NASA CR-166,027

NASA Contractor Report 166027

NASA-CR-166027
19830006723

ITERATIVE ALGORITHMS FOR LARGE SPARSE LINEAR SYSTEMS ON PARALLEL COMPUTERS

Loyce M. Adams

UNIVERSITY OF VIRGINIA
Department of Applied Mathematics
and Computer Science
Charlottesville, Virginia 22904

Grant NAG1-46
November 1982

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

TABLE OF CONTENTS PAGE

PAGE

LIST OF SYMBOLS

C^0	space of continuous functions
C^1	space of continuously differentiable functions
$L^2(\Omega)$	space of square integrable functions on Ω
Ω	problem domain
$\partial\Omega$	boundary of Ω
$H^1(\Omega)$	Sobolev space of order 1
$H_0^1(\Omega)$	functions in $H^1(\Omega)$ that are 0 on $\partial\Omega$
∇w	gradient of w
K	stiffness matrix
N	number of rows of K
$\underline{u}$	displacement vector
$\underline{f}$	force vector
$\underline{\sigma}$	stress vector
$\underline{\varepsilon}$	strain vector
p	processor
d	degrees of freedom
B	Jacobi iteration matrix
L_ω	SOR iteration matrix
ω	relaxation factor
κ	condition number
$\underline{r}$	residual vector
$\underline{p}$	conjugate direction vector
M	preconditioning matrix
m	number of steps for the m-step PCG method
γ	extrapolation factor

This Page Intentionally Left Blank

LIST OF ALGORITHMS

This Page Intentionally Left Blank

LIST OF FIGURES

LIST OF GRAPHS

This Page Intentionally Left Blank

LIST OF TABLES

CHAPTER 1

Introduction

The approximate solution of partial differential equations often leads to large sparse systems of linear equations that must be solved numerically. These systems can contain tens, or even hundreds of thousands of equations and require hours to solve on conventional mainframe computers such as the CDC CYBER 7600 series.

With the advent of parallel architectures found in vector computers such as the CRAY-1 and CYBER 203/205 or arrays of microprocessors found in the ICL DAP (Cryer[1981]), the HEP (Smith[1978]) and NASA Langley's Finite Element Machine (Jordan[1978]), it may be possible to solve these problems in a shorter time. Also, with the cost of hardware continually decreasing, arrays of microprocessors may prove to be a cost-effective architecture for solving these large problems, especially with the use of VLSI (Very Large Scale Integration) or WSI (Wafer Scale Integration) technology.

A major use of these equations is in structural engineering. Problems such as deflection of membranes are described by second order elliptic partial differential equations, and beam or plate bending problems are governed by fourth order elliptic equations. The usual way to solve these problems approximately is to first discretize the spatial domain by finite elements and then to solve the resulting system of linear equations by a direct solution technique, usually some variant of Cholesky decomposition (see e.g. Noor and Fulton[1974] and Reid[1980]) The linear system is often too large to fit completely in the computer's main memory,

especially after the fill-in due to the decomposition. Hence, these solution techniques must include the moving of data between main memory and the backing store. This data handling requires efficient memory management and can be very time consuming.

In this thesis, we investigate iterative algorithms for solving, on parallel computers, the large sparse symmetric and positive definite linear systems that arise from elliptic partial differential equations such as those from structural engineering. Iterative methods have the advantage that minimal storage space is required for implementation since no fill-in of the zero positions of the coefficient matrix for the system of linear equations occurs during computation. Hence if many processors with memories are connected together and the data is distributed among them, it may be possible to solve large problems without moving large amounts of data Another advantage of an iterative method is that the process may converge in very few steps if a good initial guess is known This is the case in some applications Also, for certain three dimensional elliptic problems, Fix and Larsen[1971] show that iterative methods can outperform Cholesky decomposition on sequential computers Iterative methods seemingly parallelize better than direct methods and are therefore potentially viable techniques for solving large sparse linear systems on parallel computers

The thesis consists of eight chapters. Chapter 2 reviews the finite element method, points out aspects of it that are amenable to parallel computation and derives the system of linear equations for two example problems

Chapter 3 describes in detail the architecture of NASA Langley's Finite Element Machine. This machine is used to describe the implementation of the parallel iterative algorithms.

In Chapter 4, two algorithms are developed for the parallel assembly of the system of linear equations by the finite element technique for the Finite Element Machine. These assembly algorithms are then compared and their speedups relative to a single processor version are determined. The last section of this chapter describes how to perform a stress analysis in parallel on the Finite Element Machine once the solution to the linear system for the displacements is found.

Chapter 5 describes several parallel linear stationary iterative methods that can be implemented on either vector computers or parallel arrays. The implementation of Jacobi's method is given in Section 5.1. Section 5.2 describes a new method, which we call Multi-color SOR, discusses its implementation on parallel machines, compares it to existing theory, and reports numerical comparisons to SOR without Multi-coloring. Section 5.3 describes a Multi-color SSOR method and its efficient implementation on parallel architectures. Finally, Section 5.4 describes how to implement block iterative methods such as block Jacobi and block SOR on these machines.

Chapter 6 describes parallel conjugate gradient methods. The implementation on the Finite Element Machine of the the standard conjugate gradient method is given in Section 6.1. Section 6.2.1 describes the implementation considerations for parallel preconditioned conjugate gradient methods and Section 6.2.2 lists some common preconditioners and discusses the difficulty encountered in their implementation on paral-

lel machines. In Sections 6 2 3 and 6 2.4 we give preconditioners that are suitable for parallel machines, analyze when they can be applied, and relate them to the preconditioners of Dubois, Greenbaum, and Rodrique[1979] and Johnson[1981]. Section 6 2 5 gives numerical results for the preconditioners of Sections 6.2.3 and 6.2.4 on two example problems.

In Chapter 7 we develop a detailed model for comparing parallel algorithms on an architecture like the Finite Element Machine. This model is then used to analyze the algorithms in Chapters 5 and 6 as a function of the number of processors in the microprocessor array and also as a function of the machine's ratio of communication to arithmetic time.

Chapter 8 summarizes the results of this work and describes areas for promising future research.

CHAPTER 2

The Finite Element Method

This chapter gives a brief description of the finite element method, highlights the aspects of the method that are amenable to parallel computation, and derives the finite element equations for two specific problems that are used in future chapters.

2.1. Description of the Method

The finite element method is a general technique for constructing approximate solutions to boundary value problems (Oden[1981], Strang and Fix[1973]). Suppose we want to solve the following second order nonhomogeneous elliptic partial differential equation with homogeneous Dirichlet boundary conditions:

$$\begin{aligned} -\sum_{i,j=1}^{2} \frac{\partial}{\partial x_i}(a_{ij}(x)\frac{\partial}{\partial x_j}u(x)) &= f(x) \quad x \in \Omega \\ u(x) &= 0 \quad x \in \partial\Omega \end{aligned} \tag{2.1}$$

where Ω is a bounded domain in R^2 and the matrix $a_{ij}(x)$ is symmetric and uniformly positive definite.

To approximate the solution of (2.1) by the finite element method, we first write (2.1) in its variational form:

Find $u(x) \in H_0^1(\Omega)$ such that

$$a(u,v) = (f,v) \quad v \in H_0^1(\Omega) \tag{2.2}$$

where

$$(w,v) = \int_{\Omega} w(x)v(x)dx$$

and

$$a(u,v) = \int_{\Omega} \sum_{i,j=1}^{2} a_{ij}(x)\frac{\partial}{\partial x_j}u(x)\ \frac{\partial}{\partial x_i}v(x)\ dx$$

Here H_o^1 denotes the set of all functions $v \in H^1(\Omega)$ such that $v=0$ on $\partial\Omega$ where $H^1(\Omega)$ is the set of all functions which together with their partial distributional derivatives belong to $L^2(\Omega)$, the space of square integrable functions on Ω.

If we choose the matrix a_{ij} in (2.1) to be the identity matrix, we get Poisson's equation.

$$\begin{aligned} -(u_{xx} + u_{yy}) &= f(x,y) \qquad (x,y) \in \Omega \\ u(x,y) &= 0 \qquad\quad (x,y) \in \partial\Omega \end{aligned} \tag{2.3}$$

Likewise, the associated weak form of Poisson's equation can be obtained from (2.2).

Find $u(x,y) \in H_o^1(\Omega)$ *such that*

$$\int_{\Omega} \nabla u(x,y) \cdot \nabla v(x,y)dxdy = \int_{\Omega} f(x,y)v(x,y)dxdy \tag{2.4}$$

for all

$$v(x,y) \in H_o^1(\Omega)$$

where ∇w is the gradient of w.

We now consider the approximation of the solution of (2.4) by the finite element method. A finite dimensional subspace $V^h \subset H_o^1(\Omega)$ must first be chosen. This subspace typically consists of piecewise polynomials defined over a triangulation of Ω, called Ω^h, where each triangle is called a finite element If the subspace V^h is spanned by the functions

φ_j, $j=1,2,\ldots,n$, called basis functions, we look for an approximate solution $u^h(x,y)$ of the form

$$u^h(x,y) = \sum_{j=1}^{n} \alpha_j \varphi_j(x,y) \tag{2.6}$$

The substitution of (2.6) into (2.4) yields

$$\int_{\Omega^h} \nabla u^h(x,y) \cdot \nabla v^h(x,y) dxdy = \int_{\Omega^h} f(x,y) v^h(x,y) dxdy \tag{2.7}$$

for *all* $v^h \in V^h$.

By choosing $v^h = \varphi_i$, $i=1,2,\ldots,n$ in (2.7), we get the following symmetric and positive definite system of linear equations for $\underline{\alpha}$.

$$K\underline{\alpha} = \underline{f} \tag{2.8}$$

where

$$k_{ij} = \int_{\Omega^h} \nabla \varphi_i(x,y) \cdot \nabla \varphi_j(x,y) \; dxdy \tag{2.9}$$

and

$$f_i = \int_{\Omega^h} f(x,y) \varphi_i(x,y) \; dxdy$$

The power of the finite element method lies in the choice of the basis functions φ_j. A linear polynomial is uniquely determined by its values at the three vertices of any triangle. Suppose φ_j is chosen to be that piecewise linear polynomial which has the value 1 at vertex V_j of the triangulation and has the value zero at the rest of the vertices of the triangulation. Then φ_j is a continuous function which belongs to $H^1(\Omega)$ and is nonzero only on those triangles having V_j as a common vertex. In addition, any C^0-piecewise linear polynomial may be represented as a

linear combination of the φ_i's. Functions belonging to $H_0^1(\Omega^h)$ are obtained by omitting those φ_i's that are defined to have the value 1 at the V_i's on the boundary of Ω^h. The points of Ω^h for which φ_i is defined to be either 1 or 0 are often referred to as nodes. If the φ_i's are linear as described above, the vertices V_i will be the nodes.

Alternatively, a quadratic polynomial is uniquely determined by its values at the vertices and at the midpoints of the sides of the triangle. Likewise, values given at ten nodes located two per side of the triangle, one at each vertex, and one in the center of the triangle uniquely determine a cubic polynomial.

By choosing the φ_i's to be piecewise polynomials with the value of either 1 or 0 at the points of the triangulation, the value of α_i in equation (2.6) will be the value of $u^h(x,y)$ at the nodal point i, denoted δ_i, and we can write equation (2.8) as

$$K\underline{\delta} = \underline{f} \tag{2.10}$$

The matrix K will be sparse due to the choice of the basis functions since the values of k_{ij} will be zero if nodes i and j are not on a common finite element. In other words, row i will have at most as many nonzero off diagonal entries as node i has neighbor nodes (two nodes are called neighbor nodes if they share a common finite element). To illustrate this, Figure 1 shows a region discretized by triangular finite elements.

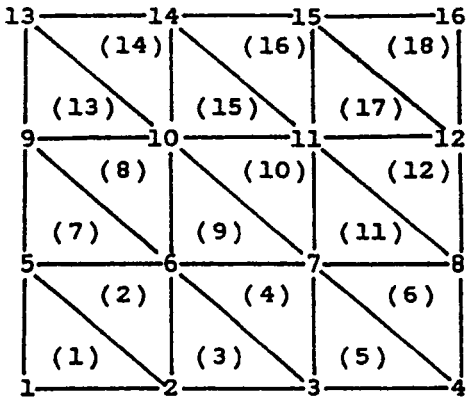


Figure 1. Region Discretization
18 elements; 16 nodes

Now, if linear piecewise polynomials are chosen for the basis functions, row 6 of K will have at most 6 off-diagonal entries; namely, k_{62}, k_{63}, k_{65}, k_{67}, k_{69}, and $k_{6,10}$. This sparsity will be a major consideration in the design of parallel algorithms for solving $K\underline{\delta}=\underline{f}$ and will be addressed in Chapters 5 and 6.

Each entry, k_{ij}, as defined by equation (2.9) is obtained by an integration over the domain Ω^h. Since φ_i and φ_j in (2.9) are both nonzero only on finite elements that contain both nodes i and j, this integration is only performed over these particular elements. Figure 1 shows that the integration is performed over six elements to calculate k_{ij} if $i=j$ and two elements otherwise. As an example, suppose $i=6$ and $j=7$ Then, from Figure 1 we obtain

$$k_{67} = \int_{(4)} \nabla\varphi_6 \cdot \nabla\varphi_7 + \int_{(9)} \nabla\varphi_6 \cdot \nabla\varphi_7 \tag{2.11}$$

Likewise, if $i=6$ and $j=6$ we get

$$k_{66} = \int_{(2)} \nabla\varphi_6 \cdot \nabla\varphi_6 + \int_{(3)} \nabla\varphi_6 \cdot \nabla\varphi_6 + \int_{(4)} \nabla\varphi_6 \cdot \nabla\varphi_6 + \int_{(7)} \nabla\varphi_6 \cdot \nabla\varphi_6 + \int_{(8)} \nabla\varphi_6 \cdot \nabla\varphi_6 + \int_{(9)} \nabla\varphi_6 \cdot \nabla\varphi_6 \tag{2.12}$$

Lastly, if $i=6$ and $j=8$ then

$$k_{68} = 0 \tag{2.13}$$

since nodes 6 and 8 have no finite element in common. These observations suggest a commonly used three step procedure for assembling the K matrix:

(1) Zero out the storage that is to be used for K. This will usually be a symmetric storage structure in which the diagonal and upper bands of K are stored.

(2) Integrate over each element, one at a time, to calculate the element's contribution to the diagonal and upper bands of K. For example, the integrations over element (1) in Figure 1 yield the following contributions to K:

$$k_{11} \int_{(1)} \nabla\varphi_1 \cdot \nabla\varphi_1 \quad k_{12} \cdot \int_{(1)} \nabla\varphi_1 \cdot \nabla\varphi_2$$
$$k_{22} \cdot \int_{(1)} \nabla\varphi_2 \cdot \nabla\varphi_2 \quad k_{15} \int_{(1)} \nabla\varphi_1 \cdot \nabla\varphi_5$$
$$k_{55} \cdot \int_{(1)} \nabla\varphi_5 \cdot \nabla\varphi_5 \quad k_{25} \int_{(1)} \nabla\varphi_2 \cdot \nabla\varphi_5$$

These values comprise what is commonly called the element matrix for element (1) and can be represented by

$$K^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 5 \end{array} \\ \begin{bmatrix} k_{11} & k_{12} & k_{15} \\ & k_{22} & k_{25} \\ & & k_{55} \end{bmatrix} \end{array}$$

(3) The values in each element matrix are added to the appropriate position in the global K matrix.

