grid stencil have the same offsets and thus can be stored contiguously as one vector. If we color the grid in the same fashion with $r = 4i+3$ the resulting matrix shown in Figure 3.17 is not a p-color matrix. Note that the number of unknowns is the same for Figures 3.16 and 3.17 but the ordering in Figure 3.17 does not satisfy the constraints (3.8). The



.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
3	4	1	2	3	4
1	2	3	4	1	2
3	4	1	2	3	4
1	2	3	4	1	2

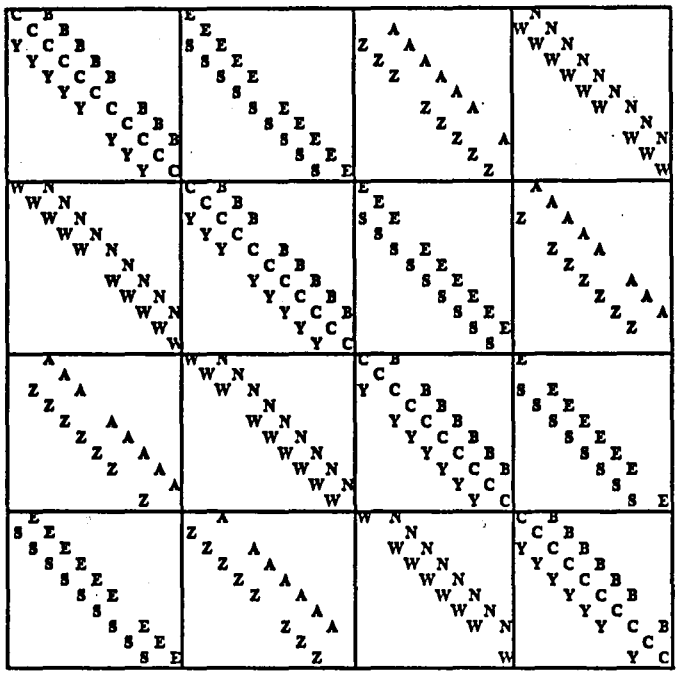
Mixed Derivative Problem - 6 × 7 Grid

Continuous 4-Coloring - Satisfies (3.8)

Figure 3.16

diagonal blocks in Figure 3.17 are now tridiagonal and thus not suitable for the multi-color ICCG algorithm we are considering. We will discuss possible uses for this ordering in chapter 5 as an extension of the basic multi-color ICCG algorithm.

If we increase p to 5 colors, the 7×6 grid can be colored by the continuous coloring rule and the 5-color matrix shown in Figure 3.18 is obtained. The vector



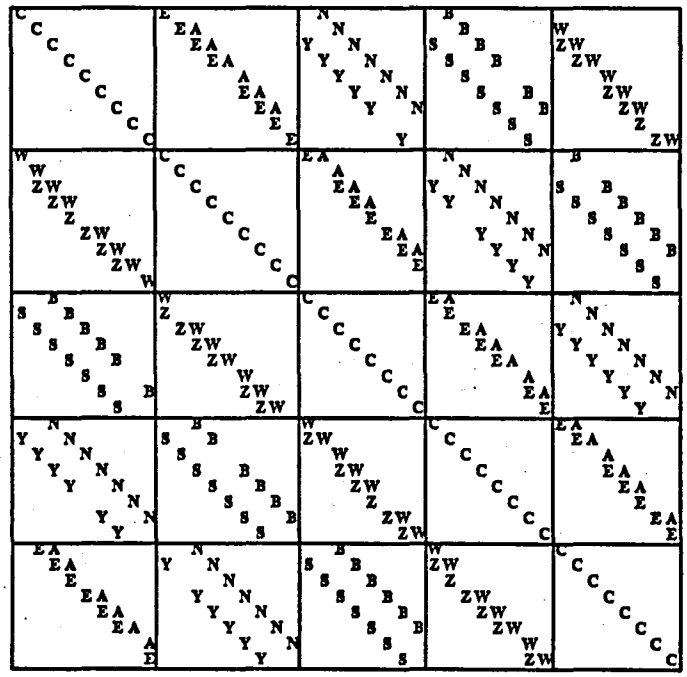
.	.	.	.	.	.	.
.	.	.	.	.	.	.
2	3	4	1	2	3	4
3	4	1	2	3	4	1
4	1	2	3	4	1	2
1	2	3	4	1	2	3

Mixed Derivative Problem - 7 × 6 Grid

Continuous 4-Coloring - Violates (3.8)

Figure 3.17

lengths are shorter than for 4 colors but for large N the difference is insignificant. In general, for a given r we can find a p for which the continuous coloring rule will result in a p-color matrix with the desired long vectors within the individual blocks.



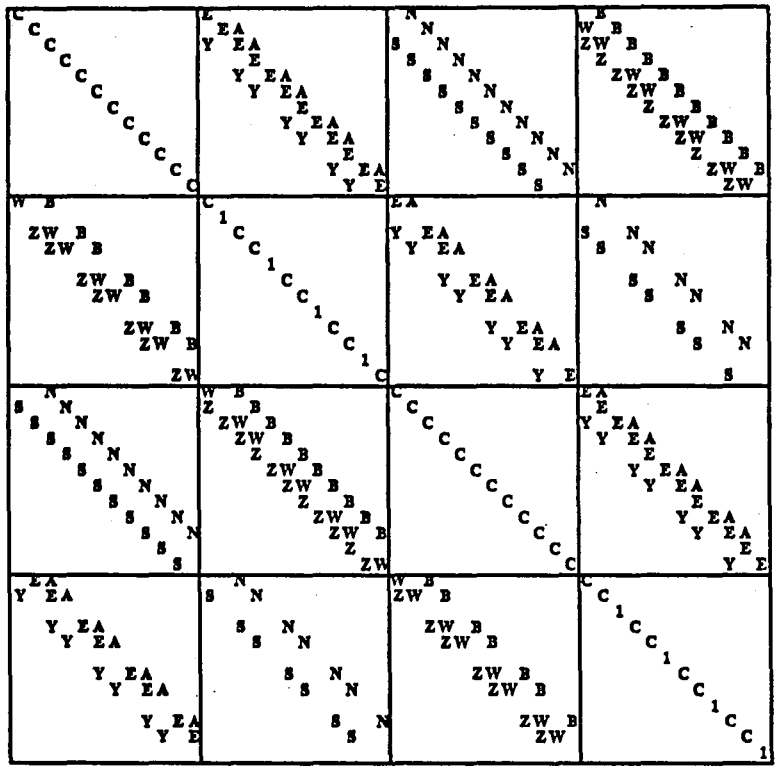
.	.	.	.	.	.	.
.	.	.	.	.	.	.
2	3	4	5	1	2	3
5	1	2	3	4	5	1
3	4	5	1	2	3	4
1	2	3	4	5	1	2

Mixed Derivative Problem - 7 × 6 Grid

Continuous 5-Coloring - Satisfies (3.8)

Figure 3.18

Another way to satisfy the constraints of Theorem 3.3 is to add additional 'dummy' rows or columns in the grid to meet the restrictions on r (or possibly c also for 3 dimensional problems). Figure 3.19 illustrates how one dummy column added to a $4i + 1$ grid causes the vectors within blocks to line up. The "0" points in the grid represent the dummy column and are considered to be the color dictated



.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
3	4	1	2	3	0
1	2	3	4	1	0
3	4	1	2	3	0
1	2	3	4	1	0

Mixed Derivative Problem - 5 × 8 Grid

Continuous 4-Coloring - Add Extra Column

Figure 3.19

by the continuous coloring rule. For example, the 0 node in row one is color number 2 while in row 2 the 0 node is color number 4. The equations for each point are just the identity $x_j = 0$ and the zero rows consist of a 1 on the main

diagonal. Extra storage and useless calculations are required by this scheme so this method may not be as appealing as increasing the number of colors to satisfy the requirements of (3.8).

3.5. Super Long Vectors in p-Color Matrices

We have shown so far how to color the grid associated with the stencil in Figure 3.4 so that diagonals within the blocks of the p-color matrix line up into vectors of length $O(N/p)$. Another consideration in choosing an ordering for the ICCGC algorithm is the matrix-vector multiplication which is required at each conjugate gradient iteration. This process is described in Appendix A. In the natural ordering of the grid points for the mixed derivative problem the matrix A can be stored in five $O(N)$ length vectors, using symmetry. A matrix-vector multiply using multiplication by diagonals would require nine vector multiplies and eight vector adds. For the matrices in Figures 3.12 and 3.16 sixteen $O(N/p)$ and one $O(N)$ length vectors are stored. A matrix-vector multiply now requires 32 $O(N/p)$ and 1 $O(N)$ length vector multiplies and 32 $O(N/p)$ length vector adds. The total number of operations remains the same as for the natural ordering but the overhead resulting from startup costs for the vector operations is nearly quadrupled. Since some of the vectors in the matrices in Figures 3.12 and 3.16 line up across blocks, if we store them contiguously we can save some of the vector startups in the matrix-vector multiplication.

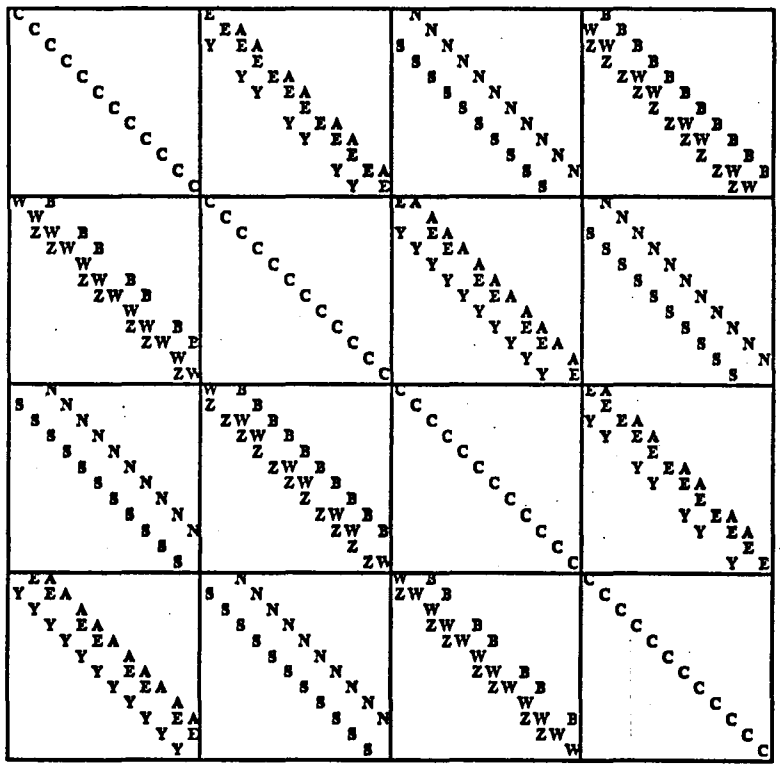
Note that it may be necessary to add some zero storage as in the case of the Y and Z vectors in blocks 1,3 and 2,4 of Figure 3.12 in order to treat the two vectors as one in the matrix-multiply. Since the offset of the Y and Z diagonal is $O(\frac{r}{4})$, this number of zeros must be added to the end of the first Y and Z diagonals and included in the vector multiply and add operations associated with the long Y and Z vectors in the matrix vector multiplication. Since the matrix is

symmetric, these long vectors are multiplied and added twice for a total of eight vector operations and eight associated startup costs. By storing the two Y vectors contiguously we save four startup costs or, put another way, we save the time it takes to do 400 adds or multiplies. The extra storage and calculations will become significant if $r/4 \approx 100$. This indicates that any savings from lining up vectors across blocks is marginal for large problems and may even be detrimental for very large problems. (i.e. $\approx 400 \times 400$) The main advantage of this strategy will probably come in three dimensional problems where relatively small numbers of grid points per row can still lead to very large matrices.

Returning to Figure 3.12, note that some of the vectors do not line up across blocks and so a natural question we are led to consider is under what conditions will the vectors lineup across blocks and can we order the grid so that all of the diagonals in blocks of the multi-color matrix line up with diagonals in appropriate blocks. We make the following conjecture.

If a class $\mathcal{R}$ problem with connectivity set Υ containing ν elements is ordered by the continuous coloring rule, then the minimum number of diagonals to store the matrix (provided appropriate N/p length diagonals are stored contiguously and zeros added where neccessary), will occur if p divides N evenly. Furthermore, the minimum number of diagonals necessary to store the matrix is ν.

This conjecture is illustrated in Figures 3.16 and 3.20. In Figure 3.20, $N = 48$ and the blocks in the 4-color matrix are all square. Note that if the diagonals are stored in correct sequence all of the B vectors in the lower triangular part of A can be stored as one vector. In like manner, the W, Z, N, and S diagonals below the main diagonal can be stored contiguously in memory, requiring 8 vectors. Of course, pointers must also be kept to allow the block structure for the forward and



.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
3	4	1	2	3	4
1	2	3	4	1	2
3	4	1	2	3	4
1	2	3	4	1	2

Mixed Derivative Problem - 6 × 8 Grid

4 Colors - r = 4i + 2

Figure 3.20

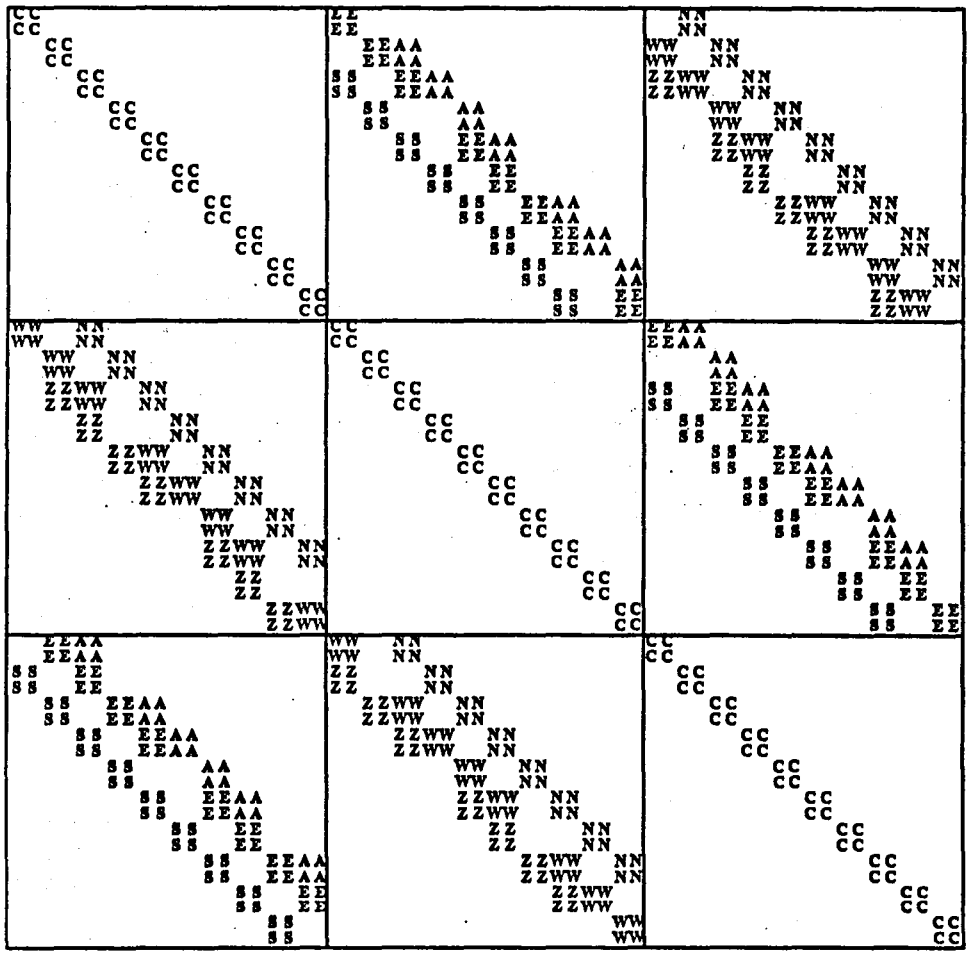
back solves. In Figure 3.16, however, $N = 42$ and the off diagonal blocks of the 4-color matrix are rectangular. This does not affect the line up of vectors within blocks but it does change the offsets of diagonals across blocks so that they no

longer all line up.

3.6. Examples of Continuous Multi-coloring for Multiple Unknowns

In section 3.4 we gave examples of different colorings using a single differential equation, in which case one equation of the discretized system is associated with each grid point. We turn now to the more complicated case where more than one unknown is to be calculated at each grid point. We consider again the plane stress problem described in section 3.1. The grid stencil shown in Figure 3.3 can be 3-colored but since the center point contains two unknowns the matrix in Figure 3.21 is not 3-colored. If we use diagonal storage of this matrix we will either have to store many zeros or use expensive gather-scatter type operations to perform the matrix vector multiplications and forward and back solves in the ICCG algorithm. Instead we also color the u and v unknowns at each point so that they decouple, that is, each grid point is associated with two colors. This is illustrated in Figure 3.22 where pairs of colors are associated with single gridpoints.

In Figure 3.22 the v unknowns alternate with the u unknowns in the ordering sequence. This is the continuous coloring rule for problems with more than one unknown per grid point. An alternative approach, shown in Figure 3.23, is to follow (3.1), and order all the u unknowns first and then the v unknowns. For both orderings vectors within blocks line up; however, the number of vectors which line up across adjacent blocks is not the same. In Figure 3.24 a comparison of storage requirements is given for the two orderings. Here we assume that wherever possible vectors that line up across blocks are stored contiguously. The additional storage noted in Figure 3.24 comes from storing vectors whose offsets within blocks are $O(r/p)$ as occurs, for example, in storing the N vectors beginning in block row 3 column 1 in Figure 3.22. It is important to note that some of the vectors in Figure 3.23 do not line up in general, even though they do for the small test case

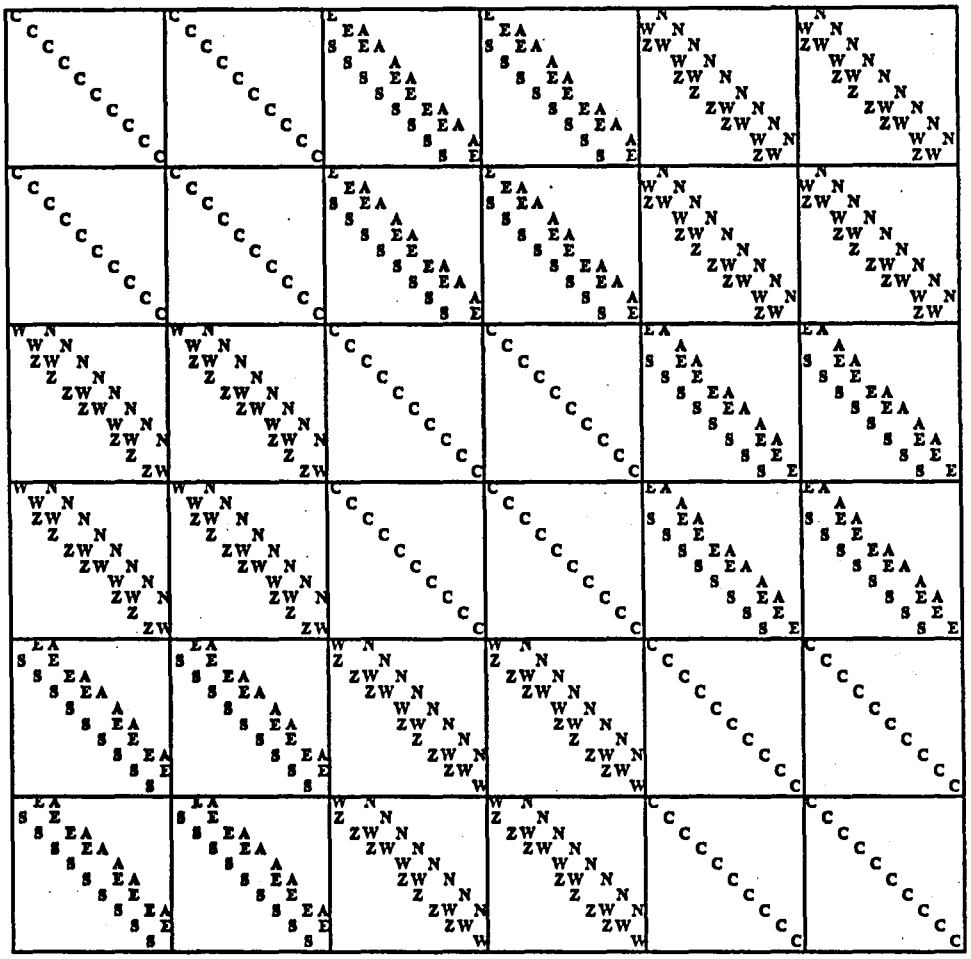


: :	: :	: :	: :	: :
1 1	2 2	3 3	1 1	2 2
2 2	3 3	1 1	2 2	3 3
3 3	1 1	2 2	3 3	1 1
1 1	2 2	3 3	1 1	2 2

3-Color Plane Stress Matrix

Figure 3.21

illustrated. For example, the Z diagonal in block 3,2 lines up with the W diagonal in block 4,3 but as the number of points on each row increases, the offset of the Z diagonal increases while the W diagonal is unchanged. In general, the offsets of A,N,S,and Z diagonals are a function of r while the E and W offsets are constants.



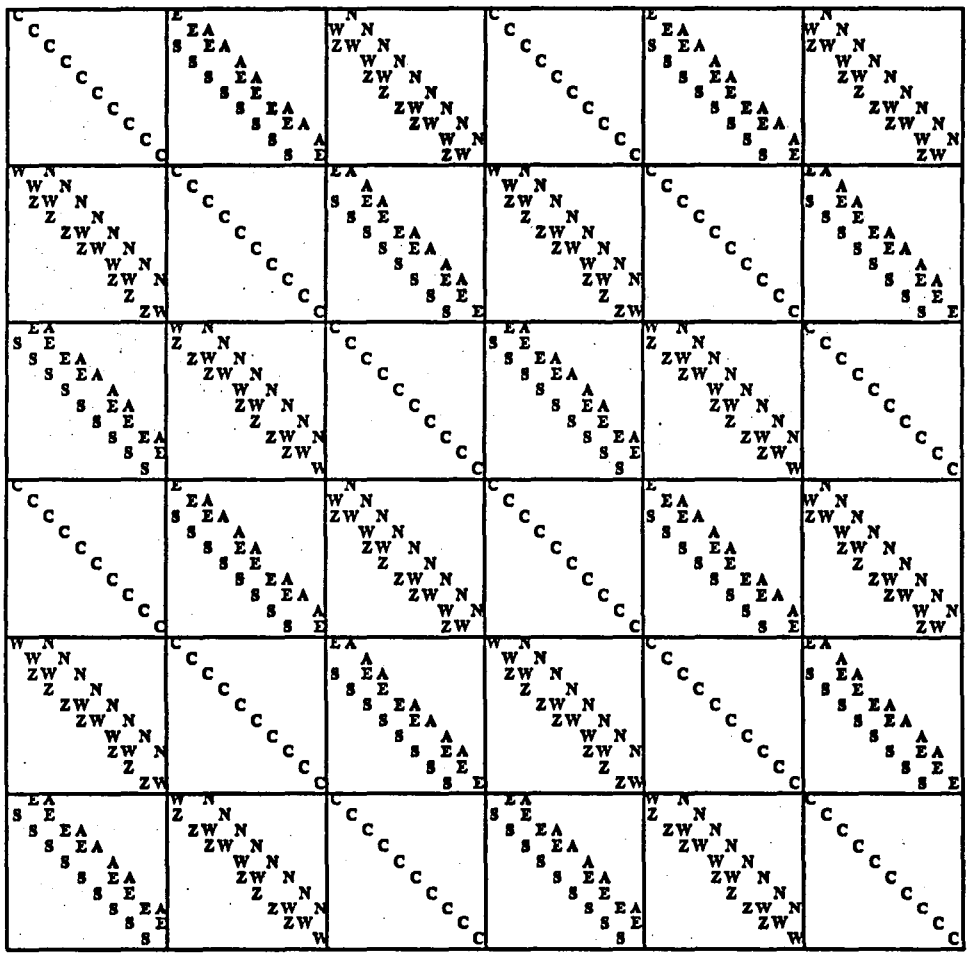
1 2	3 4	5 6	1 2	3 4
3 4	5 6	1 2	3 4	5 6
5 6	1 2	3 4	5 6	1 2
1 2	3 4	5 6	1 2	3 4

5 × 6 Grid - 3 × 2 Color Ordering

Plane Stress Matrix: u and v Alternate

Figure 3.22

For this problem the ordering in Figure 3.22, which follows the continuous coloring rule, is superior for the matrix vector multiplications required in the conjugate gradient iterations. For very large N the savings introduced by eliminating vector startups is overcome by the time added by introducing additional



1 4	2 5	3 6	1 4	2 5
2 5	3 6	1 4	2 5	3 6
3 6	1 4	2 5	3 6	1 4
1 4	2 5	3 6	1 4	2 5

5 × 6 Grid - 2 × 3 Color Ordering

Plane Stress Matrix: *u's* First

Figure 3.23

calculations in the matrix vector multiplication. This does not occur until the offsets of vectors which line up is equal to the number of operations which can be accomplished during a vector startup, ≈100, as discussed for the mixed derivative problem. We also note that the number of rows must be chosen to ensure square

Matrix Storage by Diagonals for Figures 3.20 and 3.21								
	# of Vectors of Length						*startups for matrix multiply*	*additional storage*
	$\frac{N}{p}$	$\frac{2N}{p}$	$\frac{3N}{p}$	$\frac{4N}{p}$	$\frac{5N}{p}$	N		
fig 3.20	10	3	2	3	1	1	(4)(19)+1 = 77	$12\frac{r}{p} - 5$
fig 3.21	12	12	1				(4)(25)+1 = 101	$8\frac{r}{p} - 4$

Diagonal Storage for Plane Stress Matrices

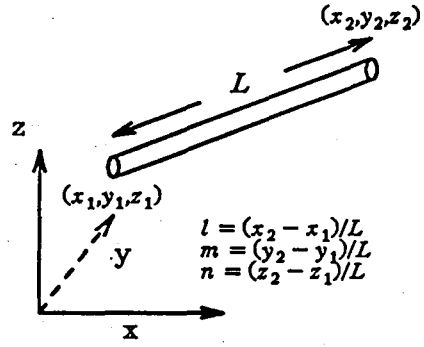
Figure 3.24

blocks through out A as in the second model problem in order to maximize the number of vectors which line up across blocks.

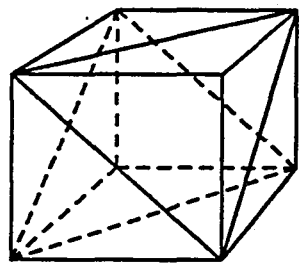
3.7. Multi-Coloring for Three Dimensional Problems

The real physical problems of interest in large scale scientific computing today typically are three dimensional, and we now discuss multi-coloring for such problems. Our model problem is a three dimensional space truss which is made up of cube-like structures which are assembled arbirtarily in the x,y, and z directions. (see Figure 3.25). The cube is diagonally braced on each face and made up of bar elements each of which has three unknowns at each end, the displacements u,v,and w in the x,y, and z directions. With each element is associated a 6 by 6 element matrix shown in Figure 3.25, which is assembled using finite element techniques into a stiffness matrix. The problem is to solve for the displacements resulting from an applied force with rigid body motions constrained.

The grid stencil for this problem is three dimensional and using 4 colors with $r = 4i + 1$ and $c = 4i + 2$ will decouple the center point, but since there are three unknowns at each node, 12 colors are needed to obtain a p-colored matrix. Theorem 3.3 allows us to easily obtain p-color matrices for this three dimensional problem.



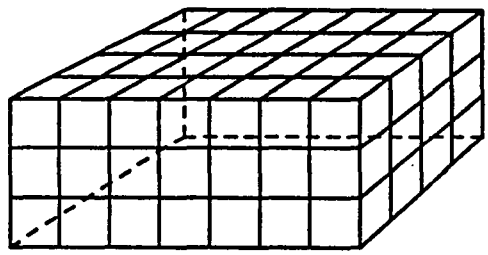
bar element



building block

$$\frac{AE}{L} \times \begin{vmatrix} l^2 & lm & ln & -l^2 & -lm & -ln \\ lm & m^2 & mn & -lm & -m^2 & -mn \\ ln & mn & n^2 & -ln & -mn & -n^2 \\ -l^2 & -lm & -ln & l^2 & lm & lm \\ -lm & -m^2 & -mn & lm & m^2 & mn \\ -ln & -mn & -n^2 & ln & mn & n^2 \end{vmatrix}$$

bar element matrix



3-D structure

Space Truss Model Problem

Figure 3.25

We color the three dimensional grid using both the continuous coloring rule and the natural ordering with (3.1) to assign the p colors. The two different orderings are illustrated in Figure 3.26. In a) we have used the continuous coloring rule; note that the last unknown in the bottom plane is color 6 and so the next plane begins with color 7. If different dimensions are required than those given above for r and c, the techniques discussed earlier, such as increasing the number of colors or adding

10,11,12	1,2,3	4,5,6	7,8,9	10,11,12
7,8,9	10,11,12	1,2,3	4,5,6	7,8,9
4,5,6	7,8,9	10,11,12	1,2,3	4,5,6
1,2,3	4,5,6	7,8,9	10,11,12	1,2,3
10,11,12	1,2,3	4,5,6	7,8,9	10,11,12
7,8,9	10,11,12	1,2,3	4,5,6	7,8,9

next plane

4,5,6	7,8,9	10,11,12	1,2,3	4,5,6
1,2,3	4,5,6	7,8,9	10,11,12	1,2,3
10,11,12	1,2,3	4,5,6	7,8,9	10,11,12
7,8,9	10,11,12	1,2,3	4,5,6	7,8,9
4,5,6	7,8,9	10,11,12	1,2,3	4,5,6
1,2,3	4,5,6	7,8,9	10,11,12	1,2,3

bottom plane

a) Truss Matrix #1

4,8,12	1,5,9	2,6,10	3,7,11	4,8,12
3,7,11	4,8,12	1,5,9	2,6,10	3,7,11
2,6,10	3,7,11	4,8,12	1,5,9	2,6,10
1,5,9	2,6,10	3,7,11	4,8,12	1,5,9
4,8,12	1,5,9	2,6,10	3,7,11	4,8,12
3,7,11	4,8,12	1,5,9	2,6,10	3,7,11

next plane

2,6,10	3,7,11	4,8,12	1,5,9	2,6,10
1,5,9	2,6,10	3,7,11	4,8,12	1,5,9
4,8,12	1,5,9	2,6,10	3,7,11	4,8,12
3,7,11	4,8,12	1,5,9	2,6,10	3,7,11
2,6,10	3,7,11	4,8,12	1,5,9	2,6,10
1,5,9	2,6,10	3,7,11	4,8,12	1,5,9