This procedure can be adapted for a parallel computer with little modification since the integrations over two different elements can be done simultaneously. This is the topic of Chapter 4.

Thus far only the solution of a scalar partial differential equation by the finite element method has been considered; however, the method can be applied to a system of equations as well. In particular, the equations that govern the static displacement of a body in plane stress will be a coupled sytem of two equations for the displacements of the body in the x and y directions respectively. The finite element method as applied to this problem will be described in the following section

2.2. Plane Stress Equations

The procedure for constructing the stiffness matrix K for a plane stress analysis of an isotropic linear elastic body Ω will be described in this section Similar descriptions may be found in Oden[1981], Norrie and deVries[1978], and Zienkiewicz[1971]. The problem is to find the displacements in both the x and y coordinate directions of a 2-dimensional body Ω that is in plane stress, such as the membrane shown in Figure 2.

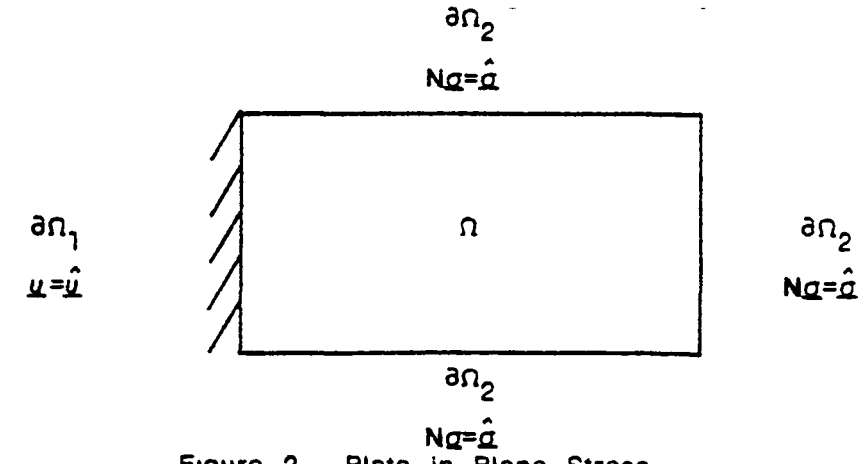


Figure 2. Plate in Plane Stress

First, we introduce the notation that will be used in this discussion

$$\underline{\sigma}(x,y) = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} \qquad \textit{stress vector}$$

$$\underline{\epsilon}(x,y) = \begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{12} \end{bmatrix} \qquad \textit{strain vector}$$

$$\underline{u}(x,y) = \begin{bmatrix} u \\ v \end{bmatrix} \qquad \textit{displacement vector}$$

$$D = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

$$E = \frac{E}{1-\nu} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu^2}{2} \end{bmatrix}$$

E=Young's modulus

ν=Poisson's ratio

$$\mathbf{N} = \begin{bmatrix} n_x & 0 & n_y \\ 0 & n_y & n_x \end{bmatrix} \qquad n_x, n_y \text{ normal components } \partial\Omega_2$$

$$\underline{f} = \begin{bmatrix} f_x \\ f_y \end{bmatrix} \qquad \text{body forces per unit area}$$

$$\hat{\underline{\sigma}} = \begin{bmatrix} \hat{\sigma}_x \\ \hat{\sigma}_y \end{bmatrix} \qquad \text{surface tractions applied on } \partial\Omega_2$$

$$\hat{\underline{u}} = \begin{bmatrix} \hat{u} \\ \hat{v} \end{bmatrix} \qquad \text{displacements on } \partial\Omega_1$$

The conservation of linear momentum for the body Ω states that any portion ω of the body must be in static equilibrium:

$$\int_{\omega} (\mathbf{D}^T \underline{\sigma} + \underline{f}) dx dy = \underline{0} \tag{2.14}$$

For sufficiently smooth $\underline{\sigma}$ and $\underline{f}$ we get the partial differential equations of equilibrium for the body Ω.

$$\mathbf{D}^T \underline{\sigma} + \underline{f} = \underline{0} \tag{2 15}$$

A material that is linearly elastic, homogenous, and isotropic satisfies the constitutive equation which relates the stresses to the strains

$$\underline{\sigma} = \mathbf{E}\underline{\epsilon} \tag{2 16}$$

The strains and displacements are related by

$$\underline{\epsilon} = \mathbf{D}\underline{u} \tag{2 17}$$

The substitution of (2.16) and (2 17) into (2 15) yields the partial differential equations in terms of the displacements only

$$\mathbf{D}^T \mathbf{E} \mathbf{D} \underline{u}(x,y) + \underline{f}(x,y) = \underline{0} \qquad (x,y) \in \Omega \tag{2 18}$$

The boundary conditions for (2.18) are.

$$\underline{u}(s) = \hat{\underline{u}}(s) \qquad s \in \partial\Omega_1 \tag{2.19a}$$

$$\mathbf{N}\underline{\sigma}=\hat{\underline{\sigma}} \text{ or } \mathbf{NED}\underline{u}=\hat{\underline{\sigma}}(s) \qquad s \in \partial\Omega_2 \tag{2.19b}$$

The variational form of the problem in (2.18), (2.19a), and (2.19b) is easily found to be

$$\int_{\Omega}(\mathbf{D}\underline{v})^T\mathbf{ED}\underline{u}\,dxdy = \int_{\Omega}\underline{v}^T\underline{f}\,dxdy + \int_{\partial\Omega_2}\underline{v}^T\hat{\underline{\sigma}}\,ds \tag{2.20}$$

for all $v_i \in H^1(\Omega)$, $u_i \in H^1(\Omega)$, i=1,2 and $\underline{u}=\hat{\underline{u}}$ on $\partial\Omega_1$ and $\underline{v}=\underline{0}$ on $\partial\Omega_1$.

Before we introduce the finite element approximation to (2.20) an alternate derivation of (2.20) will be given. This derivation comes from the minimization of the potential energy of the body. The potential energy functional $X(\underline{w})$ is given by

$$X(\underline{w}) = \frac{1}{2}\int_{\Omega}(\mathbf{D}\underline{w})^T\mathbf{E}(\mathbf{D}\underline{w})\,dxdy - \int_{\Omega}\underline{w}^T\underline{f}\,dxdy - \int_{\partial\Omega_2}\underline{w}^T\hat{\underline{\sigma}}\,ds \tag{2.21}$$

which is to be minimized among all

$$\underline{w} \in H^1_E = \{\ \underline{w} \mid w_1 \in H^1(\Omega), \underline{w}=\hat{\underline{u}} \text{ on } \partial\Omega_1.$$

To find the value of $\underline{u} \in H^1_E$ that minimizes X, the first variation $\delta(X(\underline{u},\underline{v}))$ is formed and set to zero.

$$\delta(X(\underline{u};\underline{v}))=\frac{1}{2}\int_{\Omega}(\mathbf{D}\underline{u})^T\mathbf{E}(\mathbf{D}\underline{v})\,dxdy+\frac{1}{2}\int_{\Omega}(\mathbf{D}\underline{v})^T\mathbf{ED}\underline{u}\,dxdy-\int_{\Omega}\underline{v}^T\underline{f}\,dxdy$$
$$-\int_{\partial\Omega_2}\underline{v}^T\hat{\underline{\sigma}}\,ds = 0 \text{ , for all } v \in H^1_{E_o} \tag{2.22}$$

where $H^1_{E_o}=\{\ \underline{v} \mid v_i \in H^1(\Omega)\ \ \underline{v}=0 \text{ on } \partial\Omega_1\ \}$

The boundary conditions (2.19b) are natural boundary conditions and do

not need to be explicitly imposed. They will automatically be satisfied by the minimizer of (2.21). Now, since $E=E^T$, (2.22) becomes (2.20).

The solution of (2.20) is approximated by the same technique described in the last section for Poisson's equation, that is, we choose a finite dimensional subspace $V^h \subset H^1(\Omega)$ and look for the approximation $\underline{u}^h(x,y)$ of the following form:

$$\underline{u}^h(x,y) = \Phi\underline{\delta} \tag{2.23}$$

where Φ is a 2x2n matrix of piecewise continuous basis functions and $\underline{\delta}$ is a $2n \times 1$ vector containing the unknown values of the components of $\underline{u}^h(x,y)$ at the n nodal points. In particular for linear basis functions, if we just consider the value of $\underline{u}^h(x,y)$ where (x,y) is a point inside a triangular finite element (e) with nodes 1,2, and 3, (2.20) becomes

$$\underline{u}^{h(e)}(x,y) = \Phi^{(e)}\underline{\delta}^{(e)} \tag{2.24}$$

where

$$\underline{u}^{h(e)} = \begin{bmatrix} u^{h(e)} \\ v^{h(e)} \end{bmatrix} \quad \Phi^{(e)} = \begin{bmatrix} \varphi_1 & 0 & \varphi_2 & 0 & \varphi_3 & 0 \\ 0 & \varphi_1 & 0 & \varphi_2 & 0 & \varphi_3 \end{bmatrix}$$

and

$$\underline{\delta}^{(e)} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix}$$

The φ_i's in (2.24) will be piecewise linear polynominals defined on triangular finite elements as in (2.9). Note that the same basis functions are used for both components of $\underline{u}^h$ in (2.24) since both components

are elements of the same subspace V^h.

The substitution of (2 24) into (2 20) yields the following element matrix and vectors:

$$\begin{aligned} K^{(e)} &= \int_{\Omega_e} (D\Phi^e)^T E D\Phi^e dxdy \\ \underline{f}^{(e)} &= \int_{\Omega_e} (\Phi^e)^T \underline{f} dxdy \\ \underline{a}^{(e)} &= \int_{\partial\Omega_2} (\Phi_e)^T N\sigma ds \end{aligned} \tag{2 25}$$

The equations that govern the solution at a particular node I are the global equations

$$K\underline{\delta} = \underline{f} + \underline{a} \tag{2.26}$$

and must be assembled from (2 25) by adding the contributions over each element to the appropriate position in K as was described in the last section.

Suppose that once the displacements $\underline{\delta}$ are found, the stresses $\underline{\sigma}$ are to be calculated. From (2 16), (2.17), and (2.23), the stress vector $\underline{\sigma}$ can be expressed as

$$\underline{\sigma} = E D\Phi\underline{\delta} \tag{2 27}$$

Since the elements of Φ are linear, $D\Phi$ will be a matrix of constants This implies that the stresses will be constant over a given element. This can be seen since on a particular element e, Φ will contain only three nonzero basis functions in each row that are associated with the three nodes of (e) Therefore, we may also write

$$\underline{\sigma}^{(e)} = E D\Phi^{(e)} \underline{\delta}^{(e)} \tag{2 28}$$

where $\Phi^{(e)}$ is a 2x6 matrix and $\underline{\delta}^{(e)}$ is a 6x1 vector

Hence, once $\underline{\delta}^{(e)}$ is found the solution of $\underline{\sigma}^{(e)}$ only requires one matrix multiplication. In fact, this stress matrix $E D \Phi^{(e)}$ can be saved from the calculation of $K^{(e)}$ in (2.25).

As illustrated above, the calculation of $\underline{\sigma}^{(e)}$ requires only the values of $\underline{\delta}^{(e)}$ and $E D \Phi^{(e)}$. This suggests a parallel implementation of the stress calculation by elements is possible. A more detailed explanation of this is given in Chapter 4 for linear basis functions defined on a triangulation of a plate under plane stress

CHAPTER 3

The Finite Element Machine

This chapter gives a brief summary of parallel architectures that have been built or are under development and then describes in detail the architecture of the Finite Element Machine.

3.1. Review of Parallel Architectures

For our purposes, it is convenient to classify parallel architectures into two categories, namely, vector computers and array computers. A vector computer will be considered to be a computer that has special hardware instructions which accept vectors as operands. These instructions may be implemented via hardware pipelines as was the case for the TI-ASC and the CDC STAR 100 in the early 1970's and currently for the CDC CYBER 203 and 205 and the CRAY-1 Alternatively, the elements of the vector may be loaded into separate processing elements as is the case for the ILLIAC-IV (The ILLIAC-IV may also be considered to be an array computer). A description of the architecture of these machines as well as the state of the art in 1977 of algorithms for solving partial differential equations on them is given in Ortega and Voigt[1977].

An array computer consists of an array of processing elements each of which may execute instructions. These elements may be simple chips to perform specific functions or they may be complete processors

Array computers fall into two classes depending on the manner in which they execute instructions (Flynn[1976]) In the first class, the pro-

cessing elements either all execute the same instruction or no instruction on different data. This class is called SIMD (single instruction, multiple data). The ILLIAC IV is an example of a SIMD machine Alternatively, the processors may execute different instructions in an asynchronous fashion on different data. This class is called MIMD (multiple instruction, multiple data).

During the course of a computation these processing elements must communicate with each other. Since providing a communication link from a processor to every other processor becomes prohibitive as the number of processors increase, a particular connection strategy also influences how we will further classify array architectures. Three major strategies as summarized by Ortega and Voigt[1983] are listed below

(1) P processors are arranged in the form of a regular lattice. Each processor has its own local memory and is permanently connected to a small subset of other processors, called neighbors.

(2) P processors with local memory are connected to each other by a bus.

(3) P processors and M memories are connected by an an electronic switch so that every processor has access to a subset, possibly all, of the memories.

Lattice arrays include the ILLIAC-IV, the Distributed Array Processor (DAP) of International Computers Limited (Cryer[1981]), and the Systolic Arrays of Kung[1980] The ILLIAC-IV and the DAP are SIMD machines

The Systolic Arrays are special purpose computers on a chip with a group of processing elements working in a SIMD fashion to produce output for other groups of processing elements. Hence, the idea is to have simple and cheap processors that calculate and transmit data in a regular fashion for a particular application.

Examples of two MIMD bus arrays are the CM* at Carnegie Mellon University (Swan[1977]) and ZMOB under development at the University of Maryland (Rieger,Trigg, and Bane[1981]).

Examples of two MIMD switched arrays are the C.mmp of Carnegie Mellon University (Fuller and Harbison[1978]) and the Hetrogeneous Element Processor (HEP) being implemented by Denelcor, Inc. (Smith[1978]).

3.2. The Finite Element Machine Architecture

Of particular interest to us is the architecture of the Finite Element Machine (FEM) at NASA Langley Research Center. The FEM is an array of microprocessors that can operate asynchronously, and can be classified as an MIMD computer that is arranged in a square lattice configuration with dedicated local communication links between any processor and its eight nearest neighbors. However, it is not strictly a lattice type array since a bus connects all the processors.

To summarize the motivation for FEM, consider the structure in Figure 1.

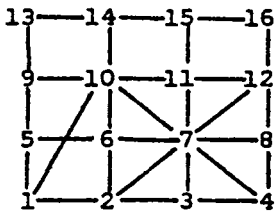


Figure 1. Example Structure

This structure is composed of simple rod elements of two nodes each. Two nodes are said to be connected if they are on the same rod. For example, node 1 is connected to nodes 2, 5, and 10 As was seen in Chapter 2, the finite element method produces a block stiffness matrix that has as many nonzero off-diagonal blocks in row i as node i has connected nodes. If an iterative method is used to solve the equations, the solution at node i is only a function of the information from node i's connected nodes. This suggests assigning a processor to each node and providing it with local communication links to processors containing connected nodes. For FEM, the idea was to provide a fixed number of local links per processor and to provide a global bus to handle the connectivities that are not satisfied locally. For example, if eight nearest neighbor links are available, processor 7 can operate with local links only, but processor 1 must communicate globally with processor 10

A detailed description of the machine is given in the following paragraphs. Most of this material can be found in Jordan, et al[1979], Jordan[1978], and [1979], Jordan and Sawyer[1978], and Podsiadlo and Jordan[1981]. Additional hardware information was obtained through personal discussions with Tom Crockett, Judson Knott, and David Loendorf at NASA A current status report on the hardware, system software, and

application software can be found in Storsaali, Peebles, Crockett, Knott, and Adams[1982].

The FEM architecture as shown in Figures 2 and 3, consists of a controller and individual microprocessors, which we will call nodal processors. The controller is connected to the nodal processors via a global bus. Each nodal processor of the FEM is connected to its eight nearest neighbors via the two-way local communication links. These local links also connect the edges of the FEM in a toroidal wrap around fashion so that every processor has eight nearest neighbors Each processor can also communicate globally to its non-nearest neighbor processors by using the serial global bus. Each microprocessor has its own operating system and memory. Once operation begins, each processor executes its program independent of the controller. This architecture can be classified as an MIMD (multiple instruction, multiple data) machine with no central shared memory.

3.2.1. Controller Hardware

The controller is a Texas Instruments 990/10 mini-computer. It has 128K words of memory and four 5-megabtye disk drives. A printer is the output device for printing files from the disk of the TI 990 That is, output from the nodal processors is transferred over the global bus and placed on a file on the 990 disk, for printing when needed.

The DX-10 full screen editor on the 990 is used by the applications programmer to edit, compile, and link the programs that are to be run on FEM The necessary data files for each processor can be created by the editor and stored on the 990 disk Alternatively, for complicated problem geometry, a program could be written to split a global data file

into appropriate data files for each processor. Nevertheless, these data files and the linked program file are stored on the 990 disk until time for downloading to FEM. This downloading is accomplished by executing commands on the controller

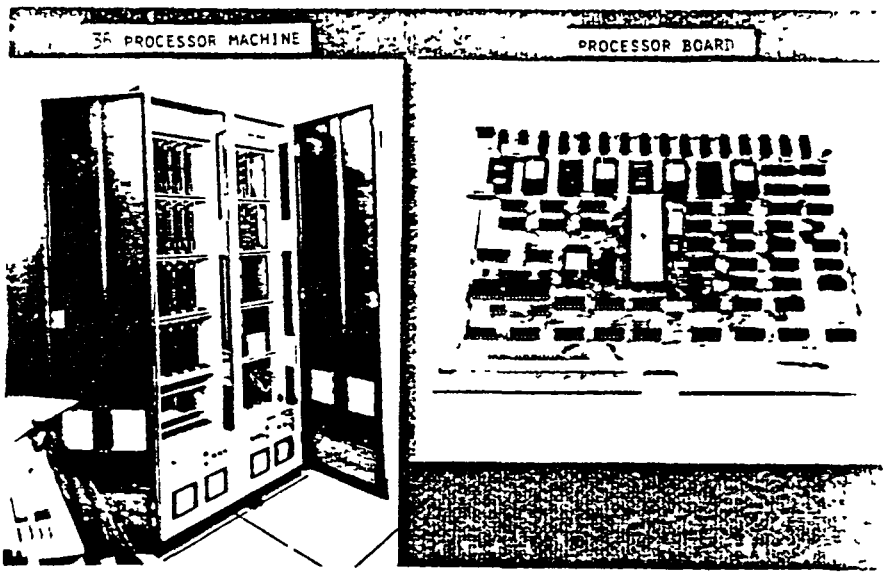
3.2.2. Nodal Processor Hardware

The nodal processors are comprised of three hardware boards each, namely, the CPU board, the IO-1 board, and the IO-2 board. These components will be described in the following sections.

3.2.2.1. CPU Board

The CPU board contains a TI 9900 16 bit microprocessor, 16K bytes of read only memory (ROM), 32K bytes of random access memory (RAM), and an Advanced Micro Devices AM9512 floating point chip. An illustration of a nodal processor is given in Figure 4.

The ROM is reserved for NODAL EXEC, the nodal executive operating system, and PASLIB, the PASCAL subroutine library that contains various basic routines such as SEND and RECEIVE for transmitting data. Approximately 4K of RAM is also reserved for NODAL EXEC. The remaining 28K RAM is available for program code, run time data structures, and special user allocated data storage, called data areas. The user may request as many as 32 separate data areas of convenient sizes for a particular application program These areas will be used to store the problem data that is downloaded from the 990 disk. In practice, the program code is loaded at address 8000 hexadecimal toward the bottom of the stack and the data areas are loaded directly above the NODAL EXEC towards the top of the stack as depicted below.



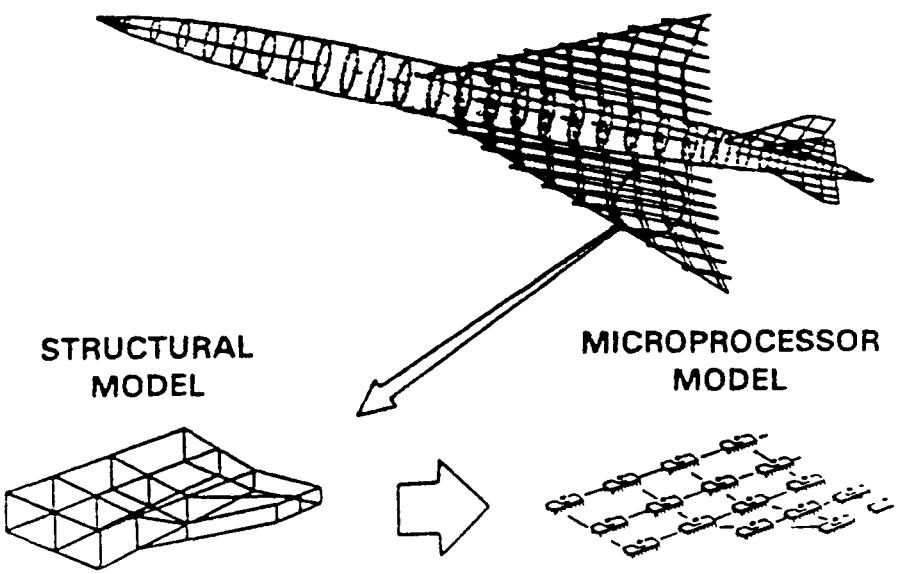


FIGURE 2. THE FINITE ELEMENT MACHINE

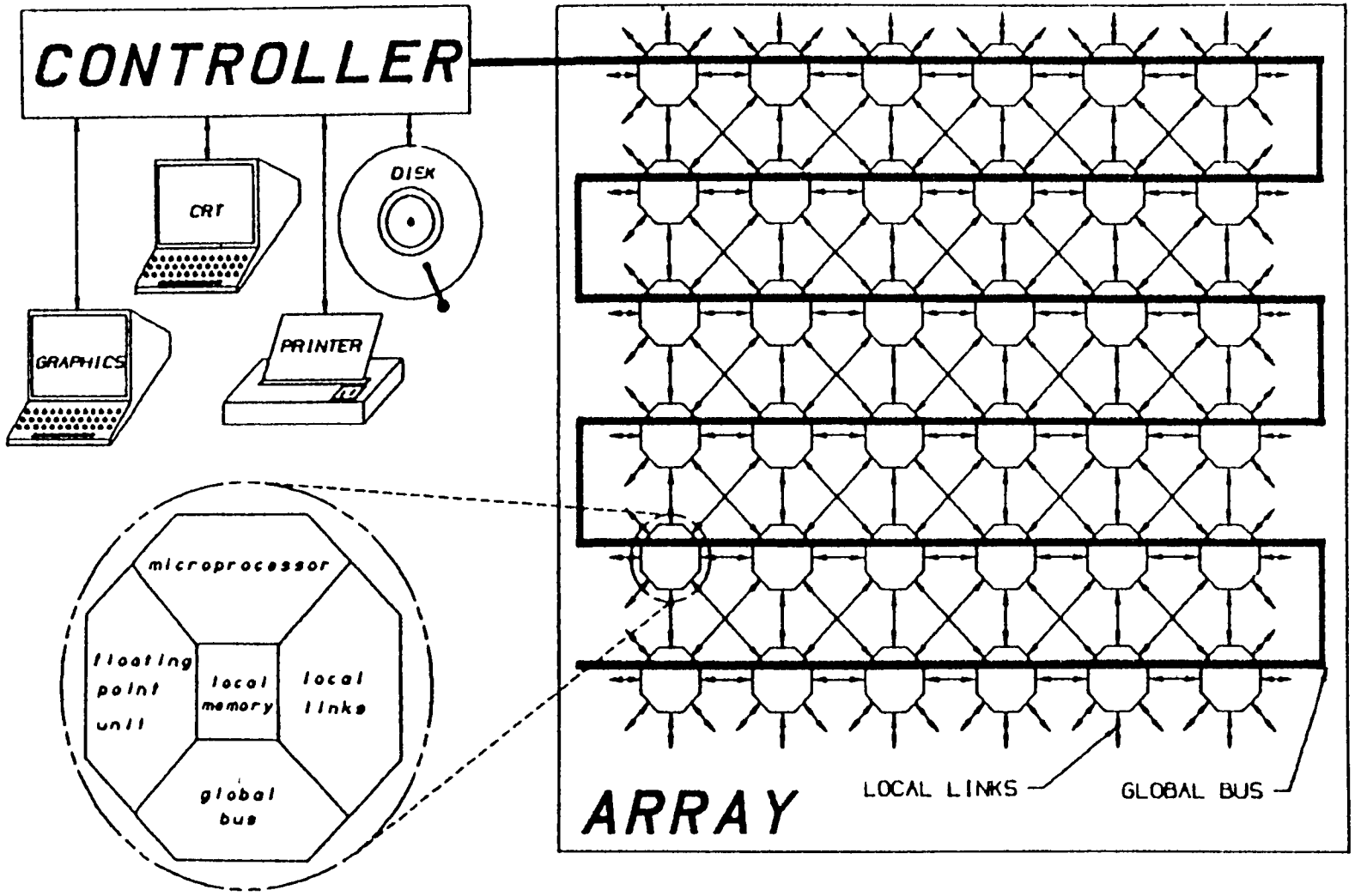


FIGURE 3. THE FINITE ELEMENT MACHINE ARCHITECTURE

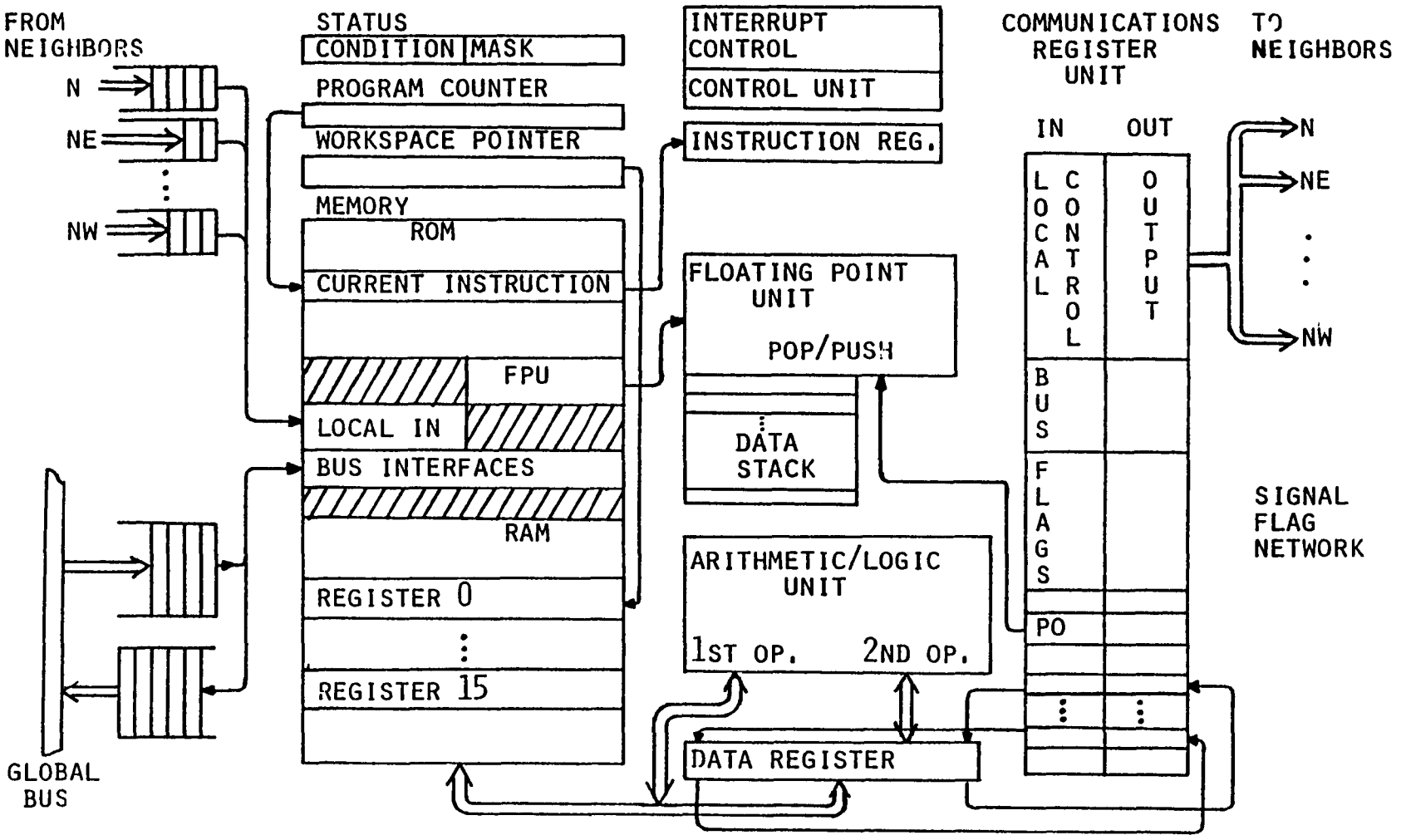
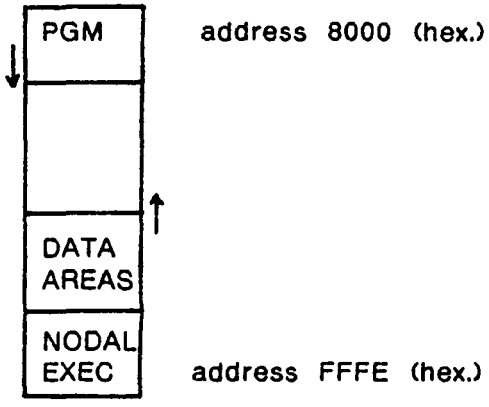


FIGURE 4. THE FEM NODAL PROCESSOR



The AM9512 floating point chip with a clock frequency of 2 MHz provides single precision (32-bit, 25 bit mantissa) and double precision (64-bit, 57 bit mantissa) add, subtract, multiply and divide operations. To use this capability, the operands must be loaded by software which requires approximately 358 microseconds for two single precision numbers. Once the operands are loaded, a single precision floating point add or subtract can be performed in approximately 29 microseconds, a single precision floating point multiply in approximately 99 microseconds, and a single precision floating point divide in approximately 114 microseconds. These times for loading the operands were provided by Tom Crockett at NASA Langley Research Center and the arithmetic times can be found in the Advanced Micro Devices Manual[1979].