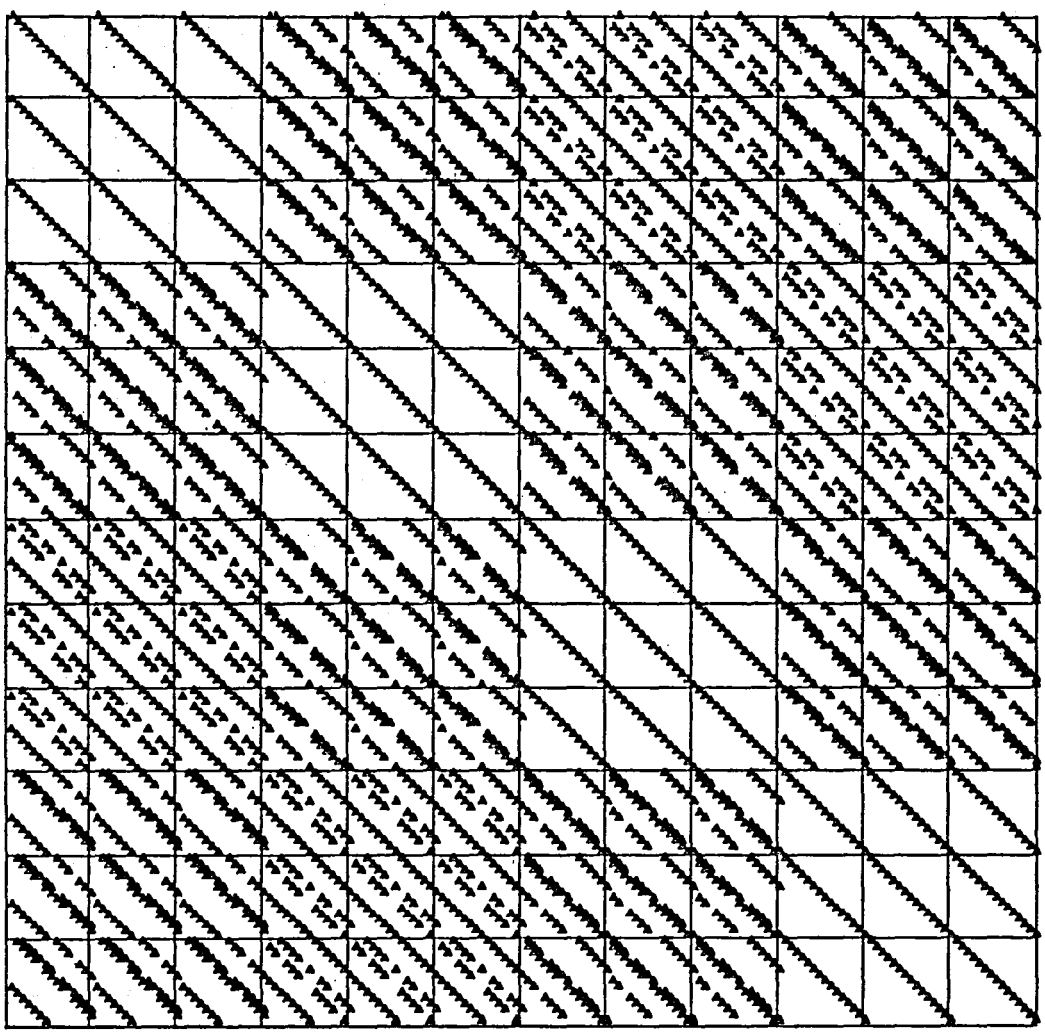
bottom plane

b) Truss Matrix #2

Grid Colorings for 3-D Space Truss

Figure 3.26

'dummy rows', can be used. In Figure 3.26b we have used (3.1) and ordered all u unknowns first, followed by the v's and then the w's. We still color the u unknowns continuously throughout the grid, as well as the v and w unknowns. For the space truss problem the element matrices themselves are often sparse and so the non-zero structure indicated by the grid stencil is not representative of the actual structure of the assembled matrix. Figure 3.27 shows the matrix structure for the ordering (3.2) if the element matrices are full but Figure 3.28 shows the actual non-zero structure for a $5 \times 6 \times 2$ node model oriented along the x,y and z

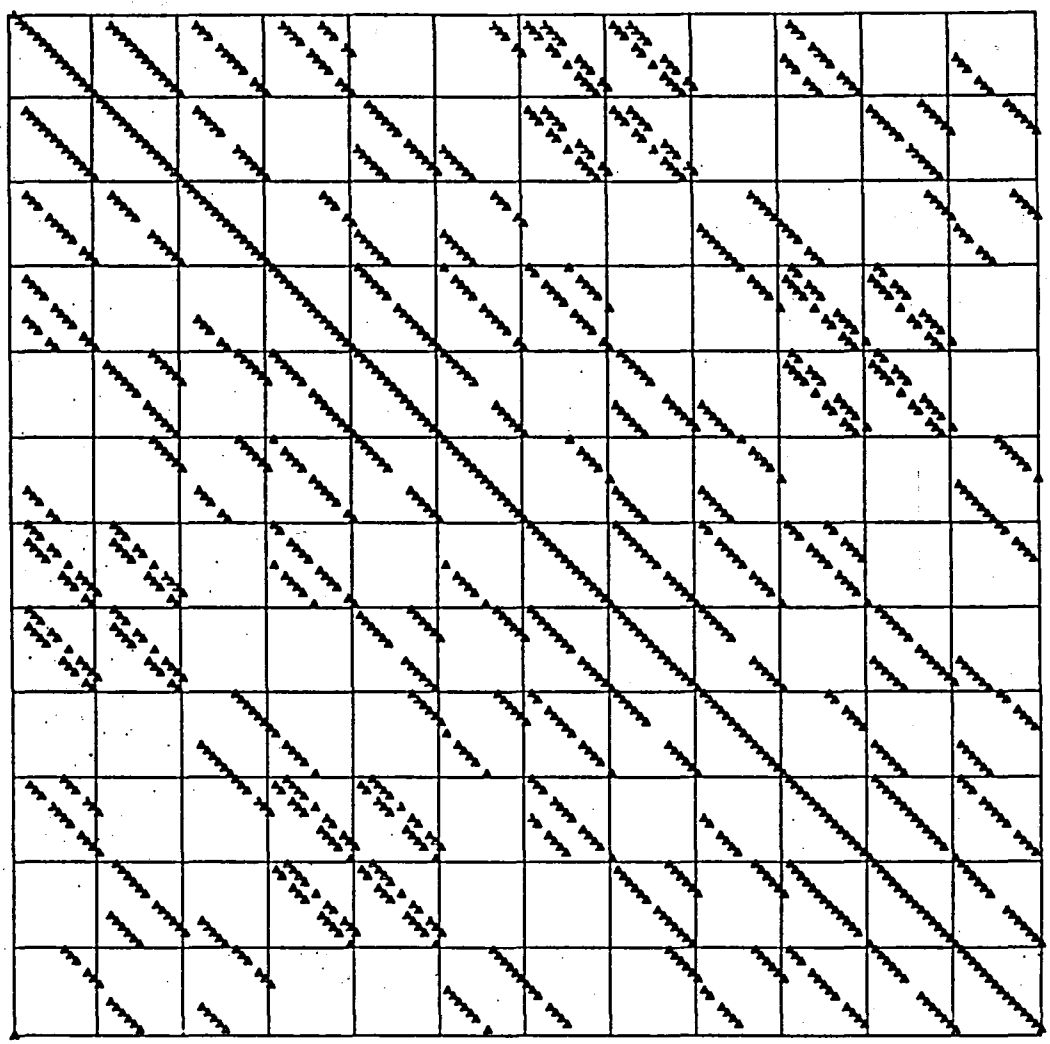


u, v, and w Alternating (3.2) - 3×4 Coloring

Truss Matrix #1 According to Grid Stencil

Figure 3.27

axes. Figures 3.29 and 3.30 show the same results for the ordering (3.1). This ordering, while not as good for the plane stress problem, appears better for the space truss problem. We see that the grid stencil for a problem does not always predict the actual non-zero structure. However, it can serve to bound the amount of storage needed for the diagonal storage scheme.

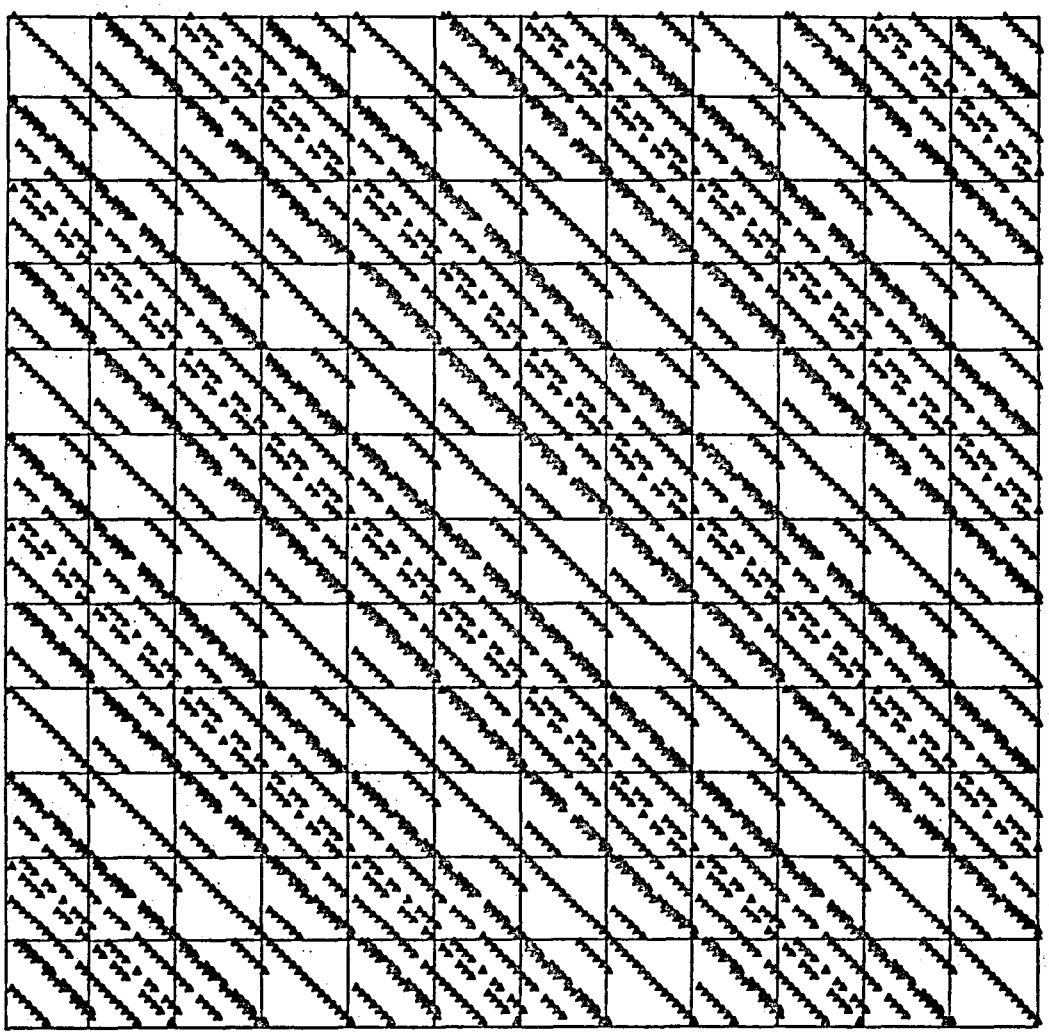


u, v, and w Alternating (3.2) - 3 × 4 Coloring

Assembled Truss Matrix #1

Figure 3.28

There remains the problem of assembling the stiffness matrix for the plane stress and the space truss problems. We present in Appendix B a row by row assembly process that forms the stiffness matrix with storage by diagonals, and with the appropriate data structures to access the diagonals. For the three dimensional coloring problem, obtaining $O(N/p)$ vector lengths will require that the number of

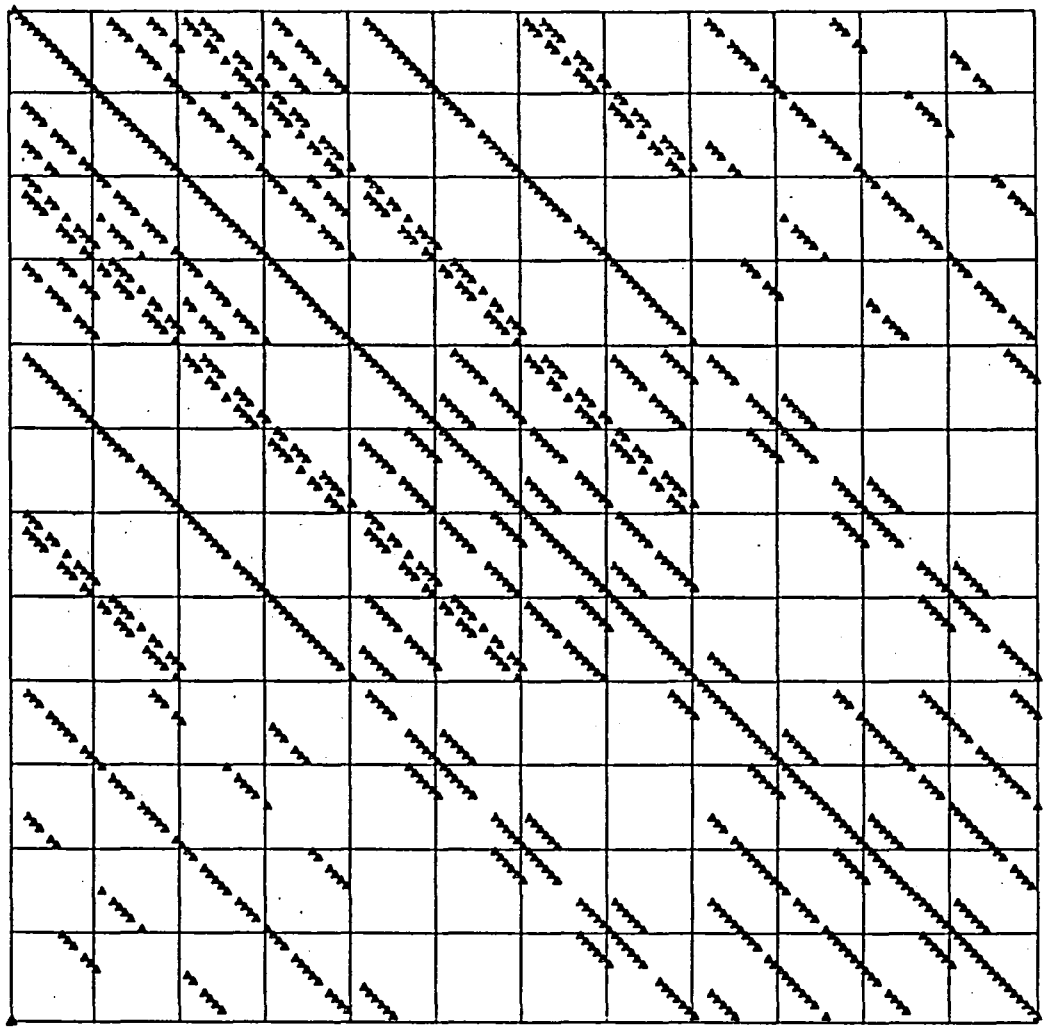


u, v, w Consecutively (3.1) - 4×3 Coloring

Truss Matrix #2 according to Grid Stencil

Figure 3.29

grid points on a row be some multiple of the number of colors divided by the number of unknowns per grid point plus a constant ($4i + j$, $1 \leqslant j \leqslant 3$ for the truss problem) and that the number of rows in a plane also be chosen so that the coloring pattern can continue onto the next plane without going out of sequence.



u, v, w Consecutively (3.1) - 4 × 3 Coloring

Assembled Truss Matrix #2

Figure 3.30

3.8. Summary

In this chapter we have discussed multi-coloring techniques used to vectorize the ICCG method. We first defined a class of problems for which the multi-coloring techniques apply. Then we discussed the natural ordering, proving in Theorem 3.1 that the matrices corresponding to natural orderings minimize the storage of the

matrix by diagonals and allow long vector operations for the matrix-vector multiplication operation. However to vectorize ICCG we desired p-color matrices and sought to extend the results of Theorem 3.1 to multi-color orderings. We gave necessary and sufficient conditions that determine if the matrix corresponding to a p-color ordering is a p-color matrix in Theorem 3.2, and also gave conditions which must be satisfied to minimize diagonal storage for general p-color matrices.

We saw that Theorem 3.2 did not address the important question of how one obtains suitable multi-color orderings given a domain and a grid stencil and so we described a procedure that is easily applied to 2 or 3 dimensional problems which we called the *continuous coloring rule*. In Theorem 3.3 we gave necessary and sufficient conditions to obtain p-color matrices using the continuous coloring rule in terms of a constraint which is easily tested given a connectivity set associated with the given problem. We concluded by discussing examples from the model problems showing how to apply Theorem 3.3 to each problem and by discussing 'super'-length vectors obtained by lining up vectors across adjacent blocks of p-color matrices whenever possible. We saw that if N divides p evenly the maximum lineup of vectors occured but that the savings in execution time is minimal.

CHAPTER 4

A Performance Model and Numerical Results

In this chapter we present a model which is used to predict performance of both the standard multi-color ICCG, ICCGC, and the Eisenstat-like implementation, ICCGE. We compare the performance of these multi-color methods to several other methods including conjugate gradient, CG, without preconditioning, conjugate gradient with Jacobi preconditioning, JCG, where the matrix is scaled so that the main diagonal is the identity, and other ICCG methods such as ICCGN, natural order incomplete Cholesky and ICCGD, the diagonal ordered incomplete Cholesky discussed in chapters 2 and 3.. We discuss the performance of the multi-color methods applied to the four model problems presented in previous chapters and note the effects of multi-color orderings on the convergence rate and execution time compared to the natural ordering. Timing results obtained from runs on the CYBER 205 at NASA's Langley Research Center are compared with results predicted by the performance model. Finally, we summarize the performance results and compare the effectiveness of the multi-color ICCG methods to other ICCG methods used on vector computers.

4.1. A Performance Model

In this section we first discuss a general performance model which is used to compare two different ICCG methods or to compare an ICCG method to conjugate gradient without preconditioning. We then give formulas that predict run times for each model problem, based on the number of iterations for convergence and the problem size. Two factors used in comparing ICCG methods are rate of convergence,

usually measured by the number of iterations required to satisfy the given convergence criteria, and the amount of time required for the preconditioning step. For method i we denote the number of iterations for convergence by I_i and the execution time per iteration by T_i. The total execution time for method i is then $T_i I_i$. To compare method i and method j we define the following terms: $I_{i,j} = I_i / I_j$, $\tau_{i,j} = T_i / T_j$, and $S_{i,j} = 1/(I_{i,j}\tau_{i,j})$. If method i is a preconditioned conjugate gradient method, then $I_{i,CG}$ is a measure of the effectiveness of the preconditioning in reducing the iterations, $\tau_{i,CG}$ is a measure of the increased time for each iteration due to preconditioning, and $S_{i,CG}$ is a measure of the performance of method i compared to conjugate gradient in terms of total execution time. Generally, we expect method i to decrease the number of iterations compared to conjugate gradient so $I_{i,CG} < 1$ while the time per iteration for method i will increase so that $\tau_{i,CG} > 1$. If the product $I_{i,CG}\,\tau_{i,CG}$ is less than one, an overall speedup in total execution time, given by $S_{i,CG}$, will occur. We can also use the factors I, τ, and S to compare two different ICCG methods in which case method i will be superior to method j if $S_{i,j} > 1$. Clearly the size of τ will be greatly influenced by the degree of vectorization possible in the preconditioning calculations.

To describe our model we consider the computations involved for each iteration of the ICCG algorithm given in Figure 1.1. These computations can be divided into three parts: the matrix vector product in step a), A, the solution of $M\hat{\mathbf{r}}^{k+1} = \mathbf{r}^{k+1}$ in e), M, and the remaining computations in a) through g), C. That is, the third part includes three linked triads and two inner products plus the convergence test. The convergence test we use calculates the 2-norm squared of the residual, $\mathbf{r}$, at each iteration and so an extra inner product is required except for the conjugate gradient method itself. We will refer to the time required for a matrix-vector multiply, A, as A_t, and likewise for M and C.

The solution of $M\hat{\mathbf{r}}^{k+1} = \mathbf{r}^{k+1}$ requires a forward and back solve and a diagonal scaling; as discussed in chapter 2, the amount of computation is the same as for a matrix-vector multiply. The execution time however may be drastically different. The matrix vector multiply is carried out using vector adds and multiplies when the matrix is stored by diagonals. For a 2-pipe CYBER 205 using 64 bit arithmetic the maximum rate for vector adds and multiplies is $\approx$ 100 Mflops. The forward and back solves in step e) are carried out in the same fashion if multi-color orderings are used, as described in chapters 2 and 3, but for the natural ordering the process is essentially scalar.

As a measure of the degree of vectorization of the preconditioner we define $\alpha = M_t / A_t$, the ratio of the time for carrying out the preconditioning to the time required for the matrix vector multiply. Note that $\alpha = 1$ on a scalar computer since the number of operations for A and M are the same. If the preconditioning step vectorizes as well as the matrix multiply, we will also have α nearly equal to one. For multi-color methods we expect $\alpha \approx 1$ while for the natural ordering $\alpha \approx 10$ or higher might be expected. We also define $\beta = C_t / A_t$, the ratio of execution time per iteration for the linked triads and innerproducts to the time for a matrix vector multiply. The execution time for one ICCG iteration can now be expressed in terms of A_t as

$$T = (1 + \alpha + \beta) A_t \tag{4.1}$$

To compare execution time per iteration for ICCG method i to method j, we compute

$$\tau_{i,j} = \frac{T_i}{T_j} = \frac{1 + \alpha_i + \beta_i}{1 + \alpha_j + \beta_j} \tag{4.2}$$

Finally, to compare the total execution time for convergence of method i with method j we have

$$S_{i,j} = \frac{1}{I_{i,j}\,\tau_{i,j}} = \frac{I_j}{I_i} \times \frac{1+\alpha_j+\beta_j}{1+\alpha_i+\beta_i} \tag{4.3}$$

Equation (4.3) serves to define the relationship between rate of convergence and computation time. For example, let us assume that $A_t >> C_t$ in the conjugate gradient iteration so that β may be neglected, and let us compare ICCGN with ICCGC. If $\alpha_{ICCGN} = 10$ and $\alpha_{ICCGC} = 1$, we have

$$S_{ICCGN,ICCGC} = \frac{I_{ICCGC}}{I_{ICCGN}} \times \frac{2}{11}$$

In other words, the natural order method must take 5.5 times fewer iterations than the multi-color method in order to achieve the same execution time as ICCGC. If compared to conjugate gradient, the ICCGN method must reduce the number of iterations by a factor of 11 just to 'break even' in terms of execution time while for ICCGC the 'break even' point is to reduce the number of iterations by a factor of 2.

We now derive timing formulas which allow us to predict performance of the multi-color ICCG methods and to compute estimates for the ratios $\tau_{i,j}$ above. In developing these formulas all scalar arithmetic is ignored. Vector assignment instructions (e.g. $\mathbf{a}(1{:}L) = \mathbf{b}(1{:}L)$) are not counted as operations but are included in the timing formulas. Execution times for vector instructions on the CYBER 205 are of the form $s + \gamma L$ where s represents the fixed startup cost independent of the vector length, γ is the incremental cost and L is the length of the vector. A summary of the timing assumptions used for vector instructions is given in Figure 4.1. Using these vector timings, we derive timing formulas for the matrix vector multiply, A, the preconditioning step, M, and the remaining computations, C, for one iteration. The predicted times are in nanoseconds (ns). We also give formulas for the number of adds and/or multiplies which we denote as *operations* in the tables that follow. For simplicity, we do not use the lining up of vectors across adjacent blocks as discussed in section 3.6 in the formulas that follow. Throughout the

Cyber 205 2-Pipe 64 Bit Arithmetic		
vector instruction	time (ns) $s + \gamma L$	operations
linked triad	1660 + 10L	2L
vector add	1020 + 10L	L
vector multiply	1040 + 10L	L
inner product	2320 + 20L	2L -1
vector assignment	800 + 10L	-

Vector Instruction Timings

Figure 4.1

tables, the grid size is $r \times c$ where r is the number of points in a row and c is the number of rows in the rectangular grid. For three dimensional problems, l is the number of planes. Figure 4.2 gives the formulas used for counting the number of multiplies and adds per iteration for CG and ICCGC for the four model problems. For the standard multi-color method, ICCGC, the operation counts for A and the M step are the same as shown in Figure 4.2.

Model Problem	A, M (ICCGC) operations/iter	C (triads,inner products) operations/iter	total (ICCGC) operations/iter
Laplace	$9rc-4r-4$	$12rc-3$	$21-8r-11$
Mxdir	$17rc-12r-8$	$12rc-3$	$46rc-24r-19$
Plane Stress	$46rc-24r-16$	$24rc-3$	$116rc-48r-35$
Space Truss	$75rcl-36rc-36rcl-24$	$36rc-3$	$186rcl-72rc-72r-51$

Number of Operations per Iteration

Figure 4.2

Model Problem	A_t time/iteration (ns)	C_t (triads,inner products) time/iteration (ns)	M_t (ICCGC) time/iteration (ns)
Laplace	$90rc - 40r + 17480$	$90rc + 11940$	$100rc - 40r + 18280$
Mxdir	$170rc - 120r + 66880$	$90rc + 11940$	$180rc - 120r + 67680$
Plane Stress	$460rc - 240r + 136840$	$180rc + 11940$	$480 - 240r + 137640$
Space Truss	$750rcl - 360rc - 360r + 297440$	$270rcl + 11940$	$780rcl - 360rc - 360r + 298240$

Predicted Execution Time per Iteration

Figure 4.3

Figure 4.3 gives the timing formulas for the 3 parts we have defined in our computational model used for the CG, JCG, and ICCGC methods. The general form used to predict the time for each part of the ICCG algorithm is

$$time\ per\ iteration = startup + 10 \times number\ of\ operations$$

For each method, the predicted time per iteration, T, is given by

$$T = A_t + C_t + M_t$$

where $M_t = 0$ for CG and JCG. The startup cost depends on the number and type of vector instructions in each part. Note that the constant term in each timing formula comes from the vector startup times as well as from any constants in the expressions for the number of operations. For example, the code used to implement the matrix vector multiply using diagonal storage of A requires an initial vector multiply of length N and then for each diagonal stored two vector multiplies and adds are executed since each diagonal stored represents a diagonal above and below the main diagonal of the symmetric matrix A. The number of diagonals stored for the Laplace problem is 4 plus the main diagonal so using the timing information in Figure 4.1 the startup overhead is

$$1040 + 4 \times 2 \times (1040 + 1020) = 17{,}520\,ns$$

The formula given for the time A_t in Figure 4.3 includes this startup overhead plus

ten times the number of operations given in Figure 4.2. The time M_t includes the time for a vector assignment statement which is $800+10rc$ and is reflected in Figure 4.3 since the timing formulas for M_t differ from those given for A_t by this amount. In a similar fashion the formulas for the number of operations and the execution time per iteration for C are derived. The three dot products and three linked triads require a total of six vector instructions but since linked triad instructions accomplish two operations per clock cycle, the number of operations per instruction is double that for vector adds and multiplies. Thus the leading term for C_t in Figure 4.3 is $90rc$ instead of $120rc$.

For the Eisenstat implementation, ICCGE, there is no explicit matrix-vector multiplication. It is carried out by doing a forward and back solve and so for the purpose of applying the performance model to this implementation, the operations equivalent to a matrix-vector multiply are counted as A, the three linked triads and inner products are counted in calculating C and β_{ICCGE}, and everything else is counted as preconditioning, M, and used to calculate α_{ICCGE}. Figure 4.4 gives formulas for the number of operations and execution times per iteration for the M portion of the ICCGE method. A_t and C_t are the same as in ICCGC so that M

Model Problem	M (ICCGE) Operations/Iter	M_t (ICCGE) Time/Iter (ns)
Laplac	$2rc-2$	$30rc+2840$
Mxdir	$11rc-6r-4$	$120rc-60r+36800$
Plane Stress	$35.3rc-16r-12$	$373.3rc-160r+94420$
Space Truss	$69rcl-32rc-32r-164$	$720rcl-320rc-320r+250920$

ICCGE Execution Time per Iteration

Figure 4.4

represents the remaining calculations in the ICCGE method and can be used to compare the cost of preconditioning for both methods. Also by arranging the calculations in this manner for the model, the term M_t reflects the additional computation for the ICCGE method compared to CG. Since the residual in the ICCGE implementation is not the residual in the original variables, the convergence test for this implementatiion is not identical to the test used for the other methods. We compute the 2-norm of $\hat{\mathbf{r}}$ for the ICCGE implementation and our numerical experiments gave nearly identical convergence results as using the 2-norm of $\mathbf{r}$ in the ICCGC algorithm. For the Laplace problem the ICCGE implementation is represented in a different form from the other problems. Recall from chapter two that for 2-colors the ICCGE implementation requires no preconditioning step if the matrix is scaled properly. Therefore, in Figure 2.3 $\mathbf{q} = \hat{\mathbf{r}}$ and we use the inner product calculated in step f) for the convergence test. The cost for the work at each iteration of ICCGE for the Laplace problem in addition to A and C is given in Figure 4.4. It includes one vector assignment and two vector adds which are not part of the standard CG implementation and so are counted as part of M in the performance model. Though not shown, the C terms for the ICCGE method for the Laplace problem contain only two dot products.