3.2.2.2. IO-1 Board

The hardware circuits for the local communication links and the sum/max network are on the IO-1 board. As stated earlier, each processor is connected locally to its eight nearest neighbors Each proces-

sor has 12 8-bit by 32-bit FIFO hardware queues for receiving values from its neighbors. Likewise, an output register is available for sending values to neighbors. Software queues for synchronous and asynchronous data receiving are also implemented. An illustration of the local communication links is given in Figure 5.

A separate hardware circuitry was designed by Jordan for calculating maximums and sums across processors (Jordan, Scalabrin, and Calvert[1979]). The sum/max hardware can be envisioned as a binary tree with each processor initially at the leaves of the tree. The values from pairs of processors are added and passed to the next level in the tree. The procedure is repeated until the final result is obtained. This allows a sum to be calculated in $\log_2 p$ time where p is the total number of processors. An algorithm for finding the maximum value when only local links and a global bus are available can be found in Bokhari[1979].

3.2.2.3. IO-2 Board

The hardware for the global bus connections, a signal flag network, to be described below, and the processor's self identification tag is resident on the IO-2 board.

In addition to serving as a connection between the controller and FEM, the global bus connects each processor to every other processor. The bus is 16 bits wide; therefore, one single precision number requires two transmissions. Information to be sent on the bus is tagged with a source, destination, and mode tag. The mode indicates whether the data is to be broadcast to all processors or if it is to go to the destination processor. Since contention is likely to occur for the bus, the outgoing data is buffered Input from the bus is detected by the address detector

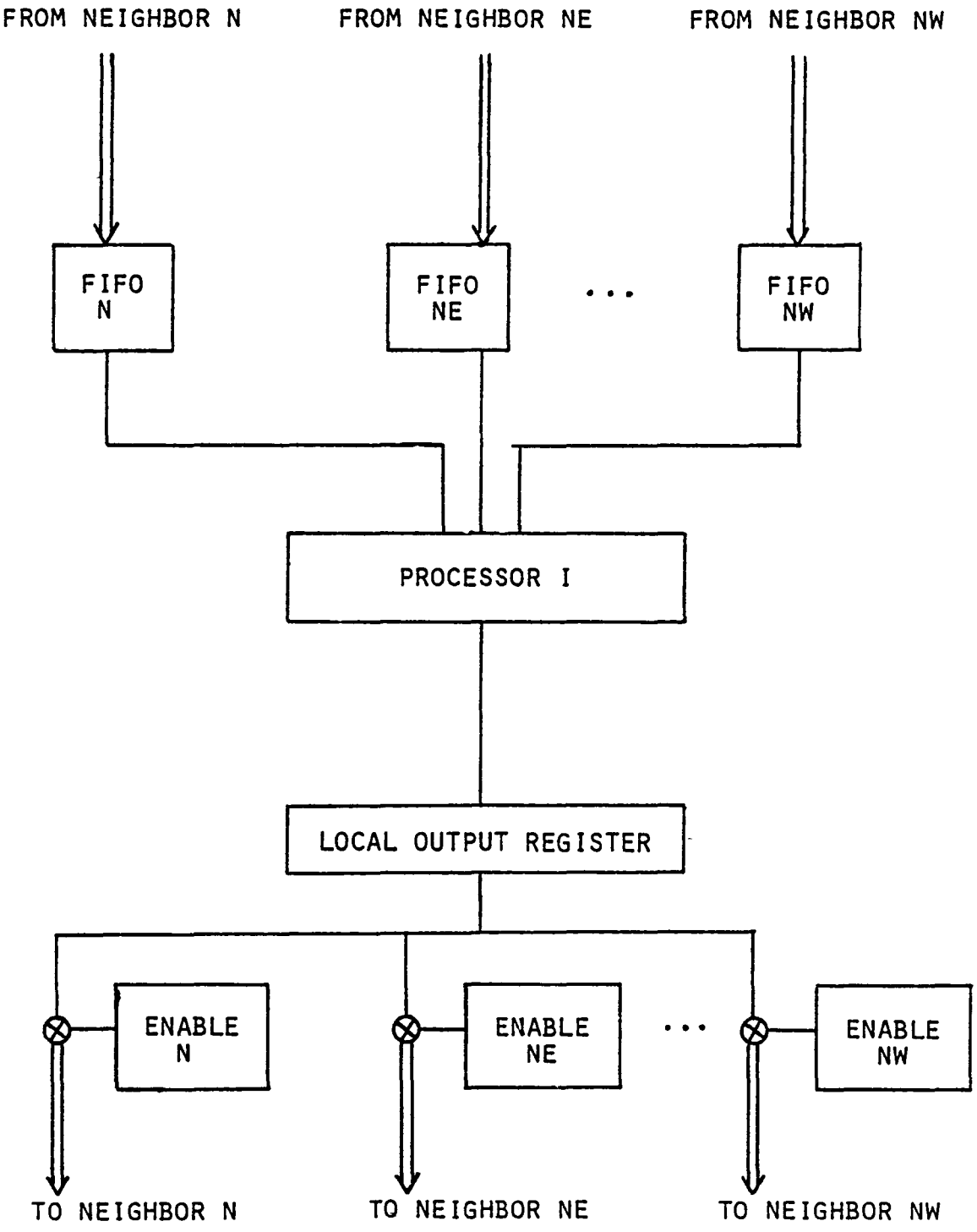


FIGURE 5. THE FEM LOCAL COMMUNICATION LINKS

and queued in an imput buffer. An overview of the global bus is given in Figure 6.

Each processor is part of eight separate signal flag networks, flags 0 through 7, which can be used for synchronization or decision making. Each flag can be enabled into or disabled from its network by a PASLIB routine. Data area 0 contains a list of processors, both local and global, among which information must be shared during program execution. These processors will be called logically connected processors. The major function of the flags is to answer the following questions.

```
Any(k)?  Is Flag k set in any enabled logically
         connected processor?

All(k)?  Is Flag k set in all enabled logically
         connected processors?

Sync(k)? Was All(k) true previously?
```

Flag 0 has a FIRST bit. This is used to determine if this processor was the first one to set the flag. An illustration of the signal flag network is given in Figure 7

The processor's self identification number is hardwired on the IO-2 board. A PASLIB routine is used to return the value of the self-id as needed during program execution. Typical instances in which the self-id is necessary are decision making during computation and interprocessor communication.

Presently at NASA. 4 processors are operational with 16 scheduled for the immediate future and 36 scheduled for December 1982. The initial design was for 1024 processors configured in a 32 by 32 array.

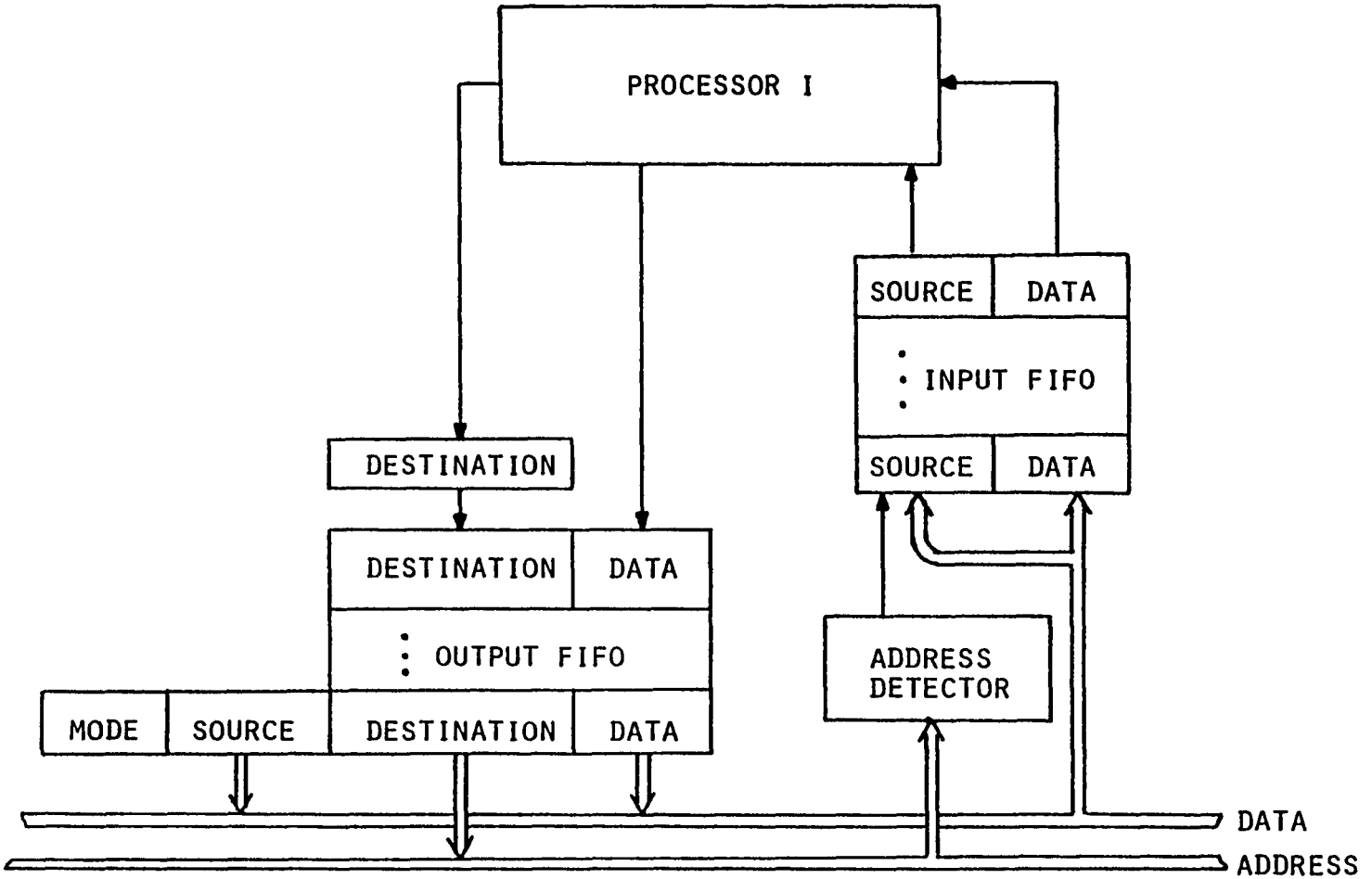


FIGURE 6. THE FEM GLOBAL BUS

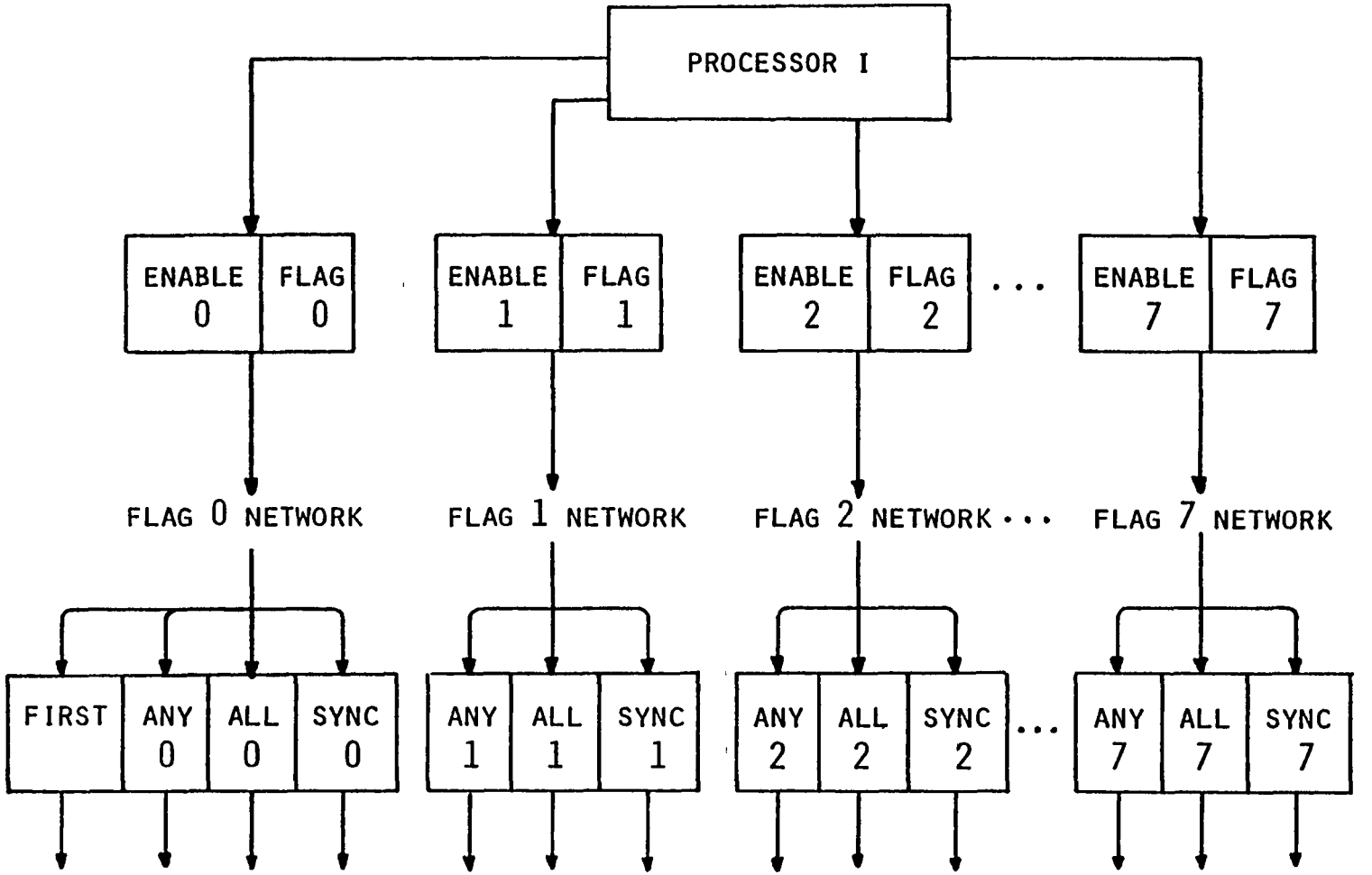


FIGURE 7. THE FEM SIGNAL FLAG NETWORK

CHAPTER 4

Parallel Matrix Assembly and Stress Calculation

4.1. Parallel Matrix Assembly

This section describes how to assemble in parallel the nonzero coefficients of the stiffness matrix K, as described in Chapter 2, on an array computer like the Finite Element Machine. A description and analysis of the assembly process with and without communication between processors will be given.

Figure 1 shows a region that is discretized by eight triangular finite elements which are comprised of three nodes each.

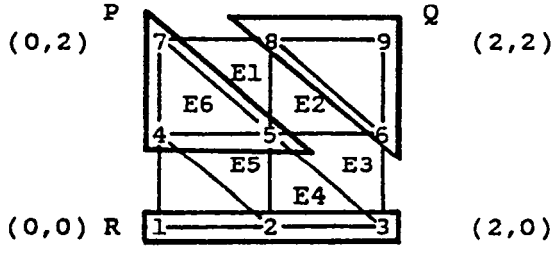


Figure 1. Region Discretized by Finite Elements

If there are d unknowns at each of the nine nodes, the resulting stiffness matrix K will have dimension $9d \times 9d$. These nine nodes (and associated data) are partitioned to the three processors (P,Q,R) so that during the solution of $K\underline{u}=\underline{f}$ each processor will calculate exactly $3d$ unknowns.

The coefficients of K that are required by processor P for the solution of the unknowns at nodes 4,5, and 7 must either be calculated by

P or calculated by processors R or Q and communicated to P before the displacement calculation begins. In both cases, storage must be allocated in P's memory for these coefficients. The amount of storage depends on the number of nodes assigned to P, the number of equations at each node, and the number of nodes that share a common finite element with P's nodes. In particular, for *d* equations at each node, a *dxd* coefficient matrix must be calculated for every pair of nodes on the same finite element if at least one of these nodes is assigned to processor P. In addition, one *dxd* symmetric matrix must be found for each node that is assigned to P. The matrices that must be stored in P's memory for the region in Figure 1 are indicated in Table 1. The nodes (4,5,7) are labeled Interior and the nodes in other processors that share a common finite element with any of these nodes are labelled Exterior and [x] represents a *dxd* matrix.

	Interior			Exterior				
	4	5	7	1	2	3	6	8
4	[x]	[x]	[x]	[x]	[x]			
5	[x]	[x]	[x]		[x]	[x]	[x]	[x]
7	[x]	[x]	[x]					[x]

Table 1. *K* Matrix Coefficients for Processor P

For an iterative solution of $K\underline{u}=\underline{f}$, Table 1 contains all the coefficients needed for processor P to solve for the displacements at nodes 4, 5, and 7. If a direct method such as Cholesky factorization were used instead, extra storage must be allocated for the coefficients that will "fill in" the band of the upper triangular factor of *K* during the decomposition process For Figure 1, space must be reserved in processor P for the 4-3, 4-6, 7-6, and 7-9 *dxd* fill in matrices. In particular, space

must be allocated in Table 1. for the fill in coefficients 4-3, 4-6, and 7-6

We now describe how to assemble the coefficients in Table 1 by considering node 5. To find the [x] coefficients in the second row and second column of Table 1 (denoted by 5-5), the coordinates of nodes 2,3,6, and 8 in addition to those of nodes 4 and 7 must be available to processor P since node 5 is on elements E1, E2, E3, E4, E5, and E6 as shown in Figure 1. Hence, the 5-5 [x] coefficients can not be found without coordinate information that resides in other processors In fact, the same conclusion holds for every [x] in Table 1. This observation leads us to consider the following strategy:

(1) Load each processor's memory at the outset with all problem data necessary for the calculation of the coefficients that are required in the solution of the displacement equations at its collection of nodes.

(2) Implement one of the following two policies.

Policy 1 Each processor will calculate the upper triangular and diagonal coefficients associated with its collection of nodes as well as the coefficients associated with the connection between the processor's interior and exterior nodes. Communication between processors will not be required for this policy.

Policy 2. Each processor will calculate the upper triangular and diagonal coefficients associated with its collection of nodes The coefficients that are associated with the connection between the processor's interior and exterior nodes will be sent and received

between processors. In particular, the lower triangular coefficients must be received and the upper triangular ones sent This communication can be done on the Finite Element Machine via the local neighbor links or the global bus.

In either case, all the coefficients that are necessary for the displacement calculation will be stored in the memory of the processor.

Both strategies can be implemented by providing each processor with a table of elements and their associated properties (type, thickness, etc.), coordinates of interior and exterior nodes, and associated processors for the exterior nodes for use in data communication. Typical problem data for processor P is given in Table 2.

Elements	Global Node Number	Node Coordinates	Processor
8 7 5	4	(0,1)	_
5 6 8	5	(1,1)	_
6 5 3	7	(0,2)	_
2 3 5	1	(0,0)	R
5 4 2	2	(1,0)	R
4 5 7	3	(2,0)	R
1 2 4	6	(2,1)	Q
	8	(1,2)	Q

Table 2. Problem Data for Processor P

During the assembly process, integrations are performed over the elements (one at a time) in a processor's element table and the resulting

contributions are added to the appropriate global coefficients For example, integrations over element 4-5-7 (E6) yields contributions to the coefficients shown below.

$$\begin{array}{c} \\ 4 \\ 5 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccc} 4 & 5 & 7 \end{array} \\ \begin{bmatrix} [x] & [x] & [x] \\ & [x] & [x] \\ & & [x] \end{bmatrix} \end{array}$$

Now consider the element 5-4-2 (E5). Processor P calculates the following contributions for the cases of commmunication (Policy 2) and no communication (Policy 1) between processors respectively while processor R calculates the same contributions regardless of the policy.

$$\begin{array}{c} \\ 2 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 4 & 5 \end{array} \\ \begin{bmatrix} & & \\ & [x] & [x] \\ & & [x] \end{bmatrix} \end{array}$$

Processor P
(Policy 1)

$$\begin{array}{c} \\ 2 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 4 & 5 \end{array} \\ \begin{bmatrix} & & \\ [x] & [x] & [x] \\ [x] & & [x] \end{bmatrix} \end{array}$$

Processor P
(Policy 2)

$$\begin{array}{c} \\ 2 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 4 & 5 \end{array} \\ \begin{bmatrix} [x] & [x] & [x] \\ & & \\ & & \end{bmatrix} \end{array}$$

Processor R

This is possible because processor R also has element 5-4-2 in its element table For Policy 2, the 4-2 and 5-2 contributions must be sent by processor R to processor P and received by processor P, whereas for Policy 1, the 4-2 (or 2-4) and the 5-2 (or 2-5) contributions are calculated by both processors P and R thereby resulting in a duplication of effort Similar arguments hold for the 5-5 contributions from the integrations over elements E1,E2,E3, and E4 in Figure 1. The amount of communication overhead and effort duplication are analyzed in section 4 2

The output of the assembly process will be the data structures KCOEFF and CONNECTED_TO which describe the K matrix coefficients necessary for the calculation of displacements at processor P's nodes, and the data structure SEND_TO which describes the processors to which values from P must be sent during displacement calculation. These structures are illustrated below for the region of Figure 1

Node	KCOEFF
4	[4] [5] [7] [1] [2]
5	[5] [4] [7] [2] [3] [6] [8]
7	[7] [4] [5] [8]

Node	CONNECTED_TO
4	5 7 1 2
5	4 7 2 3 6 8
7	4 5 8

Node	SEND_TO
4	R
5	R Q
7	Q

It is possible to use more space efficient storage structures since both the upper and lower nonzero parts of the symmetrix K matrix are stored. However, these structures were implemented to allow for ease in the computation required by the iterative solution algorithm for the displacements Also, if many processors are available so that a small number of nodes may be assigned to each processor, this extra storage will not be prohibitive

A routine that uses Policy 1 and the data structures described above was written for a FEM of any number of processors and implemented on a 4 processor FEM for the equations of plane stress on a region discretized by linear triangular elements as shown in Figure 2b The same ideas can be used for other partial differential equations and finite elements as well since the influence of a particular partial

differential equation and finite element is contained in a subprocedure that sets the value of the coefficients after performing integrations over the element

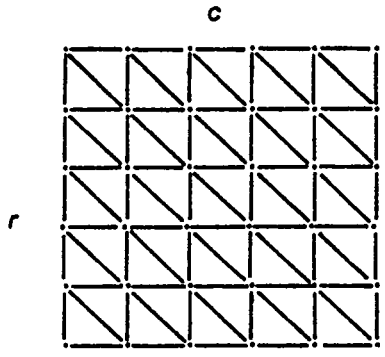
4.2. Speedup for Parallel Matrix Assembly

The speedup for the matrix assembly process is the time to assemble the matrix on a uniprocessor machine divided by the time to assemble the matrix on an array with p processors. The processor efficiency is defined to be the speedup divided by the number of processors:

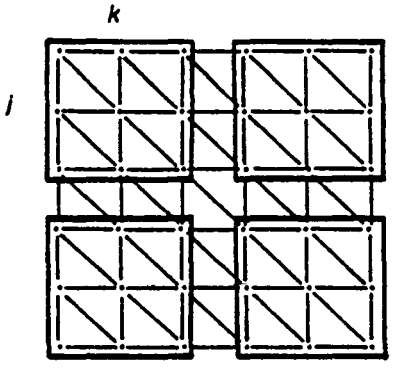
$$\begin{aligned} Speedup(p) &= Time(1)/Time(p) \\ Efficiency(p) &= Speedup(p)/p \end{aligned} \tag{4.1}$$

For a rectangular domain the speedup is easily calculated and will be described below for the symmetric stiffness matrix that results from a plane stress analysis of a plate that has been discretized by linear triangular finite elements, however, the same type of analysis can be done for other problems and finite elements as well

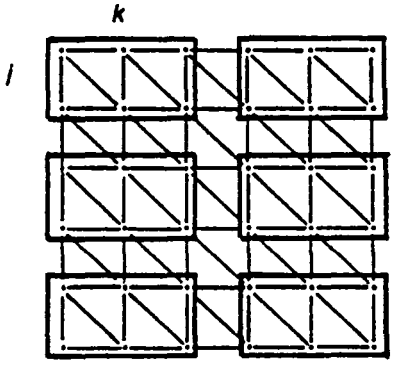
Let the plate be discretized so that N nodes are arranged in r rows and c columns as shown in Figure 2a



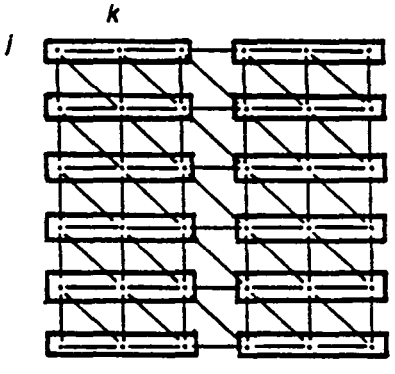
$r=6, c=6, N=36$
Figure 2a. Discretization



$l=3, k=3, p=4$
Figure 2b. Four Processors



$l=2, k=3, p=6$
Figure 2c. Six Processors



$l=1, k=3, p=12$
Figure 2d. Twelve Processors

First, the time required for a uniprocessor matrix assembly will be derived. Since K is symmetric, only the upper triangular and the diagonal part of K will be calculated. If each node in the plate represents d unknowns ($d=2$ for the plane stress problem), the K matrix can be partitioned such that each partitioned row represents the equations at a single node in the problem grid. Then for the discretization shown in Figure 2a there will be at most 3 dxd matrices in the strictly upper triangular part of each partitioned row and the diagonal entry will be a

dxd symmetric matrix. These four matrices can be visualized as the contributions from a node's northwest, north, and east neighbor nodes as well as the contribution from the node itself. The connectivities of these neighbor nodes to the particular node C are shown in Figure 3.

Figure 3. Upper Triangular Connections to Node C

Now because of symmetry only $(d^2+d)/2$ elements of the *dxd* matrix on the diagonal must be calculated. The off diagonal matrices on the other hand are not necessarily symmetric and also may be full so that all d^2 elements must be formed. The total number of entries that must be calculated for the diagonal matrices and the matrices in the upper triangular part of *K* are itemized in (4.2).

$$\begin{aligned} &N(d^2+d)/2 && \text{for the diagonal matrices} \\ &(N-c)d^2 && \text{for the north matrices} \\ &(N-r)d^2 && \text{for the east matrices} \\ &(N-r-c+1)d^2 && \text{for the northwest matrices} \end{aligned} \tag{4.2}$$

Hence, the total time (in units of the number of entries) needed to assemble the *K* matrix on a uniprocessor machine is

$$Time(1) = (3.5N-2r-2c+1)d^2+0.5Nd \tag{4.3}$$

Next, the time to assemble *K* in parallel will be given for three different array processor arrangements first, for the special case of all boundary processors, that is, 4 processors arranged in a 2x2 grid, secondly, for $p>4$ processors arranged in a $p/2 \times 2$ grid, and lastly for

$p > 4$ processors arranged in a $\sqrt{p} \times \sqrt{p}$ grid. The $p/2 \times 2$ grid is only considered here for the purpose of analyzing the cases shown in Figure 2. In practice the problems of interest will be large enough to utilize a processor grid that contains interior processors For all three cases, assume N nodes are partitioned to p processors with each processor receiving an jxk grid of nodes as illustrated in Figures 2b, 2c, and 2d

The number of upper triangular and diagonal coefficients that must be calculated by the lower right corner processor when processors are arranged in a 2x2 grid is itemized in (4.4). This lower right corner processor is the limiting processor in the sense that it has more data communication to perform.

$$\begin{aligned} &jk(d^2+d)/2 && \text{for } \textit{the diagonal matrices} \\ &jkd^2 && \text{for } \textit{the north matrices} \\ &jkd^2 && \text{for } \textit{the northwest matrices} \\ &j(k-1)d^2 && \text{for } \textit{the east matrices} \end{aligned} \tag{4 4}$$

For Policy 1, extra time must be added to (4 4) to account for the redundant calculation of the west matrices for the non-interior nodes of the processor For Policy 2, these values must be received instead of calculated and the non-interior upper triangular values sent The number of duplications for Policy 1 and the number of sends and receives for Policy 2 are given by (4 5)

$$\begin{aligned} \textit{Duplication}(4) &= jd^2 \\ \textit{Receive}(4) &= jd^2 \\ \textit{Send}(4) &= (2k+j-1)d^2 \end{aligned} \tag{4 5}$$

Let a and b represent the number of coefficient calculations required to

equal the time of one send and one receive operation respectively Then, the total time for the parallel matrix assembly is given by (4 6a) and (4.6b) for Policies 1 and 2 respectively.

$$\begin{aligned} Time_1(4) &= 3\,5jkd^2 + 0\,5jkd \\ Time_2(4) &= [3\,5jk - j + a(2k + j - 1) + b_j]d^2 + 0.5jkd \end{aligned} \tag{4.6a}$$

The speedup and efficiency can now be calculated by using (4 1) These values are given in (4 7a) and (4 7b) for Policy 1 and 2 respectively.