Using the formulas in Figures 4.3 and 4.4, we can estimate the parameters $\alpha = M_t / A_t$ and $\beta = C_t / A_t$ used in equations (4.2) and (4.3). Then, given the number of iterations for convergence for two methods to be compared, we can compute the ratio $S_{i,j}$ which predicts that method i is faster than method j if S is greater than one. For large problems the dominant term in each formula in Figure 4.3 is the leading term so we estimate the parameters using only the coefficients of the rc terms. In the next section we will compare these estimates with values based on actual run times. Figure 4.5 lists the parameters calculated from the formulas in Figures 4.3 and 4.4. We note that as the problems increase in complexity from

Model Problem	Algorithm	α	β	T/A_t	$\tau_{i,CG}$
Laplace	CG	0	1.0	2.0	1.0
	ICCGC	1.11	1.0	3.11	1.56
	ICCGE	.33	.78	2.11	1.06
Mxdir	CG	0	.53	1.53	1.0
	ICCGC	1.06	.53	2.59	1.69
	ICCGE	.71	.53	2.24	1.46
Plane Stress	CG	0	.39	1.39	1.0
	ICCGC	1.04	.39	2.43	1.75
	ICCGE	.812	.39	2.20	1.58
Space Truss	CG	0	.36	1.36	1.0
	ICCGC	1.04	.36	2.40	1.76
	ICCGE	.96	.36	2.32	1.70

$\alpha = M_t/A_t$ = ratio of preconditioning to matrix multiply time
$\beta = C_t/A_t$ = ratio of basic CG computations to matrix multiply time
T/A_t = total time per iteration in terms of matrix multiply time
$\tau_{i,CG}$ = ratio of method i time to CG time

Performance Model Calculations

Figure 4.5

the simple five diagonal matrix for the Laplace problem to the 145 diagonal matrix for the Space Truss, β decreases. This is expected since the time for the matrix vector multiply and preconditioning step increasingly dominates. The important effect of this change is seen in the increase of $\tau_{i,CG}$, which is the ratio of execution time per iteration for method i to the execution time per iteration for CG. Using (4.1) we have

$$\tau_{i,CG} = \frac{(1+\alpha+\beta)}{(1+\beta)}$$

If $\beta >> \alpha$, then $\tau_{i,CG} \approx 1$ while if $\alpha >> \beta$, then $\tau_{i,CG} \approx 1/(1+\alpha)$. For a fixed α, the cost of preconditioning is greater when β is small compared to α and the value of τ will increase as β decreases relative to α. The larger $\tau_{i,CG}$ the greater is the requirement that the preconditioned method i reduce the number of iterations in order to achieve a reasonable speedup. Another effect noted from Figure 4.5 is that the ICCGE method is the best performer in terms of execution time but less so as the problem complexity increases. This is because the number of colors used for the

Model Problem	Algorithm	Total Number of Operations
Laplace	CG,JCG	$(21I)rc-(4I)r-7I-1$
	ICCGC	$(30I)rc-(8I)r-11I-1$
	ICCGE	$(21I+4)rc-(4I+2)r-6I-3$
Mxdir	CG,JCG	$(29I)rc-(12I)r-11I-1$
	ICCGC	$(46I)rc-(24I)r-19I-1$
	ICCGE	$(40I+8)rc-(18I+6)r-15I-5$
Plane Stress	CG,JCG	$(70I)rc-(24I)r-19I-1$
	ICCGC	$(116I)rc-(48I)r-35I-1$
	ICCGE	$(105.3+22)rc-(40I+12)r-31I-9$
Space Truss	CG,JCG	$(102I)rcl-(36I)rc-(36I)r-27I-1$
	ICCGC	$(177I)rcl-(72I)rc-(72I)r-51I-1$
	ICCGE	$(171I+36)rcl-(68I+18)rc-(68I+18)r-43I-11$

Total Operations for I Iterations

Figure 4.6

four problems is increasing so that the savings realized by the Kt multiplication described in chapter 2 is diminished.

Finally, Figures 4.6 and 4.7 give formulas for total operation counts and for predicting total execution time for each problem in terms of the problem size (r, c, l) and the number of iterations, I. These formulas are obtained by combining the appropriate terms in Figures 4.2, 4.3 and 4.4 and adding appropriate terms for initialization before the first iteration. We have factored each formula into terms that depend on the problem size. The general formula used for all of the methods

Model Problem	Algorithm	Predicted Run Time (ns)
Laplace	CG,JCG	$(180I+30)rc-(40I)r+29420I+2260$
	ICCGC	$(280I+30)rc-(80I)r+47700I+2260$
	ICCGE	$(190I+70)rc-(40I+20)r+29940I+10480$
Mxdir	CG,JCG	$(260I+30)rc-(120I)r+78820I+2260$
	ICCGC	$(440I+30)rc-(240I)r+146500I+2260$
	ICCGE	$(380I+110)rc-(180I+60)r+115620I+18740$
Plane Stress	CG,JCG	$(640I+60)rc-(240I)r+148780+2260$
	ICCGC	$(1120I+60)rc-(480I)r+286420I+2260$
	ICCGE	$(1013.3I+250)rc-(400I+120)r+243200I+35140$
Space Truss	CG,JCG	$(1020I+90)rcl-(360I)rc-(360I)r+308580I+2260$
	ICCGC	$(1800I+90)rcl-(720I)rc-(720I)r+607620I+2260$
	ICCGE	$(1720I+390)rcl-(680I)rc-(680I)r+560300I+125540$

Predicted Run Times for I Iterations

Figure 4.7

is

$$total\ time = startup\ time + iterations \times (A_t + C_t + M_t)$$

These predicted times will be compared with actual run times on the CYBER 205 in the next section.

4.2. Multi-color ICCG Performance

We now turn to actual performance results for multi-color ICCG algorithms for the four model problems. Results are given for the standard multi-color implementation, ICCGC, as well as for the ICCGE implementation. Performance results are also given for the conjugate gradient methods, CG and JCG, so that speedups can be calculated for the ICCGC and ICCGE algorithms. Timing results are compared with the predicted run times given in Figure 4.7 and the quantity $T_{i,CG}$ from Figure 4.5 is used with the actual iteration counts to predict the reduction in run time.

Results from CYBER 205 runs are given in Figure 4.8 for the four model problems. For each problem the problem size is given in terms of the grid size and number of unknowns. For example, the Laplace results given are for a 97 by 97 grid or 9409 unknowns. For the plane stress problem there are two unknowns at each grid point so for an 80 by 81 grid there are $80 \times 81 \times 2 = 12960$ unknowns. Likewise, the three dimensional grid is given for the space truss problem where there are three unknowns at each grid point. The four problems require 2,4,6, and 12 colors, respectively, to achieve a p-color matrix as described in chapter 3 following the continuous coloring rule. The predicted times from the performance model of the previous section are given in Figure 4.8 in parentheses after the actual measured runtimes.

For large problems, a good estimate of the maximum computation rates in Mflops (10^6 operations per second) is given by dividing the leading terms in Figure

Model Problem	Algorithm	Iterations	Time (sec)	Operations	Mflops
Laplace	CG	266	.495 (.486)	55,747,215	113
	JCG	259	.482 (.473)	54,280,183	107
99×130	ICCGC	130	.376 (.369)	38,891,709	103
9999	ICCGE	131	.256 (.253)	27,494,382	107
Mxdir	CG	230	.700 (.688)	74,649,029	107
	JCG	230	.700 (.688)	74,649,029	107
106×106	ICCGC	87	.453 (.441)	44,743,490	99
11236	ICCGE	88	.394 (.386)	39,470,743	100
Plane	CG	794	3.67 (3.40)	358,618,833	98
Stress	JCG	766	3.54 (3.38)	345,972,325	98
80×81×2	ICCGC	298	2.38 (2.24)	222,845,889	94
12960	ICCGE	298	2.18 (2.02)	202,581,633	93
Space	CG	2506	9.83 (9.45)	960,732,737	98
Truss	JCG	1989	7.80 (7.50)	762,528,896	98
45×46×2×3	ICCGC	784	5.40 (5.13)	484,283,855	90
12,420	ICCGE	783	5.18 (4.95)	470,957,828	91

Experimental Results from 2 Pipeline CYBER 205

Figure 4.8

4.2 by the corresponding leading terms in Figure 4.3 and multiplying by 1000. For the Laplace problem the estimated maximum rates for A, C, and M are 100 Mflops, 133 Mflops and 90 Mflops respectively. Using Figures 4.6 and 4.7 the predicted maximum computation rate for the Laplace problem is 117 Mflops. The predicted times in Figure 4.8 are uniformly less than the actual run times, as are the Mflop

rates computed from the data in Figure 4.8. This is expected since all scalar arithmetic and overhead from subroutine calls, etc. is ignored. Since the largest amount of computation for each method consists of vector adds and multiplies, we expect that computation rates of approximately 100 Mflops would be achieved if the method vectorizes well. The computation rates given in Figure 4.8 show that the multi-color ICCG methods do achieve this degree of vectorization. Some rates over 100 Mflops are given for the Laplace problem and for the mixed derivative problem, reflecting the effect of the linked triad instructions in the conjugate gradient iterations.

Figure 4.9 shows how accurately the performance model described in the previous section can predict the actual speedup in run times when the iteration

Model Problem	Algorithm	$I_{i,CG}$	predicted $\tau_{i,CG}$	predicted $S_{i,CG}$	actual $S_{i,CG}$
Laplace	ICCGC	.49	1.65	1.31	1.31
	ICCGE	.49	1.06	1.93	1.93
Mxdir	ICCGC	.395	1.69	1.50	1.55
	ICCGE	.395	1.46	1.73	1.78
Plane Stress	ICCGC	.375	1.75	1.53	1.54
	ICCGE	.375	1.58	1.68	1.68
Space Truss	ICCGC	.313	1.76	1.81	1.82
	ICCGE	.312	1.69	1.88	1.89

$I_{i,CG}$ = ratio of iterations for method i to CG
$\tau_{i,CG}$ = ratio of predicted time per iteration for method i to CG
$S_{i,CG}$ = Speedup in terms of execution time for method i compared to CG

Comparison of Results to Model Predictions

Figure 4.9

counts are given for the two methods being compared. From Figure 4.8 we compute $I_{i,CG}$ and using the estimate $\tau_{i,CG}$ from Figure 4.5, the predicted speedup for method i is given by $S_{i,CG}$. Using the runtimes given in Figure 4.8, we calculate the actual speedup for method i given in the last column of Figure 4.9.

4.3. Comparison to Natural Order ICCG Algorithms

In this section we attempt to compare the performance of multi-color ICCG to other ICCG methods which have been used on vector computers. A major concern with using multi-color orderings is the effect on the rate of convergence of the ICCG algorithm. We will address this concern by summarizing published results for convergence of ICCG methods based on the natural ordering, ICCGN, and by presenting our results for ICCGN for three of the four model problems. We also model an ICCG algorithm based on a diagonal ordering of the unknowns, ICCGD, for the Laplace problem and predict its performance on the CYBER 205.

Several problems are encountered in comparing our results with published results. Although Laplace's equation is the standard model used to test the various algorithms, the problems chosen are not always identical. Convergence criteria are not the same for each problem and sometimes are not even given. Finally, the ICCG method is not always compared to conjugate gradient so that it is impossible to tell how much the given method reduces the execution time for the particular problem.

A comparison of the number of iterations for convergence for several preconditioned conjugate gradient algorithms is given by Jackson and Robinson [1981] including the standard no-fill ICCG method, partial fill ICCG methods and MICCG methods using the column sum constraint in calculating the incomplete factors. No vector computer timings are given but the iteration counts are useful in comparing convergence rates of the various ICCG methods. Their results indicate lower

iteration counts for partial fill ICCG methods compared to standard ICCG and even better results for the MICCG methods. For a Laplace problem with $30 \times 30 = 900$ unknowns the iteration counts ranged from 28 for standard ICCG to 10 for a block MICCG method which is based on the block tridiagonal structure of the Laplace matrix and uses the column sum constraint in approximating the inverse of the $r \times r$ diagonal blocks by diagonal matrices. All of the methods they present are based on the natural ordering and do not vectorize well with the exception of the block incomplete method mentioned above which would vectorize with $O(r)$ length vectors.

Schrieber and Tang [1982] give a few convergence results for the multi-color ICCG method for Laplace's equation using the red-black ordering and a 4-color ordering. For a problem with 2500 unknowns the standard ICCG method based on the natural ordering required 25 iterations to converge. The red-black ordering required 34 iterations and the 4-color ordering required 29 iterations. Their results suggest that the multi-color ICCG methods may require more iterations for convergence but the increase was not substantial.

Figure 4.10 summarizes some other published timing results for Laplace's equation on the CRAY-1 and CYBER 205 for various ICCG methods. We describe some details of the algorithms and results below.

Two vectorized versions of ICCG based on the natural ordering are discussed in van der Vorst [1985] and results are given from runs on both the CYBER 205 and CRAY computers. The natural ordered matrix for Laplace's equation on a $r \times r$ grid is treated as block tridiagonal. A straight forward vectorization of ICCG(0) is carried out with vector lengths that are $O(r)$ using special optimized scalar arithmetic software for the recursive equations necessary to solve lower and upper bidiagonal $r \times r$ systems as part of the forward and back solves. Estimates are given for execution times of both the matrix vector multiply and the preconditioning step in terms of the number of unknowns, $N = r^2$, as $90N$ns and $520N$ns.

ICCG Preconditioning Methods for Laplace's Equation		
Author	Iterations	Timing
van der Vorst [1985]	$N = 3541$ ICCG(0) - 101 VICCG(0) - 104 $N = 22500$ ICCG(0) - 246 VICCG(0) - 264	*ICCG*(0) *seconds* CRAY-1 .392 CRAY X-MP .219 2.98 CYBER 205 .512 11.87 *VICCG*(0) *seconds* CRAY-1 .269 CRAY X-MP .148 2.22 CYBER 205 .218 3.71
Kightley & Jones [1985]	$N = 2744$ JCG - 32 ICCG - 17 VICCG - 17 LICCG - 23 $N = 15625$ JCG - 112 ICCG - 37 VICCG - 37 LICCG - 98	CRAY-1 times (sec) .049 .105 .090 .064 .94 1.14 .086 1.34
Meurant [1985]	$N = 2500$ ICCG(0) - 35 INV3(1) - 16	CRAY-1 ICCG(0) 29 mflops INV3(1) 31 mflops
Lichnewsky [1983]	$N = 2500$ VECGIC-2D - 34	CRAY-1 29 mflops

Laplace's Equation Results

Figure 4.10

respectively, for a 2-pipe CYBER 205. Note that this result agrees with the leading term in the timing formulas for A_t for Laplace's equation given in Figure 4.3. In terms of the model presented in section 4.1, the ratio of execution time for the preconditioning step to the execution time for the matrix vector multiply, α, is 5.8, considerably higher than for any of the multi-color methods. Based on the model we also estimate the ratio of the total time per iteration for ICCGN to the total time per iteration for CG, τ, as approximately 3.9. Since the ICCGE algorithm for

the Laplace problem has the same cost as CG, this ICCGN method will be slower than ICCGE on the CYBER 205 unless it converges more than 3.7 times as fast as ICCGE.

A modified ICCGN algorithm, VICCG(0), is also given by van der Vorst based on a truncated Neumann expansion of $r \times r$ unit lower bidiagonal matrices and requires only $120N$ns so that α would be only 1.33. In this case τ would be approximately 1.6 for large problems. All of the methods presented by van der Vorst are carried out using vector lengths that are $O(r)$ and so the parameter α is likely to be considerably higher than 1.6 unless a very large grid is used. No comparison to conjugate gradient is made in this paper but the results show that the ICCG(0) and VICCG(0) algorithms are nearly identical in terms of iterations for convergence and VICCG(0) outperforms ICCG(0) by a factor greater than 2. Note that this performance result is predicted by the ratio of $\tau_{i,CG}$ computed above.

Results for some three dimensional problems are given by Kightley and Jones [1985] for Poisson's equation on the unit cube with 2744 unknowns and for a fluid flow problem based on a Poisson-like matrix with mixed boundary conditions containing 15625 unknowns. The methods are standard ICCG based on the natural ordering, VICCG, as described above, and a 'long vector' truncated ICCG method, LICCG, which uses a Neumann expansion to approximate the incomplete Cholesky factors using the sub diagonals of A which are of length $O(N)$ rather than the shorter $O(r)$ lengths used by Van der Vorst. These methods are compared to Jacobi preconditioned conjugate gradient (i.e. the diagonal scaling of A) and the results indicate that JCG was faster than all of the ICCG methods except for the VICCG method of Van der Vorst on the flow problem. The LICCG method is the only one which has the desired $O(N)$ length vectors for the CYBER 205 but it does not do as well as VICCG in terms of iterations for convergence. For the Poisson problem with 2744 unknowns LICCG was the fastest ICCG method even though the number

of iterations is greater, reflecting the greater degree of vectorization. These results also indicate that many of the ICCG methods are not competitive with simpler preconditionings on some problems.

Vectorized versions of block preconditioners given by Concus, et. al. [1985] are discussed by Meurant [1984] and performance results are given for both the CRAY and CYBER 205. All of the block methods are based on the $r \times r$ block tridiagonal structure of the natural order matrix for the Laplace problem. They differ in the ways used to approximate the inverse of the tridiagonal blocks of the block diagonal matrix in the incomplete factorization. As such they all have the limitation of $O(r)$ vector lengths and the Mflop rates given for the algorithms reflect this. Again, no comparison is made to no preconditioning but the results do show an improvement in convergence rates over standard no-fill incomplete Cholesky and the claim is made that for a large class of problems block methods are to be preferred over point ones. For a good comparison of these block methods to both conjugate gradient without preconditioning and point ICCG methods, see Concus et. al. [1985]

Another approach, presented by Lichnewsky[1983], reorders the unknowns by a 'subdomain approach' mainly applicable for multi-processor applications. He suggests that the multi-coloring strategy of Schreiber and Tang could be used within the subdomains to achieve long vector lengths. He also gives results for a vectorized ICCG algorithm based on odd-even ordering by lines on the entire domain. Again, the vector lengths are $O(r)$ and results are only given for the CRAY-1. Comparisons to other ICCG methods, including those given by Meurant [1984], show similar convergence results.

4.4. Diagonal Ordered ICCG for Laplace's Equation

We turn now to an analysis of two ICCG algorithms based on diagonal orderings. The diagonal ordering is discussed by van der Vorst [1983] as a method

to vectorize the natural ordering but, as he notes, this ordering is difficult to apply to a general problem and requires twice as many vector operations (2r) for a problem on an $r \times r$ grid as well as expensive gather and scatter operations. The advantage of the diagonal ordering is that the recursion in the forward and back solves is now vectorized. Moreover, the convergence rate is the same as for the natural ordering since the same computations are performed by the diagonal ordered method as for the natural ordered method. The main disadvantage for vector computers like the CYBER 205 is that the average vector length for ICCGD is r/2 (for rectangular regions the average vector length is less than half the smaller dimension). Moreover, reordering of the solution vector is necessary at each iteration to preserve the vectorization of the matrix-vector multiplication. One solution to this problem, however, is to use the Eisenstat implementation discussed previously, thereby avoiding the need for reordering by eliminating the matrix-multiply at each step. Van der Vorst claims no savings using the diagonal ordering on the CYBER 205 but significant savings were realized on the CRAY-1.

We now compare multi-color ICCG methods to two different implementations of ICCG that use diagonal orderings. The diagonal ordering discussed in section 3.3 occurs after the incomplete factorization and is used to vectorize the forward and back solves necessary at each iteration. The first algorithm, ICCGD, is just the standard ICCG(0) method. The matrix-vector multiplication is carried out with the matrix A in the natural ordering stored by diagonals. The preconditioning step, therefore, must be preceeded by a vector gather instruction and followed by a vector scatter since the unknowns are ordered by diagonals for the forward and back solves. The second algorithm, ICCGDE, uses the Eisenstat modification to eliminate the need to do matrix-vector multiplies explicitly and, as in the red-black ICCGE method used for the Laplace problem, appropriate scaling of A saves additional operations. For ICCGDE the vector gather-scatter operations need only be carried

out at the outset and at the end of the computations.

To derive timing formulas for ICCGD we consider the number of vector multiplies and adds for a forward and back solve on a $r \times c$ dimension grid. Figure 4.11 shows a psuedo code for the preconditioning for ICCGD. The ICCGDE algorithm has no preconditioning step but the forward and back solves are carried out in the same manner. A vector assignment statement initializes the forward solve while the vector multiply used to accomplish the diagonal scaling before the backward solve initializes the backward solve. The number of vectors used to store the lower triangular matrix L is $2(r+c-2)$. Since, for each vector, an add and multiply are executed, there are $4r+4c-8$ vector instructions executed for one forward solve. The total number of operations for a forward solve is $4rc-2r-2c$. The total number of operations for the preconditioning is $9rc-4(r+c)$ and the total time including the vector assignment is $100rc+8200(r+c)-14640$. Note that

set $\hat{\mathrm{r}} = \mathrm{r}$

forward solve

for i = $1,r+c-2$
    $\hat{\mathrm{r}}(i) = \hat{\mathrm{r}}(i) - \mathrm{w}(i) * \mathrm{r}(i)$
    $\hat{\mathrm{r}}(i) = \hat{\mathrm{r}}(i) - \mathrm{w}(i) * \mathrm{r}(i)$

diagonal scaling ($D\mathrm{q} = \hat{\mathrm{r}}$)

$\hat{\mathrm{r}} = \hat{\mathrm{r}} * D^{-1}$

backward solve

for i=$r+c-2,1$
    $\hat{\mathrm{r}}(i) = \hat{\mathrm{r}}(i) - \mathrm{e}(i) * \mathrm{r}(i)$
    $\hat{\mathrm{r}}(i) = \hat{\mathrm{r}}(i) - \mathrm{n}(i) * \mathrm{r}(i)$

$$LDL^T \hat{\mathrm{r}} = \mathrm{r}$$

Vectorized Preconditioning for ICCGD, ICCGDE

Figure 4.11

although the number of operations is actually a little less than for the preconditioning in the ICCGC routine (see Figures 4.6 and 4.3) the main difference in the two timing results is due to the large positive coefficient of the $r+c$ term in the ICCGD preconditioning. For rectangular grids the effect is even greater.

The cost of preconditioning for ICCGD increases greatly as the grid of unknowns becomes elongated in one dimension. In Figure 4.12 we give a comparison of the two preconditioners on almost square grids and on very elongated grids. The timings given for ICCGD are estimates from timing formulas. No experimental results were obtained for this method but the timing formulas are believed to be accurate estimates of the actual run time. The multi-color method is clearly superior in execution time in all cases, particularly for the rectangular grids.

We now compare the standard ICCGD algorithm with the Eisenstat implementation , ICCGDE, used earlier for the red-black ordering. We can predict overall performance of these algorithms by applying the model described in section 4.1. To compute α and β for each problem we use the time for a matrix-vector multiply for the natural ordering obtained from timing formulas derived as in earlier examples. The time for preconditioning for the ICCGD method includes the

Grid		ICCGD - Predicted Times			ICCGC - Actual Run Times		
r	c	operations	time (sec)	mflops	operations	time (sec)	mflops
99	101	89,191	(.0026)	34	89,595	.00101	88
11	909	86,311	(.0085)	10	89,947	.00102	88
49	51	22,091	(.00106)	21	22,295	.00027	84
7	357	21,035	(.00322)	6.5	22,463	.00027	84

Times per Iteration for Laplace Problem

Comparison of ICCGC and ICCGD Preconditioners

Figure 4.12

Grid Size	ICCGD			ICCGDE		
	α	β	$\tau_{D,CG}$	α	β	$\tau_{DE,CG}$
99×101	3.46	1.0	2.73	2.90	1.00	1.95
11×909	9.94	1.0	5.97	9.39	1.00	5.19
49×51	5.10	1.00	3.55	4.54	1.00	2.77
7×357	14.31	1.00	8.16	13.77	1.00	7.38

Time per Iteration in Terms of Matrix-Vector Multiply for:
α – preconditioning (ICCGD), diagonal matrix multiply (ICCGDE)
β – remaining CG calculations
τ – ratio of ICCGD or ICCGDE to CG

Performance Model for ICCGD and ICCGDE

Figure 4.13

overhead of the gather and scatter operations which must be performed at each iteration and this is reflected in Figure 4.13 in the larger value of α for ICCGD compared to ICCGDE. We can also see the effects of a change in dimensions of the problem on the various parameters. The ICCGDE method does not have an explicit matrix-vector multiply and in this case α is a measure of the time for the equivalent of a forward and back solve, a vector multiply and a vector assignment divided by the time for a matrix-vector multiply for the natural ordering. Each β represents the time per iteration for the usual dot products and three linked triads. For the ICCGD algorithm, (4.1) and (4.2) are used to compute the values for $\tau_{D,CG}$ shown in Figure 4.13. The form used to calculate $\tau_{DE,CG}$ is $\tau_{DE,CG} = (\alpha_{DE} + \beta_{DE})/(1 + \beta_{CG})$.

The results of Figure 4.13 suggest that the ICCGD and ICCGDE algorithms may actually be slower than CG for many problems since preconditioning does not always reduce the number of iterations by a large enough factor. Nevertheless, it is important to remember that these performance results only give a comparison of the computation rates for these methods and a complete answer to the question of which

method is better requires knowledge of the convergence rates. Since CG and the ICCGE method have almost the same cost on Laplace's equation (i.e. $\tau_{E,CG} = 1.06$), a diagonal order based algorithm must reduce iterations by greater than the ratios, τ, computed in Figure 4.13 in order to be faster than ICCGE.

We turn finally to the comparison of convergence rates for ICCGC and ICCGE compared to ICCGN. Our results confirm the convergence results given by Schrieber and Tang [1982] which show some increase in the number of iterations for convergence for multi-color methods but not by a great deal. It is possible, however, to construct pathological problems where the ICCGN method appears far superior to the multi-color method. However, we do not feel that such problems are representative of real problems of interest. As an example, we consider the Laplace problem on a grid with spacing between the grid points equal to h in one direction and k in the other direction. Then the main diagonal of A is $2(h^2+k^2)/hk$, the diagonals above and below the main diagonals are $-k/h$ and the outer diagonals are $-h/k$. As the aspect ratio, h/k, is changed the entries in the matrix change but the condition number of the matrix remains the same, as can be shown using a Kronecker product representation of A. The conjugate gradient iterations first increase as the aspect ratio is increased but then decrease as the aspect ratio continues to increase. The ICCGN algorithm converges more rapidly, however, as the aspect ratio increases. For example, on the Laplace problem described in the previous section with $N = 97 \times 97 = 9409$, increasing the aspect ratio to 100 caused the number of iterations for ICCGN to drop from 70 to 6 while the ICCGC iterations increased to 108. For very high aspect ratios, however, the matrix becomes essentially a tridiagonal matrix since two of the off diagonals become very large and the other two become very small. Incomplete Cholesky decomposition of a tridiagonal matrix is exact so we expect that for these matrices the preconditioning by incomplete Cholesky would be very effective.

Model Problem	CG	ICCGC	ICCGN	ICCGC/ICCGN
Laplace $N = 9999$	266	130	84	1.5
Mxdir $N = 11236$	230	87	45	1.9
Plane Stress $N = 12960$	794	298	223	1.3

Iterations for ICCGN and ICCGC Convergence

Figure 4.14

As a further comparison of ICCGN to ICCGC we programmed point ICCGN for the Laplace problem, the mixed derivative problem and the plane stress problem. No attempt was made to optimize the CYBER 205 code for maximum scalar speed and so the runtimes were extremely slow; Figure 4.14 summarizes only the iteration counts for three problems. The sizes for each problem were the same as in Figure 4.8. Although the ICCGN algorithm does require fewer iterations, the improvement is not nearly enough to offset the negative effects of poor vectorization of the preconditioning step.

Summary

The results presented in this chapter compare multi-color ICCG methods with both standard conjugate gradient and other ICCG methods. A performance model was given which accurately predicts the results obtained from experiments and can also be used to compare other ICCG methods. For each model problem we saw that the ICCGC methods performed at high computation rates on the CYBER 205 and achieved modest speedups in execution time compared to conjugate gradient.

CHAPTER 5

Conclusions and Future Research Areas

In this thesis we have examined multi-color orderings applied to the incomplete Cholesky conjugate gradient method. We sought to expand earlier results on multi-color orderings for application to a wide class of problems and to see what effect multi color orderings had on the rate of convergence of the basic ICCG method as compared to the natural order point ICCG method and other vectorized ICCG methods. We now summarize our work, draw conclusions about the experimental results, and discuss future research in related areas.

5.1. Summary

Following a discussion of the basic ICCG method and some commonly used modifications for both scalar and vector computers we described the multi-color ICCG method, ICCGC, based on reordering of the unknowns to obtain p-color matrices. The resulting block incomplete Cholesky method could be implemented with sufficiently long vectors so that the preconditioning step, requiring a forward and back solve of block lower and upper triangular systems, could be implemented with adds and multiplies of vectors of length $O(N/p)$. Chapter 3 defined a class of problems, class $\mathcal{R}$, for which multi-color orderings could be applied. This class included two and three dimensional problems on rectangular domains with Dirichelet boundary conditions and with possibly more than one unknown per grid point and a uniform grid stencil. For the problems in class $\mathcal{R}$ we used storage of the matrix by diagonals and in Theorem 3.1 we proved that for problems with one unknown per grid point, the natural ordering of the grid points resulted in a matrix which could

be stored in the minimum number of diagonals possible. In Theorem 3.1 we also proved that for problems with more than one unknown per grid point, using the natural ordering of the grid points with either a consecutive (3.1) or alternating (3.2) ordering of the unknowns at each grid point, the number of diagonals in the matrix was the same as the number of non-zero coefficients of the equations associated with all of the unknowns at any interior grid point which had no boundary values as grid stencil neighbors. Furthermore, bounds were given for the number of diagonals in the matrices using these orderings.

We then discussed multi-color orderings, defined in terms of p disjoint sets of the unknowns in the grid, and in Theorem 3.2 gave neccessary and sufficient conditions to ensure that a p-color matrix results from a given p-color ordering of the unknowns. We also identified a relationship between the ordering of the unknowns within the disjoint sets which would minimize the number of diagonals within each block row of the p-color matrix and illustrated the relationship with several examples. We presented a method of obtaining p-color orderings, called the *continuous coloring rule*, which is easy to apply to any class $\mathcal{R}$ problem. In Theorem 3.3 a condition was given which applied to all continuous color orderings and satisfied the necessary and sufficient conditions of Theorem 3.2 to obtain p-color matrices. The condition in Theorem 3.3 placed restrictions on the dimensions of the grid and we discussed ways to handle this problem including increasing the number of colors and adding extra 'dummy rows'. Increasing the number of colors seemed to be the most promising solution to this problem and examples were given of both solutions. We noted that the continuous coloring rule also gave matrices which contained the minimum number of diagonals within each block row and further noted that many of the diagonals lined up with diagonals in adjacent blocks so that a further savings could be achieved in the matrix vector multiply in the conjugate gradient step by storing these vectors contiguously. We conjectured that the

maximum lineup of vectors occurs when p divides N evenly. We saw that for a fairly general class of problems p-color orderings could be easily obtained which had the desirable long vectors suitable for vector computers such as the CYBER 205. To our knowledge such a wide application of p-color orderings has not been given and no other easy to use method like the continuous coloring rule has been applied to obtain p-color matrices, particularily to three dimensional problems like the space platform model problem.