$$\begin{aligned} Speedup_1(4) &= \frac{(3.5N - 2r - 2c + 1)d^2 + 0.5Nd}{3.5jkd^2 + 0\,5jkd} \\ Efficiency_1(4) &= \frac{(3.5N - 2r - 2c + 1)d^2 + 0.5Nd}{3\,5Nd^2 + 0.5Nd} \end{aligned} \tag{4.7a}$$

$$\begin{aligned} Speedup_2(4) &= \frac{(3.5N - 2r - 2c + 1)d^2 + 0\,5Nd}{[3\,5jk + (b_j - j + a(2k + j - 1))]d^2 + 0\,5jk} \\ Efficiency_2(4) &= \frac{(3.5N - 2r - 2c + 1)d^2 + 0.5Nd}{[3.5N + p(b_j - j + a(2k + j - 1))]d^2 + 0\,5Nd} \end{aligned} \tag{4 7b}$$

If $p > 4$ processors are arranged in a $p/2 \times 2$ grid so that each processor is on the grid boundary, the number of upper triangular and diagonal coefficients that must be calculated by the limiting processors on the left boundary of the processor array is itemized in (4 8)

$jk(d^2 + d)/2$	for *the diagonal matrices*
jkd^2	for *the north matrices*
$j(k - 1)d^2$	for *the northwest matrices*
jkd^2	for *the east matrices*

(4 8)

For Policy 1, the redundant calculation of the south and southeast border matrices must be added to (4.8). This duplication and the receive and send communications that are necessary to implement Policy 2 are given in (4.9).

$$\begin{aligned} Duplication\,(p) &= (2k+j-1)d^2 \\ Send\,(p) &= (2k+j-1)d^2 \\ Receive\,(p) &= (2k+j-1)d^2 \end{aligned} \tag{4.9}$$

The total time to assemble in parallel the K matrix for Policies 1 and 2 is given in (4.10a) and (4.10b) respectively.

$$Time_1\,(p) = (3.5jk+2k-1)d^2+0.5jkd \tag{4.10a}$$

$$Time_2\,(p) = [3.5jk-j+(a+b)(2k+j-1)]d^2+0.5jkd \tag{4.10b}$$

The speedup and efficiency are given by (4.11a) (4.11b) for Policy 1 and Policy 2 respectively.

$$\begin{aligned} Speedup_1\,(p) &= \frac{(3.5N-2r-2c+1)d^2+0.5Nd}{(3.5jk+2k-j)d^2+0.5jkd} \\ Efficiency_1\,(p) &= \frac{(3.5N-2r-2c+1)d^2+0.5Nd}{(3.5N+p(2k-1))d^2+0.5Nd} \end{aligned} \tag{4.11a}$$

$$\begin{aligned} Speedup_2\,(p) &= \frac{(3.5N-2r-2c+1)d^2+0.5Nd}{[3.5jk-j+(a+b)(2k+j-1)]d^2+0.5jkd} \\ Efficiency_2\,(p) &= \frac{(3.5N-2r-2c+1)d^2+0.5Nd}{[3.5N-pj+p(a+b)(2k+j-1)]d^2+0.5Nd} \end{aligned} \tag{4.11b}$$

Lastly, assume that $p>4$ processors are arranged in a square $\sqrt{p} \times \sqrt{p}$ grid so that there will be one or more processors completely inside the processor grid. For this case, the upper triangular and diagonal coefficients calculated by a limiting interior processor are

itemized in (4.12).

$$\begin{aligned} &jk(d^2+d)/2 && \text{for the diagonal matrices} \\ &jkd^2 && \text{for the north matrices} \\ &jkd^2 && \text{for the northwest matrices} \\ &jkd^2 && \text{for the east matrices} \end{aligned} \tag{4.12}$$

For Policy 1, the duplicated calculations for this processor arrangement are due to the calculation in each processor of the west, south, and southeast matrices associated with the connection between the interior and exterior nodes and is given in (4.13). The sends and receives necessary to implement Policy 2 are also given in (4.13).

$$\begin{aligned} Duplication(p) &= (2k+2j-1)d^2 \\ Send(p) &= (2k+2j-1)d^2 \\ Receive(p) &= (2k+2j-1)d^2 \end{aligned} \tag{4.13}$$

The total time to calculate the K matrix for Policy 1 and 2 is given in (4.14a) and (4.14b) respectively

$$Time_1(p) = (3.5jk+2k+2j-1)d^2+0.5jkd \tag{4.14a}$$

$$Time_2(p) = [3.5jk+(a+b)(2k+2j-1)]d^2+0.5jkd \tag{4.14b}$$

The associated speedups and efficiencies are given by (4.15a) and (4.15b)

$$\begin{aligned} Speedup_1(p) &= \frac{(3.5N-2r-2c+1)d^2+0.5Nd}{(3.5jk+2k+2j-1)d^2+0.5jkd} \\ Efficiency_1(p) &= \frac{(3.5N-2r-2c+1)d^2+0.5Nd}{(3.5N+p(2k+2j-1))d^2+0.5Nd} \end{aligned} \tag{4.15a}$$

$$Speedup_2(p) = \frac{(3.5N-2r-2c+1)d^2+0.5Nd}{[3.5/k+(a+b)(2k+2/-1)]d^2+0.5/kd}$$

$$Efficiency_2(p) = \frac{(3.5N-2r-2c+1)d^2+0.5Nd}{[3.5N+p(a+b)(2k+2/-1)]d^2+0.5Nd} \qquad (4.15b)$$

The values of the total assembly time, the time due to duplication, the speedup, and the efficiency are given for Policy 1 in Table 3, for the particular p, $/$, and k values corresponding to Figures 2b, 2c, and 2d.

p	$/$	k	Total	Duplication	Speedup	Efficiency
1	6	6	448	0	----	---
4	3	3	135	12	3.32	83%
6	2	3	110	28	4.07	68%
12	1	3	65	24	6.89	57%

Table 3. Assembly Times for Figure 2. (Policy 1.)

Equations (4.9) and (4.13) show that the duplication is a decreasing function of the number of processors when $p>4$ but the efficiency also decreases since the duplication comprises a larger percentage of the parallel time. This is also seen from Table 3.

Results on a 2x2 FEM for the 36 node plane stress problem with 2 equations per node and 9 nodes per processor (illustrated in Figure 2b) show a speedup of 3.2 over the corresponding uniprocessor algorithm for Policy 1. This number compares quite well with the value of 3.32 in Table 3.

The values of the sequential assembly time, the parallel assembly time, the time due to communication overhead, the speedup, and the

efficiency are given for Policy 2 in Tables 4a, 4b, and 4c for the particular p, j, and k values corresponding to Figures 2b, 2c, and 2d

p	j	k	Total	Overhead	Speedup	Efficiency
1	6	6	448	0	----	---
4	3	3	167	44	2.68	67%
6	2	3	138	56	3.25	54%
12	1	3	89	48	5.03	42%

Table 4a. Assembly Times for Figure 2.
$a=1; b=1$ (Policy 2)

p	j	k	Total	Overhead	Speedup	Efficiency
1	6	6	448	0	----	---
4	3	3	145	22	3.09	77%
6	2	3	110	28	4.07	68%
12	1	3	65	24	6.89	57%

Table 4b. Assembly Times for Figure 2.
$a=0.5, b=0.5$ (Policy 2)

p	j	k	Total	Overhead	Speedup	Efficiency
1	6	6	448	0	----	---
4	3	3	134	11	3.34	84%
6	2	3	96	14	4.67	78%
12	1	3	53	12	8.45	70%

Table 4c. Assembly Times for Figure 2.
$a=0.25, b=0.25$ (Policy 2)

The values in Tables 3 and 4a, 4b, and 4c are for either a 2x2 or a $p/2 \times 2$ processor grid. Equations (4.13), (4.14), and (4.15) were used to predict the corresponding times for a plate with 768 nodes arranged in 16 rows and 48 columns in order to to investigate the effect of completely interior processors when $p > 4$. The results are given in Table 5 for Policy 1 and Table 6a, 6b, 6c. for Policy 2.

p	l	k	Total	Overhead	Speedup	Efficiency
1	16	48	11012	0	----	---
4	8	24	2880	32	3.82	96%
16	4	12	844	124	13.05	82%
64	2	6	240	60	45.88	72%
256	1	3	73	28	150.85	59%

Table 5. Assembly Times for 16x48 Plate (Policy 1)

p	l	k	Total	Overhead	Speedup	Efficiency
1	16	48	11012	0	----	---
4	8	24	3100	252	3.55	89%
16	4	12	968	248	11.38	71%
64	2	6	300	120	36.71	57%
256	1	3	101	56	109.03	43%

Table 6a. Assembly Times for 16x48 Plate $a=1; b=1$ (Policy 2)

p	i	k	Total	Overhead	Speedup	Efficiency
1	16	48	11012	0	----	---
4	8	24	2972	126	3.70	93%
16	4	12	844	124	13.05	82%
64	2	6	240	60	45.88	72%
256	1	3	73	28	150.85	59%

Table 6b. Assembly Times for 16x48 Plate
$a=0.5; b=0.5$ (Policy 2)

p	i	k	Total	Overhead	Speedup	Efficiency
1	16	48	11012	0	----	----
4	8	24	2911	63	3.78	95%
16	4	12	782	62	14.08	88%
64	2	6	210	30	52.44	82%
256	1	3	59	14	186.00	73%

Table 6c. Assembly Times for 16x48 Plate
a=0.25;b=0.25 (Policy 2)

The results in Tables 3, 4, 5, and 6 show that for values of a and b below 0.5 the best policy for assembling the stiffness matrix K on an array of $p>4$ processors will be Policy 2, that is, communication between the processors is warranted. For the special case of 4 processors, the values of a and b must be lower than 0.25 before Policy 2 is recommended. The conditions that must be satisfied for Policy 2 to be the best policy are easily found from equations 4.6a and 4.6b, 4.10a and 4.10b, 4.14a and 4.14b for the cases of a 2x2 processor grid, a $p/2 \times 2$ grid, and a $\sqrt{p} \times \sqrt{p}$ grid respectively. These conditions

are given in (4 16) below.

$$\begin{aligned} a(2k+j-1)+j(b-1) &< 0 \qquad \text{for } p=4 \\ (a+b) &< 1 \qquad \text{for } p>4 \end{aligned} \tag{4 16}$$

When the problem of Figure 2a. is solved with 4 processors, $j=3$ and $k=3$ so that the conditions in (4.16) become the following:

$$\begin{aligned} 8a + 3b &< 3 \qquad \text{for } p=4 \\ (a+b) &< 1 \qquad \text{for } p>4 \end{aligned} \tag{4.17}$$

For the problem of the 16x48 plate, $j=8$ and $k=24$ when 4 processors are used so that the conditions in (4.16) become the following:

$$\begin{aligned} 55a + 8b &< 8 \qquad \text{for } p=4 \\ (a+b) &< 1 \qquad \text{for } p>4 \end{aligned} \tag{4.18}$$

Recall that a and b are the number of K matrix coefficient calculations that comprise the time to send and receive a value between processors respectively. Equation (4.16) shows that Policy 2 is more likely to be the optimal policy when the values of a and b are small The values of a and b will decrease for two reasons. First, if the communication between processors is made faster a and b will necessarily be less. Secondly, if higher order elements or more complicated integration rules are used the time to calculate one coefficient will increase which will in effect make a and b less For these situations, the assembly process should include communication between processors.

4.3. Parallel Stress Calculation

The purpose of this section is to describe the stress calculation, and demonstrate that it can be made with no communication between processors.

After the system of displacement equations has been solved (the solution algorithms will be discussed in Chapters 5. and 6.), the displacements at the nodal points in processor P's CONNECTED_TO data structure are in processor P's local memory since these values were either calculated by processor P or were passed to it during execution. Hence, the nodal displacement values on the elements in processor P's element table are resident in processor P's memory. As an illustration, consider the processor assignment in Figure 5.

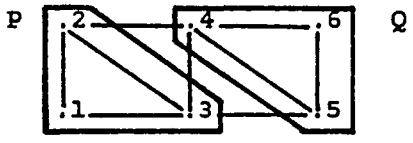


Figure 4. Processor Assignment

Displacements at the nodes 2, 3, 4, and 5 are in processor P's local memory after the displacement calculation is complete. Likewise, the same values are in processor Q's memory.

For the case of linear basis functions on the triangular elements, the stresses are constant across the triangles. The obvious question is whether processor P or Q should calculate the stresses on a given triangle. Define as the first node of the triangles in Figure 5 the node associated with the right angle and then number the remaining two nodes in a counterclockwise fashion. A good rule would be to require the processor that has the first node of the element as an interior node to calculate the stresses on that element since this will require no duplication of effort. For example, processor P calculates stresses on element 1-3-2 and element 3-5-4 whereas processor Q calculates

stresses on element 6-4-5 and element 4-2-3.

Recall from Chapter 2 that the actual stress calculation, in the linear element case, involves the pre-multiplication of the element's nodal displacement vector by a stress matrix that is a function of the coordinates of the element's nodal points. If the coordinate information in Table 2. is available to the stress procedure, this matrix is rapidly calculated.

Stress calculation results for a 36 node plane stress problem (50 elements) run on a 2x2 FEM showed a speedup of 4 over a uniprocessor version for the stress calculation. These results indicate that a speedup of $O(p)$ can be expected when p processors are used to calculate the stresses. This perfect speedup is a consequence of the absence of both communication between processors and redundant calculations.

The use of higher order basis functions will not produce constant stresses over an element, but the stresses can be calculated from the element's nodal displacements and nodal coordinate values without any duplication of effort among the processors Hence, for these basis functions, $O(p)$ speedup is also predicted

CHAPTER 5

Parallel Linear Stationary Iterative Methods

In this chapter we consider the implementation of linear stationary iterative methods for the solution of

$$K\underline{u}=\underline{f} \tag{5.1}$$

on both vector computers and parallel arrays. For concreteness, we will use the CYBER 203/205 as an example of the former and NASA Langley's Finite Element Machine as an example of the latter. The implementation of Jacobi's method is discussed in Section 5.1, the description and implementation of a new method, Multi-color SOR, is given in Section 5.2, a Multi-color SSOR method is discussed in Section 5.3, and implementation considerations for block iterative methods is addressed in Section 5.4.

5.1. The Jacobi Iterative Method

Let the matrix K with elements k_{ij} be split as

$$K = D-L-U \tag{5.2}$$

where D is the diagonal part of K and $-L$ and $-U$ are the strictly lower and upper triangular parts of K respectively. Then the Jacobi iterative method for solving (5.1) is given by

$$D\underline{u}^{k+1} = (L+U)\underline{u}^{k} + \underline{f} \tag{5.3}$$

or

$$\underline{u}^{k+1} = B\underline{u}^{k} + \underline{c} \tag{5.4}$$

where

$$\underline{c} = D^{-1}\underline{f}$$

$$B = D^{-1}(L+U) \tag{5.5}$$

and the matrix B is called the Jacobi iteration matrix. The conditions for the iteration (5.4) to converge are given below (Young[1971]).

Jacobi Convergence Theorem

> Let K be a real symmetric positive definite matrix. Then the Jacobi iteration converges if and only if $D+L+U$ is positive definite.

This theorem shows that the Jacobi method is not guaranteed to converge for all symmetric and positive definite matrices K such as those arising from finite element discretizations as discussed in Chapter 2

Closely related to the Jacobi method is the simultaneous overrelaxation method (JOR method) defined by

$$\underline{u}^{k+1} = B_\omega \underline{u}^k + \omega \underline{c} \tag{5.6}$$

where

$$B_\omega = \omega B + (1-\omega)I \tag{5.7}$$

and B_ω is called the JOR iteration matrix.