Having developed a method for obtaining p-color matrices, we tested the effectiveness of multi-color ICCG on four model problems. We were interested in the degree of vectorization achieved by the ICCGC method but, more importantly, in the overall speedup of ICCGC compared to conjugate gradient, CG. We developed a performance model in chapter 4 which compared the time for a matrix-vector multiply with the time for the preconditioning step. Since the two computations in the basic ICCG method have nearly the same number of operations the comparison of execution time of the two parts of the computation was a measure of the degree of vectorization of the preconditioning. We saw that ICCGC vectorized very well, with the preconditioning step nearly equal to the matrix vector multiplication in execution time. However, for the diagonal ordered ICCG method used on Laplace's equation we saw that the preconditioning step did not vectorize nearly as well and on rectangular domains where one of the dimensions was much larger than the other in term of grid points the degree of vectorization was very poor. We also gave formulas to count the arithmetic operations and predict execution time for each of the model problems. The performance model predicted the actual speedups achieved by ICCGC compared to ICCG within one decimal place accuracy and sometimes even better. We also gave results for the Eisenstat-like modification to the ICCGC algorithm showing that it does save execution time but less so as the number of colors increases. Finally, we modeled the performance of diagonal ordered ICCG and

compared the convergence of natural ordered ICCG for three of the model problems, noting that the multi-color orderings required a greater number of iterations to convergence. However, the increase in iterations for ICCGC compared to the natural ordering was small compared to the speedup due to the vectorization achieved by the multi-color orderings.

5.2. Conclusions

1) Multi-color orderings are an effective means to achieve matrix structures for which block methods can be carried out with long vector operations. Because of the structure of the p-color matrices the forward and back solves needed to carry out the preconditioning can be implemented with the desired long vector lengths. The multi-color ICCG methods we implemented ran at near the maximum possible rate on the CYBER 205.

2) The continuous coloring rule provides an easy to use method for obtaining p-color matrices even for more complicated three dimensional problems. We gave a condition in Theorem 3.3 which could easily be used to determine the number of colors necessary to obtain a p-color matrix given a grid stencil and the dimensions of the grid. We also noted that for three dimensional problems a simple but effective strategy to follow to obtain the number of colors was to chose p and then apply (3.8), repeating with the next larger p if necessary until (3.8) is satisfied.

3) Our results showed that the ICCGC methods are competitive with other vectorized ICCG methods in terms of overall speedup of execution time compared to conjugate gradient. We noted that often published results for vectorized ICCG methods do not include a comparison to conjugate gradient without preconditioning and in some cases when such a comparison is included, conjugate gradient was faster in overall execution time.

4) The performance model we have presented can be used to compare other preconditioning methods to ICCGC and CG and represents the proper way to compare methods implemented on vector computers by modeling both the degree of vectorization and the overall performance in terms of execution time. We were also able to note the effect of poorly vectorized preconditioners by measuring the time for the preconditioning step in terms of the matrix-vector multiply time.

5.3. Future Areas of Research

The speedups achieved by ICCGC are only modest and some method of improving the convergence of the method would greatly improve the results. We have so far only worked with basic ICCG and an open question is how much the ICCG convergence results can be improved by applying some of the modifications of the basic method to the multi-color ICCG method. It is also possible that multi-color orderings can be used in the context of other block iterative methods to improve vectorization of those methods. Some of these methods include allowing partial fill or adding column sum constraints. Another promising modification may be to allow fill in the D_i blocks in the decomposition and apply the Neumann expansion to estimate the D_i^{-1}. Still another is to use matrices such as in Figure 3.17 which are obtained by the continouus coloring rule but are not p-color matrices and apply some of the commonly used methods for approximating the inverses of tridiagonal inverses to approximate the D_i^{-1}.

More research related to the theoretical results we obtained in chapter 3 is also needed. The proof (or disproof) of the conjecture remains, as well as further results extending the general statement of conditions to achieve the minimum number of diagonals within the blocks of the p-color matrix when there is more than one unknown per grid point. More theoretical results are also necessary to compare the properties of p-color matrices to their natural ordered counterparts. Research on the

convergence test used for ICCGC, particularily in the use of the two norm of $\hat{\mathbf{r}}$, might further reduce the time and overhead spent testing for convergence.

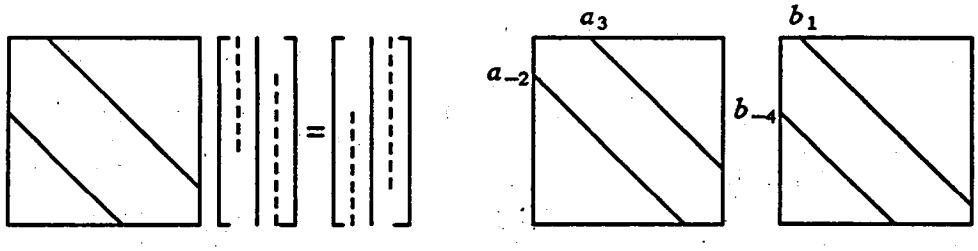
Finally, application of multi-color orderings to irregular domains is another important area of investigation.

APPENDIX A

Incomplete Matrix Multiplication by Diagonals

In this appendix we discuss matrix multiplication by diagonals as applied to multi-color ICCG methods. We adapt an algorithm for matrix multiplication by diagonals given by Madsen et al. [1976] for banded matrices. A general discription of the problem is given first, followed by the actual implementation used in programs written for the CYBER 205.

Recall that the main computation performed in the block incomplete factorization for ICCGC is incomplete matrix-matrix multiplication while matrix-vector multiplication is the main computation in the forward and back solves in each iteration as well as the formation of $A\mathbf{p}$ in the CG algorithm. The matrix-vector multiplication is the easier of the two operations and Figure A.1a illustrates the process. Here, a superdiagonal multiplies the corresponding first positions of $\mathbf{w}$ while a subdiagonal multiplies the corresponding last positions of $\mathbf{w}$. These



(a) $A\mathbf{w}=\mathbf{z}$ (b) AB

Matrix Multiplication by Diagonals

Figure A.1

contributions are added to the correct positions of the result vector z.

The more difficult problem is the incomplete multiplication $AB=C$ where A and B are $q \times q$ matrices. Each diagonal of the product is of the form

$$c_k = \sum a_i b_j$$

where the a_i and b_j are diagonals of A and B and the summation is over all products which contribute to the product diagonal c_k. For the incomplete multiplication, we do only those calculations for diagonals c_k which are allowed to be non-zero. The programming problem is to determine the diagonals of A and B which contribute to a diagonal of C, the starting positions and lengths of the diagonal operands and the starting position of the result in the product diagonal. For example, in Figure A.1b, suppose that c_{-1}, the first subdiagonal of C, is an allowed non-zero diagonal. Then $a_{-2}b_1$ and $a_3 b_{-4}$ contribute to c_{-1} in the following way. The $q-2$ long vector a_{-2} multiplies the first $q-2$ positions of the $q-1$ long vector b_1 and is stored in c_{-1} beginning in the second position. Then the last $q-4$ elements of the $q-3$ long vector a_3 multiply the $q-4$ elements of b_{-4} and the product vector is added to the first $q-4$ positions of c_{-1}.

We turn now to implementation details of the two processes described above. The data structure used for the programs to solve the four model problems was as follows. The lower triangular part of A is stored by diagonals in a one dimensional array. One table of integer pointers contains necessary information for each diagonal including the row and column in A where the diagonal begins, the length of the diagonal, the starting position of the diagonal in the array, and the offset of the diagonal within the block structure of the matrix. That is, the row and column information tells where the diagonal is located within the matrix A while the offset field tells where the diagonal is located within the particuliar block in which it is located in the multi-colored matrix. A second table of integers tells how many diagonals are located within each block of the multi-colored matrix and

which diagonals are located in each block.

Matrix-vector multiplication by diagonals is easily implemented if for each diagonal of A the row and column in A where each diagonal begins and the length of the diagonal is known. Figure A.2 is a portion of the CYBER 200 FORTRAN subroutine used to do the matrix-vector multiplication in the conjugate gradient iteration. The first vector instruction multiplies the main diagonal of A by the input vector. The DO loop goes through the data structure containing the necessary integer pointers for each of the ND diagonals in A looking up the row, column, length, and starting position for each vector. Each diagonal is used twice since the matrix is symmetric. Note that the row and column information is used directly to specify the starting positions in the input and resultant vectors and the roles of the row and column pointers are exchanged for the second vector instruction in the loop, accounting for symmetry of the matrix. Note also that the forward and back solves can be carried out in a fashion very similar to the matrix-vector multiply, using the row and column information to determine the starting positions in the appropriate vectors. An additional savings is possible in the formation of $A\mathbf{p}$ if diagonals which line up across blocks are stored contiguously and treated as one

```
      SUBROUTINE ATIMV(VIN,VOUT)

      VOUT(1;N)=VIN(1;N)*A(IDST;N)

      DO 1 I = 1,ND
         COL = ID(I,1)
         ROW = ID(I,2)
         L = ID(I,3)
         IST = ID(I,4)
         VOUT(ROW;L) = VOUT(ROW;L) + VIN(COL;L)*ARRAY(IST;L)
         VOUT(COL;L) = VOUT(COL;L) + VIN(ROW;L)*ARRAY(IST;L)
1     CONTINUE
```

Matrix-Vector Multiplication

Figure A.2

vector during the matrix-vector multiplication. Experimental results with the model problems verified that this savings was most significant for smaller problems while for larger problems where vector lengths were long within blocks of the multi-color matrix the savings was insignificant. Figure A.3 gives some timing results for the plane stress problem comparing the execution time for $A\mathbf{p}$ with and without this savings for a problem with 840 unknowns and one with 12960 unknowns. Figure A.3 shows the time for one matrix-vector multiply as well as one ICCGC iteration. For the smaller problem, a savings in execution for ICCGC of approximately eight percent is realized while for the larger problem a savings of just over one percent is achieved. Note the increase in operations required for the lineup of vectors, reflecting the extra zeros stored and used in the calculations. For this problem approximately half of the vector instructions can be eliminated by taking advantage of the vectors that line up.

Matrix-Vector Multiply for Plane Stress Problem				
Grid Coloring	Time (sec)	Number of Operations	Vector Instructions	Total ICCG Time/Iteration
without line up 80 X 81	.003159	296,144	133	.0080
with line up 80 X 81	.003105	297,000	69	.0079
without line up 20 X 21	.000393	18,824	133	.001001
with line up 20 X 21	.000308	19,040	69	.000935

Matrix Multiplication Comparison

Figure A.3

The matrix-matrix multiplication performed in the block incomplete factorization requires that blocks of A be treated as separate matrices and so the offset field mentioned above is used rather than the row and column fields for each diagonal. We wish to do computations of the form $C=C-DE$ where the non-zero structure of C occurs along a few preselected diagonals and the multiplication DE is only carried out for those diagonals of D and E which contribute to the allowed structure of C. To describe the implementation of this incomplete matrix-matrix multiplication we use the following notation to describe a diagonal within $D, E, or\ C$. $D_{j,j+\alpha}$ denotes all elements of D which lie on a diagonal within D which is α units above (or below if α is negative) the main diagonal. Using this notation, and assuming that the matrices are not necessarily square, we can write the following expression for one of the vector multiplies contributing to the desired product.

$$C_{j,j+\alpha+\beta} = C_{j,j+\alpha+\beta} - D_{j,j+\alpha}\, E_{j+\alpha,j+\alpha+\beta} \tag{A.1}$$

It is easy to see that given any diagonal of C with some offset γ, and any two diagonals of D and E having offsets α and β, the two diagonals contribute to diagonal γ of matrix C only if $\alpha+\beta=\gamma$. If C is $p\times q$, A is $p\times s$ and E is $s\times q$, then the following inequalities must be true for all allowable values of j.

$$(a)\ \ 1 \leqslant j \leqslant p \tag{A.2}$$

$$(b)\ \ 1 \leqslant j+\alpha \leqslant s$$

$$(c)\ \ 1 \leqslant j+\alpha+\beta \leqslant p$$

From these inequalities we can derive the following relation for j:

$$\max(1, 1-\alpha, 1-\alpha-\beta) \leqslant j \leqslant \min(p, s-\alpha, q-\alpha-\beta) \tag{A.3}$$

Using (A.3) we compute the length of the vector multiply and add for each pair of diagonals of D and E. If we denote the left hand term in (A.3) as $j_{\min}$ and the right hand term as $j_{\max}$, then the length for the vector instructions, l, is $j_{\max}-j_{\min}+1$. Determining the starting positions for the two vector operands and the resultant vector is the more difficult problem. Recall that for each stored

for each γ in C

for each α in D

for each β in E

if $\alpha+\beta=\gamma$ then do

$j_{\min}=MAX(0,-\alpha,-\alpha-\beta)$

$j_{\max}=MIN(p,s-\alpha,q-\alpha-\beta)$

$lth=j_{\max}-j_{\min}$

$D_{start}=IDD+j_{\min}-MAX(0,-\alpha)$

$E_{start}=IDE+j_{\min}-MAX(-\alpha,-\alpha-\beta)$

$C_{start}=IDC+j_{\min}-MAX(0,-\alpha-\beta)$

$ARRAY(C_{start};lth)=ARRAY(C_{start};lth)-ARRAY(D_{start};lth)*ARRAY(E_{start};lth)$

Matrix - Matrix Multiplication Algorithm

Figure A.4

diagonal we know the starting position but for particuliar values of α and β the starting position for one of the three vectors is shifted. The algorithm shown in Figure A.4 is used to determine the starting positions for each of the vectors given the starting locations of each vector, *IDD*, *IDE*, and *IDC*, and the offsets for each vector.

We conclude with an example to illustrate the above discussion. Let us consider the mixed derivitive model problem on a 6×6 grid using four colors similiar to the 6×8 grid shown in Figure 3.20. Figure A.5 lists the integer pointers stored for this problem. The integer array ID(I,6) contains the row, column, length, starting location and offset for each of the 16 diagonals in this matrix. The fifth field (*long*) in ID contains a second length for use when one wishes to take advantage of diagonals which line up in A. For example, the first

ID(I,6) Data Structure						
I	column	row	length	start	long	α
1	3	10	7	1	25	2
2	1	10	9	28	27	0
3	1	11	8	55	25	-1
4	3	19	7	81	16	2
5	12	19	7	10	0	2
6	10	19	9	37	0	0
7	1	20	8	99	17	-1
8	10	20	8	64	0	-1
9	12	28	7	90	0	2
10	3	28	7	116	7	2
11	21	28	7	19	0	2
12	2	28	8	125	8	1
13	19	28	9	46	0	0
14	10	29	8	108	0	-1
15	19	29	7	73	0	-1
16	1	29	8	134	8	-1
17	1	1	9	142	36	0
18	10	10	9	151	0	0
19	19	19	9	160	0	0
20	28	28	9	169	0	0

NTABLE(I,J,K) Data Structure					
I	J	K=1	2	3	4
1	1	1	17		
2	1	3	1	2	3
2	2	1	18		
3	1	2	4	7	
3	2	3	5	6	8
3	3	1	19		
4	1	3	10	12	16
4	2	2	9	14	
4	3	3	11	13	15
4	4	1	20		

Data Structures for Mixed Derivative Problem

Figure A.5

vector stored is the B diagonal in Figure 3.20, block 2,1. The second vector in ID is the W diagonal in block 2,1 of the matrix in Figure 3.20 and the third vector in ID is the Z diagonal in the same block. Each of these diagonals lines up with successive B, W, and Z diagonals respectively. Careful inspection of the starting positions in the ID data structure in Figure A.5 reveals that these successive diagonals are stored so that they form continuous B, W, and Z vectors. Vectors with a zero in column 5 line up with some previous vector. Note that for the

mixed derivative problem using column 5 for the matrix-vector multiplication, Ap, requires $4 \times 8 = 32$ vector instructions while using column 3 requires $4 \times 16 = 64$ vector instructions with correspondingly shorter vector lengths.