The conditions for this iteration to converge are given below (Young [1971]):

JOR Convergence Theorem

> Let K be a real symmetric positive definite matrix Then the JOR iteration converges if and only if $2\omega^{-1}D-K$ is positive definite. The condition that $2\omega^{-1}D-K$ is positive definite may

be replaced by $0<\omega<\frac{2}{1-u_{min}}<2$ where $u_{min}\leq 0$ is the smallest eigenvalue of B.

This theorem implies that by appropriately choosing ω, the JOR method can be made to converge if $u_{min}\leq 0$. However, this choice of ω depends on knowledge of the smallest eigenvalue of B. In fact, Hayashi and Yokomana [1977] report that JOR diverged for finite element discretizations of typical structural problems such as cantilevered beams and simply supported plates unless the relaxation parameter ω was carefully chosen. Hence, for problems of interest to us, the JOR or Jacobi methods are not suitable because the eigenvalues of B are rarely known in advance and convergence is not guaranteed for $\frac{2}{1-u_{min}}<\omega<2$. However, the implementation of these methods on vector computers or parallel arrays is of interest to us since these methods may successfully be used as preconditioners for the conjugate gradient method as discussed in Chapter 6 or in the implementation of an SOR method as will be discussed in Section 5.2.

We now describe how Jacobi's method can be implemented on a parallel computer For concreteness, we consider an elliptic equation of the form

$$u_{xx} + au_{xy} + u_{yy} = f \tag{5.8}$$

on the unit square with Dirichlet boundary conditions where a is a given constant and f is a given function of x and y. We discretize (5.8) with the usual second-order finite difference approximations (see, e g , Forsythe and Wasow [1960]) which give the difference equations

$$u_{i+1,j}+u_{i-1,j}+u_{i,j+1}+u_{i,j-1}-4u_{ij}$$
$$+\frac{a}{4}[u_{i+1,j+1}-u_{i-1,j+1}+u_{i-1,j-1}-u_{i+1,j-1}] = h^2 f_{ij} \tag{5.9}$$

where h is the spacing between grid points, $i,j=1,2,\ldots,N$, $h(N+1)=1$, u_{ij} denotes the approximate solution at the i,jth grid point, and $f_{ij}=f(ih,jh)$.

Now, the Jacobi method (5.4) for (5.9) can be written in the form used for implementation as

$$u_{ij}^{(k+1)} = \frac{1}{4}[u_{i+1,j}^{(k)}+u_{i-1,j}^{(k)}+u_{i,j+1}^{(k)}+u_{i,j-1}^{(k)}]-\frac{1}{4}h^2 f_{ij}$$
$$+\frac{a}{16}[u_{i+1,j+1}^{(k)}-u_{i-1,j+1}^{(k)}+u_{i-1,j-1}^{(k)}-u_{i+1,j+1}^{(k)}] \tag{5.10}$$
$$u_{ij}^{(o)} = \hat{u}_{ij}$$

First, we consider the implementation of (5.10) on the CDC CYBER 203/205 where vectors consist of contiguous storage locations and the efficiency of the vector operations is strongly dependent on vector length with the maximum efficiency achieved for very long vectors For vectors of length 1000 around 90% efficiency is obtained, but this drops to approximately 50% or less for vectors of length 100 and less than 10% for vectors of length 10 Hence, we would like to keep vector lengths on the order of 1000 or more whenever possible Now for (5 9) suppose that h= 01 so that N=99 and $n=N^2\approx 10^4$ If we consider the boundary values of the square region to be unknowns and order the grid points, including the boundary points, from left to right, bottom to top, the unknown vector $\underline{u}$ in (5.4) will have length $(N+2)^2$ and the new iterate $\underline{u}^{k+1}$ can be completely vectorized as a matrix vector product followed by the addition of two vectors Also note that the relaxation parameter in (5 6) causes no problem and hence the JOR method is

implemented in the same fashion However, the boundary values must not be changed by the iteration and this is prevented by use of the control vector feature on the CYBER 203/205 which allows suppression of storage of updated values into the boundary locations. (See, e.g. Lambiotte [1975] or Ortega and Voigt [1977] for more details on this procedure.) Since the calculation of new values corresponding to the boundary points is superfluous, an inefficiency of approximately 4% for N=99 is introduced; however, almost full efficiency of the vector operations results.

Next, we describe the implementation of (5.10) on the Finite Element Machine. Now, the grid point stencil for (5 9) is given in Figure 1,

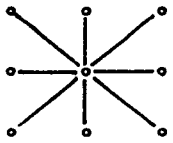


Figure 1. Stencil for (5 9)

and matches exactly the eight local neighbor connections of the FEM that was discussed in Chapter 3. Hence, if we have as many FEM processors as the N interior grid points, each interior point and its associated row of K matrix coefficients and $\underline{f}$ vector component could be assigned to one processor. The boundary nodes would not be assigned to processors, but instead their values are stored in the processors which need them. The data communication between processors can be done completely with the local communication links and the convergence flag in all processors is checked by the signal flag network Let $\underline{u}_p$ and $\underline{u}_n$ denote the portions of $\underline{u}$ that are assigned to processor p and

the logical neighbors of processor p respectively; then, the algorithm that is executed in each central processor is given below:

For $k=1,2,\ldots,k_{max}$ *do*

(1) Solve for $\underline{u}_p^{k+1}$.

(2) Send $\underline{u}_p^{k+1}$ to the logical neighbors via the local links and global bus if needed.

(3) If $\left\| \underline{u}_p^{k+1} - \underline{u}_p^{k} \right\|_\infty < \epsilon$ raise the convergence flag.

(4) If the convergence flag is raised in all processors then stop else continue.

(5) Receive $\underline{u}_n^{k+1}$ from the logical neighbor processors via the local links and global bus if needed.

Algorithm 1. Parallel Jacobi (One point/processor)

However, in practice it will most certainly be the case that the number of interior grid points N will greatly exceed the number of processors p. For this situation, we simply assign $[N/p]$ points per processor in such a way as to take advantage of the local links of FEM For example, suppose that $N=4p$. Then we assign the grid points to the processors as shown in Figure 2.

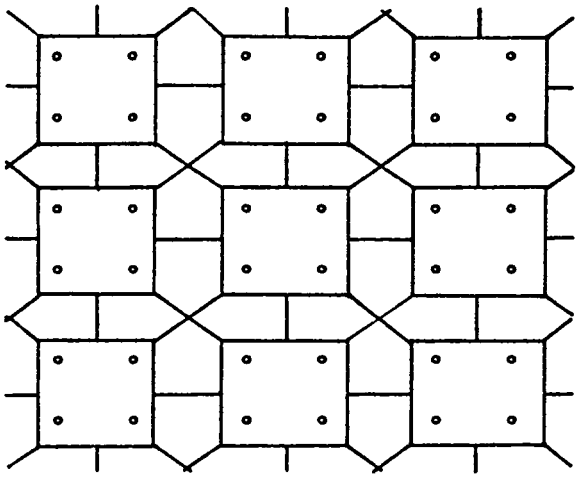


Figure 2. Processor Assignment for Jacobi's Method

and note that each processor must be connected only to its eight nearest neighbors since each point is connected to its eight nearest grid points as shown in Figure 1. Hence, only the local communication links will be required for communication. Algorithm 1 is then modified as follows.

For $k=1,2,\ldots,k_{max}$ *do*

(1) Solve for each component of $\underline{u}_p^{k+1}$ in-sequence

(2) Send the necessary components of $\underline{u}_p^{k+1}$ to the local neighbors via the local links (Only local links are needed for the stencil of Figure 1)

(3) If $\left\| \underline{u}_p^{k+1} - \underline{u}_p^{k} \right\|_\infty < \epsilon$ raise the convergence flag.

(4) If the convergence flag is raised in all processors then stop else continue.

(5) Receive $\underline{u}_n^{k+1}$ from the logical neighbor processors via the local links. (Only local links are needed for the stencil of Figure 1)

Algorithm 2 Parallel Jacobi (Multiple points/processor)

For processors on the border of the processor array, values will not be sent to and received from all eight of the neighbor processors and consequently, the algorithms in these processors may be different to reflect this; or alternatively, the same algorithm could be used in all processors with a test included to determine a processor's position in the array. A third option would be to maintain the same algorithm in each processor and provide each processor with the appropriate lists of unknowns and associated processors to send data to and receive data from. This was the approach taken in Chapter 4 where the connectivity of the grid points was determined by the assembly process If this connectivity data were coupled with an algorithm that assigns the points to the processors, the appropriate information for communication would be available to each processor and the algorithms in all processors would be the same

Algorithm 2 will allow each processor to run without waiting on other processors with the exception of the synchronization in step (5) and the convergence test in (4). Because the processors may complete the updating of $\underline{u}^{k+1}$ in different times due to a number of factors, slightly different clock times in the processors, different memory access times, especially for those processors connected to the boundary, different number of unknowns per processor if p does not evenly divide N, synchronization of the processors to some degree is realized by the synchronous RECEIVE command which causes processors to wait until the value to be received is available before computation continues This, in effect, allows the slower processors to catch up and also ensures that the same answer will be obtained for the problem on the processor array as on a single processor Note that the processors are not

required to operate in a SIMD or lockstep fashion, but the information for the next iteration must be obtained from neighbor processors before the iteration continues. If we relax this requirement and use an asynchronous RECEIVE, the processors may run asynchronously and the delay times will be reduced. The numerical iterates will however deviate from the true mathematical iteration but Baudet [1978] shows that this may be beneficial

The second source of delay is the convergence test for the iterative process. The local convergence test in (3) of Algorithm 2 can be done in all processors simultaneously and therefore incurs no delay. However, at the end of each iteration, the convergence flag must be checked in all processors as indicated by (4) of Algorithm 2. If all the flags are not set, the processor continues with the next iteration. Hence, the entire process will not terminate until all unknowns have satisfied the convergence criterion and towards the end of the process a portion of the processors may be doing unnecessary work. This seems to be an unavoidable inefficiency.

In the absence of these delays, if p evenly divides N, and if the processors operate at the same speed, the Jacobi method on the Finite Element Machine will have speedup $O(p)$ The actual speedup, however, will be a function of the ratio of the communication to calculation times of the processors and is discussed in Chapter 7.

5.2. The Multi-Color SOR Method

5.2.1. Motivation

Let the matrix K be split as given by (5.2). Then the SOR iteration applied to (5.1) is given by

$$\frac{1}{\omega}(D-\omega L)\underline{u}^{k+1} = \frac{1}{\omega}[\omega U+(1-\omega)D]\underline{u}^{k}+\underline{f} \tag{5.11}$$

or

$$\underline{u}^{k+1} = L_{\omega}\underline{u}^{k} + \underline{c} \tag{5.12}$$

where

$$\underline{c}=\omega(D-\omega L)^{-1}\underline{f}$$

$$L_{\omega}=(D-\omega L)^{-1}(\omega U+(1-\omega)D) \tag{5.13}$$

L_{ω} is called the SOR iteration matrix and ω is the relaxation parameter chosen to enhance convergence.

The conditions for (5.11) to converge for symmetric matrices with positive diagonal elements is given by the Ostrowski-Reich Theorem (Varga[1962]).

Ostrowski-Reich Theorem

> Let K be a symmetric matrix with positive diagonal elements. Then the SOR method converges if and only if K is positive definite and $0<\omega<2$.

Since the problems of interest to us are symmetric and positive definite, SOR is guaranteed to converge if we choose $0<\omega<2$

The SOR iteration (5.12) can be written for implementation as

$$u_i^{k+1} = (1-\omega)u_i^k + \frac{\omega}{k_{ii}}\Big(f_i - \sum_{j=1}^{i-1} k_{ij}u_j^{k+1} - \sum_{j=i+1}^{n} k_{ij}u_j^k\Big) \qquad (5.14)$$

This form shows that the SOR iteration is sequential in nature since the values of u_j, $j=1,2,\ldots,i-1$ must be computed before u_i on iteration $k+1$. This was not true for the Jacobi iteration of (5.10) where only previously computed values were required for the update of a given component of $\underline{u}$. Despite this sequential nature, several authors (e.g. Hayes[1974], Lambiotte[1975]) have observed that if (5.1) arises from a five-point difference discretization of Poisson's equation and the equations are ordered according to the classical Red/Black partitioning of the grid points then an SOR sweep may be carried out, in essense, by two Jacobi sweeps, one on the equations corresponding to the red points and one for the equations corresponding to the black points. Thus, in this case, the SOR method can be effectively implemented on vector or parallel computers.

On vector computers, all the unknowns associated with the red grid points would be combined into one long vector and similarly for the unknowns associated with the black grid points For parallel arrays, an equal number of red and black equations would be assigned to each processor. The SOR iteration would be comprised of two Jacobi sweeps, one Red sweep followed by one Black sweep with each sweep performed simultaneously by the processors. After each sweep, the updated values of the respective color would be communicated between processors After the Black sweep, and hence one SOR iteration, the convergence test would be performed as described in the last section

This strategy does not work, however, for higher order finite difference or for finite element discretizations for more general elliptic equations which contain cross partial derivative terms. In these cases, it is necessary to generalize the Red/Black partitioning of the grid points to a "Multi-color" partitioning; for example, a three color partitioning, say Red/Black/Green, might give the desired result. In general, the number of colors necessary will depend on the connectivity pattern of the grid points. If p colors are used, an SOR sweep can be implemented by p Jacobi sweeps, one for each set of equations assoicated with a given color For vector computers, this reduces the effective vector length to $O(n/p)$ while for parallel arrays it is necessary that each processor hold a multiple of p equations where this multiple will be determined by the particular discretization. Clearly, there will be a point of diminishing returns as p increases, but for most differential equations and discretizations of interest it seems that no more than 6 colors will suffice and for the size of n we have in mind ($n \approx 10{,}000$ +), the Multi-color strategy can be very effective.

We note that the Multi-color orderings for SOR have been used before (Young[1971], Hackbush[1977], Hotovy and Dickson[1979]) but not in context of parallel computation for finite element discretizations.

In the next section, we describe the method in more detail and give the appropriate coloring (ordering) of the grid points for several finite difference and finite element discretizations and discuss how to implement the resulting Multi-color SOR method on parallel computers. In Section 5 2 3 we compare the Multi-color SOR method to existing theory, and in Section 5 2.4 we give numerical comparisons of the Multi-color ordering

with the lexiographical (rowwise) ordering of the grid points.

5.2.2. Multi-Color Orderings

As a first example, we consider the elliptic equation (5.8) that is discretized as given in (5.9) and partition the grid points by the Red/Black scheme as shown in Figure 3. We then number the Red grid points from left to right, bottom to top followed by the Black grid points in the same fashion.