To illustrate the use of NTABLE we consider the calculation $L_{3,2} = A_{3,2} - A_{3,1} * L_{2,1}^T$ which is a portion of the equation describing the calculation of $L_{3,2}$ according to equation (2.4). From NTABLE(3,2,1) we see that $L_{3,2}$ has 3 diagonals and the 2nd, 3rd and 4th fields identify the diagonals as numbers 5, 6, and 8 in ID. The allowable non-zero structure of the incomplete multiplication described above will lie along the offsets 0, 2, and -1 given by field 6 in ID. In similar fashion we find the diagonals of $A_{3,1}$ and $L_{2,1}^T$ to be 4, 7 and 1, 2, and 3. The offsets can also be looked up in ID but we must remember that $L_{2,1}^T$ is above the main diagonal and so we take the additive inverse of each offset for vectors 1, 2, and 3. Finally we take all possible combinations of pairs of offsets from the two operand matrices allowing multiplications only where the sum of the pair is equal to one of the offsets of the diagonals in $A_{3,2}$. For this example diagonal 4 will be multiplied by diagonals 1 and 2 while diagonal 7 will be multiplied by 2 and 3. Unnecessary multiplications are diagonal 4 times diagonal 3 and diagonal 7 times diagonal 1. A partial fill strategy, which could be easily implemented, would allow fill in blocks of L below the diagonal blocks by allowing all of the multiplications. Of course, one would have to alter ID and NTABLE accordingly.

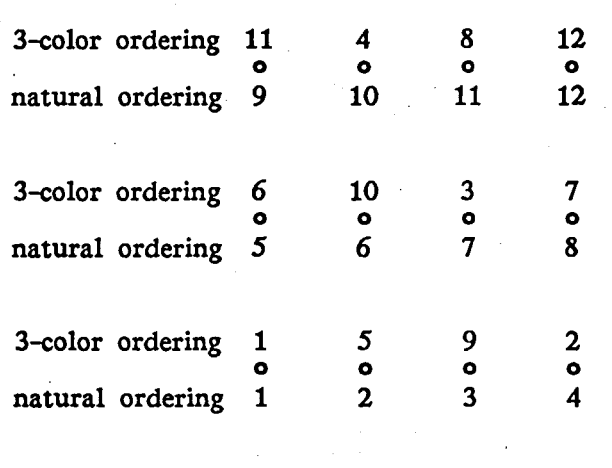
APPENDIX B

Matrix Assembly by Diagonals

We discuss now a general procedure used to assemble matrices row by row using diagonal storage. This procedure was used to assemble the matrices for the four model problems discussed in this dissertation. We store the lower triangular portion of each symmetric matrix. We assume a rectangular grid of unknowns in two or three dimensions with possibly more than one unknown at each grid point. For a grid with r points per row, c rows, and l planes with k unknowns at each point, then there are $N = r \times c \times k$ equations and A is a symmetric $N \times N$ matrix. Associated with each unknown is an equation in N variables whose non-zero coefficients are described by a grid stencil. The structure of the matrix depends upon the grid stencil and the ordering of the grid points.

To specify the ordering used, two ordering vectors, ORD(I,1) and ORD(I,2), are formed. These N-long vectors are permutations of the integers 1 thru N. To see how these vectors are used we consider an example for a 4×3 grid. The unknowns will be numbered from 1 to N from left to right, bottom to top as in the natural ordering. The first ordering vector, ORD(I,1), indicates which unknown is associated with row I in the matrix A. The second ordering vector, ORD(I,2), tells which row in the reordered matrix is associated with the Ith unknown.

The information in the two ordering vectors is used as follows. Suppose we are assembling row 7 for the grid in Figure B.1 for the Laplace problem. In the 3-color ordering used, ORD(7,1) = 8 so the fourth grid point in row 2 of the grid is associated with equation 7 in A. To calculate the coefficients for this equation we use the 5 point stencil. The north, south and west neighbors are the 12th, 4th



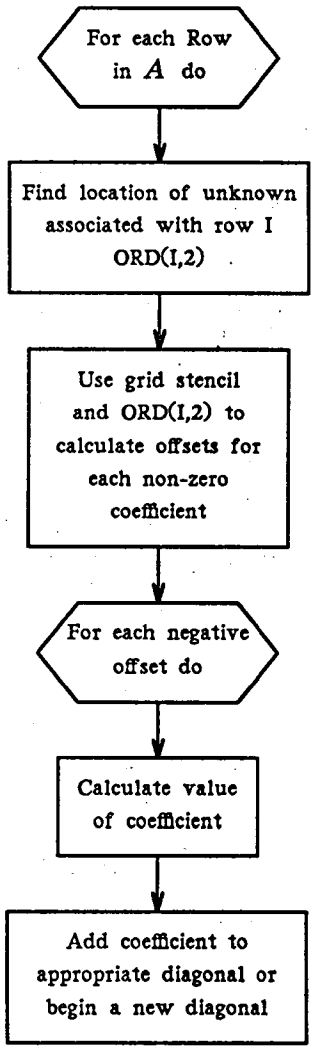
I	ORD(I,1)	ORD(I,2)
1	1	1
2	4	5
3	7	9
4	10	2
5	2	6
6	5	10
7	8	3
8	11	7
9	3	11
10	6	4
11	9	8
12	12	12

4 × 3 Grid of Unknowns

Figure B.1

and 7th grid points respectively. There is no east coefficient for this grid point. To calculate where these coefficients are in A, however, we need to know which equation is associated with each point. For the west coefficient, ORD(7,2) = 3, meaning a non-zero entry will occur in column 3 of row 7 in A. This coefficient will be stored in the diagonal with offset -4. If this diagonal already has entries from previous rows, the new coefficient is added but if there is no diagonal with offset -4 a new diagonal is started and appropriate entries are made in the data structure. Likewise, the south coefficient is stored in the diagonal with offset -5. However, for the north coefficient the offset of the diagonal in A is 5. This diagonal is above the main diagonal in A and hence is not stored.

We now summarize the overall procedure. Given the ordering vectors and a grid stencil we proceed through the rows of A. At each row, using the stencil, we calculate the offset of each coefficient. If the offset is positive, we do nothing since that coefficient is above the main diagonal of A. If an offset is negative, we then compute its value and add it to the appropriate diagonal in A or create a new diagonal if necessary. Appropriate adjustments are made to the data structure so



Row by Row Matrix Assembly

Figure B.2

that when the last row is finished the number of diagonals is known and the starting positions, offsets, and lengths for each is stored. Figure B.2 gives a flowchart for the row by row assembly process.

Using this row by row assembly process for each of the model problems, we were able to experiment with various multi-color orderings, noting the effect of changes in the orderings on the performance of the ICCGC algorithm. As an example,

Ap cost per iteration				
grid coloring	time (sec)	number of operations	vector instructions	CG time/iteration
continuous coloring 80 X 81	.00316	296,144	133	.00476
not continuous 81 X 80	.00479	448,368	205	.00625

Plane Stress Problem With/Without Continuous Coloring Rule

Figure B.3

for the plane stress problem an 80 × 81 grid with two degrees of freedom per grid point can be colored with 6 colors following the continuous coloring rule discussed in chapter 3. If the grid is 81 × 80, 6 colors still decouple the equations into a 6-color matrix but the continuous coloring rule cannot be followed and so more diagonals are required to store A. The effect on the storage requirements and execution times is summarized in Figure B.3. Both the storage required for A and the execution time for a matrix-vector multiply are increased by over fifty percent.

References

Adams, L. [1983a]. "Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers," Ph.D Dissertation, Department of Applied Mathematics, University of Virginia; also published as NASA CR-166027, NASA Langley Research Center.

Adams, L. [1983b]. "An M-Step Preconditioned Conjugate Gradient Method for Parallel Computation," *Proc. 1983 Int. Conf. Par. Proc.*, pp. 36-43.

Adams, L. [1985]. "M-Step Preconditioned Conjugate Gradient Methods," *SIAM J. Sci. Stat. Comput.*, **6**, pp. 452-463.

Adams, L. and Ortega, J. [1982]. "A Multi-Color SOR Method for Parallel Computation," *Proc. 1982 Int. Conf. Par. Proc.*, pp. 53-56.

Axelsson, O. [1984]. "On Some Versions of Incomplete Block-Matrix Factorization Iterative Methods," *J. Lin. Alg. Appl.*, **58**, pp. 3-15.

Becker, E., Carey, G., and Oden, J. [1981]. *Finite Elements : An Introduction*, **1**, Prentice-Hall, Englewood Cliffs, New Jersey., pp. 242-245.

Concus, P., Golub, G. H., and Meurant, G. [1985]. "Block Preconditioning for the Conjugate Gradient Method," *SIAM J. Sci. Stat. Comput.*, **6**, pp. 220-252.

Dubois, P., Greenbaum, A., and Rodrigue, G. [1979]. "Approximating the Inverse of a Matrix for Use in Iterative Algorithms on Vector Processors," *Computing*, **22**, pp. 257-268.

Eisenstat, S. [1981]. "Efficient Implementation of a Class of Preconditioned Conjugate Gradient Methods," *SIAM J. Sci. Stat. Comput.*, **2**, pp. 1-4.

Evans, D. (ed.) [1983]. *Preconditioning Methods: Analysis and Applications*, Gordon and Breach, New York.

Gustaffson, I. [1978]. "A Class of First Order Factorization Methods," *BIT*, **18**, pp. 142-156.

Hestenes, M. and Stiefel, E. [1952]. "Methods of Conjugate Gradients for Solving Linear Systems," *J. Res. Nat. Bur. Standards Sect. B*, **49**, pp. 409-436.

Horowitz, E. and Sahni, S. [1978]. *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, Maryland., pp. 343-347..

Jackson, C. and Robinson, P. [1981]. "A Numerical Study of Various Algorithms Related to the Preconditioned Conjugate Gradient Method," AERE Harwell Report No. HL82/3304

Johnson, O., Mitchelli, C., and Paul, G. [1983]. "Polynomial Preconditioners for Conjugate Gradient Calculations," *SIAM J. Numer. Anal.*, **20**, pp. 362-376.

Kershaw, D. [1978]. "The Incomplete Cholesky-Conjugate Gradient Method for the Iterative Solution of Systems of Linear Equations," *J. Comp. Phys.*, **26**, pp. 43-65.

Kershaw, D. [1982]. "Solution of Single Tridiagonal Linear Systems and Vectorization of the ICCG Algorithms on the CRAY-1," *Parallel Computations*, G. Rodrigue (ed.), Academic Press, New York., pp. 85-99.

Kightley, J. and Jones, I. [1985]. "A Comparison of Conjugate Gradient Preconditionings for Three-Dimensional Problems on a Cray-1," *Comp. Phy. Comm.*, **37**, pp. 205-214.

Lichnewsky, A. [1983]. "Some Vector and Parallel Implementations for Preconditioned Conjugate Gradient Algorithms," Proceedings of the NATO Workshop on High-Speed Computations, Springer-Verlag, Berlin.

Madsen, N. K., Rodrigue, G. H., and Karush, J. I. [1976]. "Matrix Multiplication by Diagonals on a Vector/Parallel Processor," *Inf. Proc. Letts.*, **5**, pp. 41-45.

Manteuffel, T. A. [1980]. "An Incomplete Factorization Technique for Positive Definite Linear Systems," *Math. Comp.*, **34**, pp. 473-497.

Meijerink, J. A. and van der Vorst, H. A. [1977]. "An Iterative Solution for Linear Systems of Which the Coefficient Matrix is a Symmetric M-Matrix," *Math. Comp.*, **31**, pp. 148-162.

Meijerink, J. A. and van der Vorst, H. A. [1981]. "Guidelines for the Usage of Incomplete Decompositions in Solving Sets of Linear Equations as They Occur in Practical Problems," *J. Comp. Phys.*, **44**, pp. 134-155.

Meurant, G. [1984]. "The Block Preconditioned Conjugate Gradient Method on Vector Computers," *BIT*, **24**, pp. 623-633.

Munksgaard, N. [1980]. "Solving Sparse Symmetric Sets of Linear Equations by Preconditioned Conjugate Gradients," *ACM Trans. Math. Software*, **6**, pp. 206-219.

Poole, E. and Ortega, J. [1984]. *Incomplete Cholesky Conjugate Gradient on the CYBER 203/205*, Supercomputer Applications, R. Numrich(Ed.),Plenum Press., pp. 19-28.

Robert, I. [1982]. "Regular Incomplete Factorizations of Real Positive Definite Matrices," *J. Lin. Alg. Appl.*, 48, pp. 105-117.

Schreiber, R. and Tang, W. [1982]. "Vectorizing the Conjugate Gradient Method," Proc. Symp. Cyber205 Applications, Ft. Collins, Co.

van der Vorst, H. [1982]. "A Vectorizable Variant of some ICCG Methods," *SIAM J. Sci. Stat. Comput.*, **3**, pp. 350-356.

van der Vorst, H. [1983]. "On the Vectorization of Some Simple ICCG Methods," Paper presented at the 1st Inter. Coll. on Vector and Parallel Computing in Scientific Applications, Paris.

van der Vorst, H. [1985]. "The Performance of Fortran Implementations for Preconditioned Conjugate Gradients on Vector Computers," Report of the Dept. of Mathematics and Informatics no. 85-09, Delft University of Technology.

Young, D. [1971]. *Iterative Solution of Large Linear Systems*, Academic Press, New York., pp. 100-105.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-178117		

4. Title and Subtitle	5. Report Date
Multi-Color Incomplete Cholesky Conjugate Gradient Methods for Vector Computers	May 1986
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Eugene L. Poole	
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
University of Virginia Applied Mathematics Department School of Engineering and Applied Science Charlottesville, VA 22901	NAG1-242
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Contractor Report
National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code 505-90-21-02

15. Supplementary Notes

This report was prepared in partial fulfillment of the requirements for a Ph.D degree in Applied Mathematics from the University of Virginia.
Langley Technical Monitor: John N. Shoosmith

16. Abstract

In this research, we are concerned with the solution on vector computers of linear systems of equations, $Ax = b$, where A is a large, sparse symmetric positive definite matrix. We solve the system using an iterative method, the incomplete Cholesky conjugate gradient method (ICCG). We apply a multi-color strategy to obtain p-color matrices for which a block-oriented ICCG method is implemented on the CYBER 205. (A p-colored matrix is a matrix which can be partitioned into a pXp block matrix where the diagonal blocks are diagonal matrices.) This algorithm, which is based on a no-fill strategy, achieves O(N/p) length vector operations in both the decomposition of A and in the forward and back solves necessary at each iteration of the method.

We discuss the natural ordering of the unknowns as an ordering that minimizes the number of diagonals in the matrix and define multi-color orderings in terms of disjoint sets of the unknowns. We give necessary and sufficient conditions to determine which multi-color orderings of the unknowns correspond to p-color matrices. A performance model is given which is used both to predict execution time for the ICCG methods and also to compare an ICCG method to conjugate gradient without preconditioning or another ICCG method. Results are given from runs on the CYBER 205 at NASA's Langley Research Center for four model problems.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
vector computers, incomplete Cholesky, conjugate gradient, p-colored matrices, CYBER 205	Unclassified - unlimited STAR Category - 64

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	124	A06

For sale by the National Technical Information Service, Springfield, Virginia 22161

End of Document