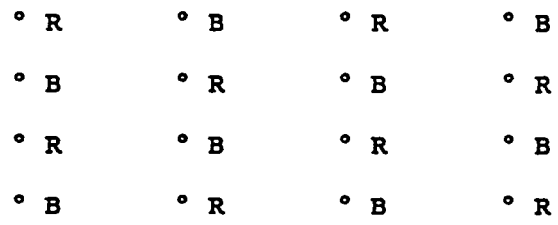


Figure 3. Red/Black Ordering

Now if a=0, so that (5 8) is just Poisson's equation, then (5 9) represents the usual five-star discretization of (5.8). It is well-known (see e.g. Young [1971]) that the difference equations (5.9) may be written in the partitioned matrix form

$$\begin{bmatrix} D & B \\ B^T & D \end{bmatrix} \begin{bmatrix} u_r \\ u_b \end{bmatrix} = \begin{bmatrix} f_r \\ f_b \end{bmatrix} \tag{5.15}$$

where D is a diagonal matrix and u_r and u_b denote the vectors of unknowns associated with the red and black grid points respectively. The Gauss-Seidel iteration for (5.15) may be written as

$$D\underline{u}_r^{k+1} = -B\underline{u}_b^k + \underline{f}_r$$

$$D\underline{u}_b^{k+1} = -B^T\underline{u}_r^{k+1} + \underline{f}_b \qquad (5.16)$$

and each part of (5.16) can then be effectively implemented in a parallel fashion, with the introduction of the SOR parameter causing no problem.

If $a \neq 0$, the form (5.15) of the difference equations is still valid although D is no longer a diagonal matrix. Hence, the unknowns corresponding to the red points are coupled to each other in (5.16) and likewise for the black points; whereas for $a=0$ they completely uncouple. The result is that (5.16) is no longer implementable in a parallel fashion. This is illustrated by the grid point stencil for (5.9) with the Red/Black ordering as shown in Figure 4.

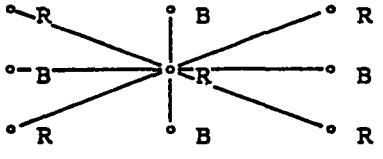


Figure 4. Stencil for (5.9)

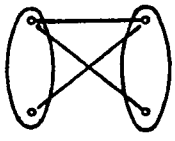
The center Red point can be seen from Figure 4 to be connected to the Red points at the four corners, and a similar stencil is obtained for the Black center points.

We wish to introduce another partitioning of the grid points for which unknowns within each subset of the partitioning are uncoupled. This is possible only if the graph associated with the discretized domain can be colored with p colors so that nodes of a given color have no edge between them. A graph with this property will be called p-partite which is formally defined below

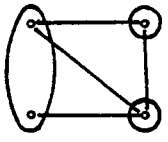
Definition 1

A graph G(V,E) with a set of vertices V and edges E is *p-partite* if its vertices form p disjoint subsets $S_1, S_2, \ldots, S_p$, with $\bigcup_{i=1}^{p} S_i = V$ such that if $uv \in E(G)$ then $u \in S_i$ and $v \in S_j$ for some $i \neq j$.

Examples of a bi-partite and a 3-partite graph are given below:



Bi-partite



3-partite

Definition 1 requires that nodes within the same subset are not connected by an edge; however, no restrictions are made on the number of nodes in a subset S_i that can have edges to nodes in subset S_j. In fact, all $\binom{p}{2}$ pairs of subsets could be connected by an edge from any node in one subset to any node in another In this case, a p-partite graph would consist of $\binom{p}{2}$ bi-partite graphs.

For example, for the stencil of Figure 4, if we use four colors, we can partition the grid points into four subsets labeled Red, Black, White, Orange so that each center point connects with only points of different colors. A suitable coloring for the stencil in Figure 4 is illustrated in Figure 5. We note that the coloring repeats beyond the given subregion.

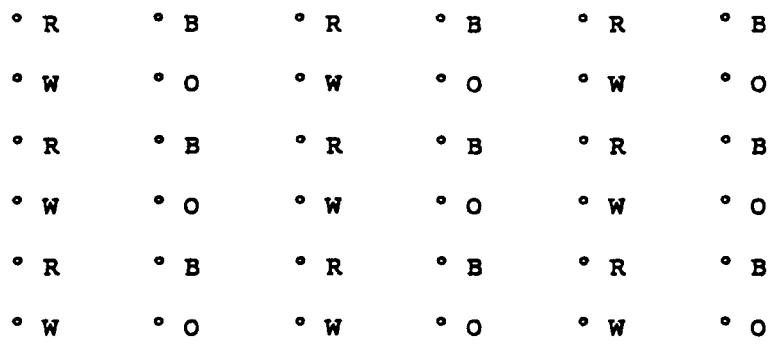


Figure 5. Four color partitioning of the gridpoints

In this case, the system (5.9) can be written in a partitioned form analogous to (5.15) as

$$\begin{bmatrix} D_1 & B_{12} & B_{13} & B_{14} \\ B_{21} & D_2 & B_{23} & B_{24} \\ B_{31} & B_{32} & D_3 & B_{34} \\ B_{41} & B_{42} & B_{43} & D_4 \end{bmatrix} \begin{bmatrix} \underline{u}_r \\ \underline{u}_b \\ \underline{u}_w \\ \underline{u}_o \end{bmatrix} = \begin{bmatrix} \underline{f}_r \\ \underline{f}_b \\ \underline{f}_w \\ \underline{f}_o \end{bmatrix} \tag{5.17}$$

where D_1, D_2, D_3, and D_4 are diagonal matrices. The Gauss-Seidel iteration in terms of (5.17) is then

$$\begin{aligned} D_1 \underline{u}_r^{k+1} &= -B_{12}\underline{u}_b^{k} - B_{13}\underline{u}_w^{k} - B_{14}\underline{u}_o^{k} + \underline{f}_r \\ D_2 \underline{u}_b^{k+1} &= -B_{21}\underline{u}_r^{k+1} - B_{23}\underline{u}_w^{k} - B_{24}\underline{u}_o^{k} + \underline{f}_b \end{aligned} \tag{5.18}$$

with similar equations for $\underline{u}_w^{k+1}$ and $\underline{u}_o^{k+1}$.

Now, since the D_i are diagonal, (5.18) is easily implementable on parallel architectures by taking four sweeps of Jacobi's method to comprise one Gauss-Siedel iteration. In particular, for vector computers, the vectors are of length $O(n/4)$ and the update of the vectors $\underline{u}_r$, $\underline{u}_b$, $\underline{u}_w$, and $\underline{u}_o$ must occur in sequence with each vector update being fully

vectorized into matrix-vector multiplications and vector additions For parallel arrays, the grid points must be partitioned into subsets and each subset assigned to a processor. The primary goal of this assignment for a machine such as the Finite Element Machine, or on a similar array with perhaps many more processors but limited processor to processor interconnections, is to keep as many processors as possible running at a given time. This, in turn, requires maximum use of the processor interconnections and minimum use of the global bus since contention for the bus will tend to introduce delays which cause processors to be idle.

This objective can be achieved by ensuring that each processor holds at least as many unknowns as a certain multiple of the number of colors where the multiple is the number of rows above the center point in the gridpoint interconnection stencil. Also, we would like to ensure as much as possible that all processors hold the same number of each color of grid points, thereby increasing the likelihood that all processors finish each Jacobi sweep on a particular color at the same time so as to reduce delays in data communication between sweeps.

Figure 6 shows the assignment of the grid points of Figure 5 to the processors.

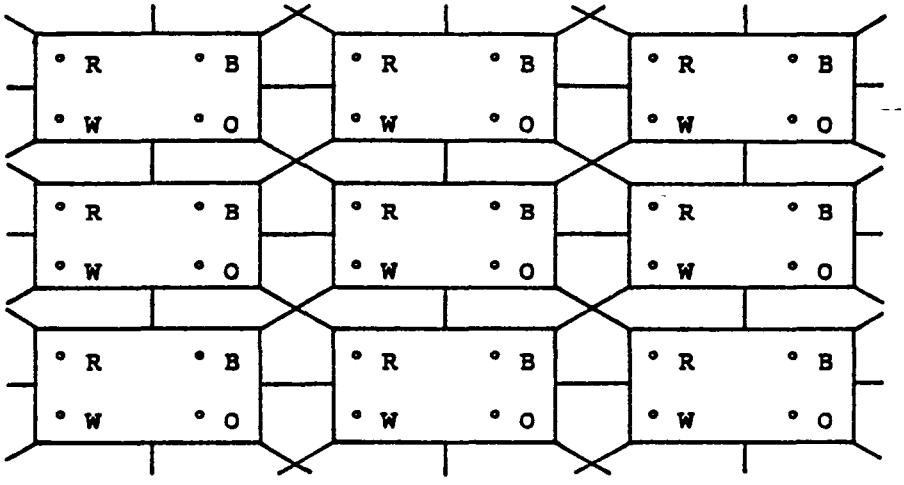


Figure 6. Processor Assignment for (5.19)

Each processor in Figure 6 holds an equal number of Red, Black, Orange, and White points. If fewer processors are available, we can assign a $2k \times 2l$ block (instead of a 2×2 block) of points to each processor since the same number of each color of points will be in any two disjoint blocks. During the solution of (5.18), the processors in the interior of the processor array communicate with all their eight compass point neighbors as can be seen from Figure 6 above and the grid point stencil in Figure 1. On the Finite Element Machine this communication can be done via the local communication links and no use of the global bus will be necessary. Each boundary processor will communicate with fewer than eight neighbor processors, the exact number depending on its location.

Let $u_{c,p}$ and $u_{c,n}$ denote the portion of nodes of color c assigned to processor p and the portion of nodes of color c that are needed by processor p for the calculation of $u_{c,p}$ but reside in other (perhaps neighbor) processors respectively. The Multi-color SOR algorithm that is executed by processor p is given below:

For k=1,2,....k_{max} do

(1) For c=1,2,...,nc do

(1) Solve for $u_{c,p}^{k+1}$

(2) Send necessary portion of $u_{c,p}^{k+1}$ to logical neighbors.

(3) Receive $u_{c,n}^{k+1}$ from logical neighbors.

(2) If $\left\| u_{c,p}^{k+1} - u_{c,p}^{k} \right\|_{\infty} < \epsilon$ set the convergence flag.

(3) If all processors have convergence flag set then stop.

Algorithm 3. Multi-color SOR

We now give the coloring of the grid points and the associated processor assignment for some common finite difference and finite element discretizations. First, consider the nine-point discretization illustrated by the stencil in Figure 7

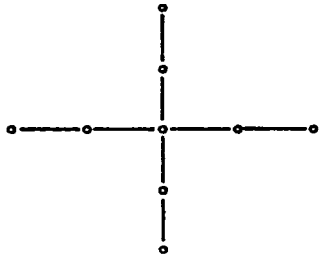


Figure 7. 9-point Discretization

The grid points are partitioned into three subsets by using the three colors Red, Black, and Green as shown in Figure 8.

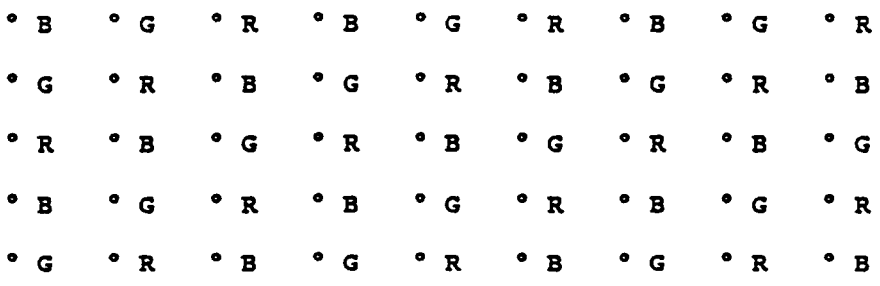


Figure 8. 3-Coloring for Figure 7.

The points can be assigned to processors in blocks of size $kx3j$ with a minimum block size of $1x3$ as shown in Figure 9.

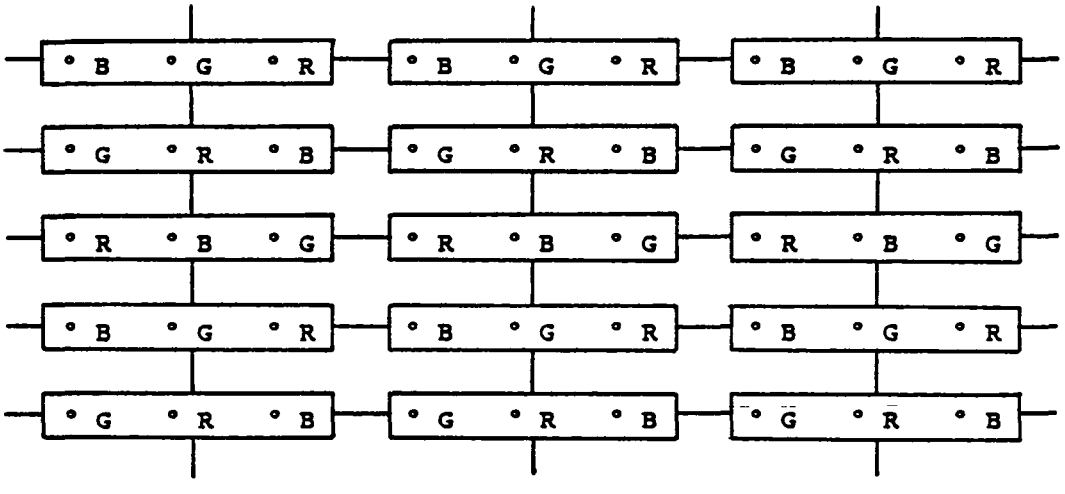


Figure 9. Processor Assignment for Figure 8.

With this assignment the North, South, East, and West local links of FEM can be used but the global bus is needed to communicate values to the next North, next South, next East, and next West neighbor processors. If the blocks were instead sized with $k>1$, only the eight local communication links are required.

Secondly, consider the thirteen-point discretization that is often used for the bi-harmonic equation and is illustrated by the grid point stencil

in Figure 10.

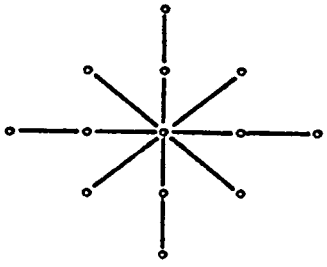


Figure 10. 13-point Discretization

The grid points are partitioned into six disconnected subsets by the use of the six colors Red, Black, White, Orange, Yellow, and Purple as shown in Figure 11.

```
° W   ° B   ° P   ° W   ° B   ° P   ° W   ° B   ° P
° R   ° Y   ° O   ° R   ° Y   ° O   ° R   ° Y   ° O
° P   ° W   ° B   ° P   ° W   ° B   ° P   ° W   ° B
° O   ° R   ° Y   ° O   ° R   ° Y   ° O   ° R   ° R
° B   ° P   ° W   ° B   ° P   ° W   ° B   ° P   ° W
° Y   ° O   ° R   ° Y   ° O   ° R   ° Y   ° O   ° R
° W   ° B   ° P   ° W   ° B   ° P   ° W   ° B   ° P
° R   ° Y   ° O   ° R   ° Y   ° O   ° R   ° Y   ° O
```

Figure 11. 6-Coloring for Figure 10.

In order to maintain the same number of each color in two distinct processors, the points must be assigned in blocks of $2k \times 3l$ with the minimum block size being 2x3 as shown in Figure 12. Note that only the eight local communication links of FEM are required.

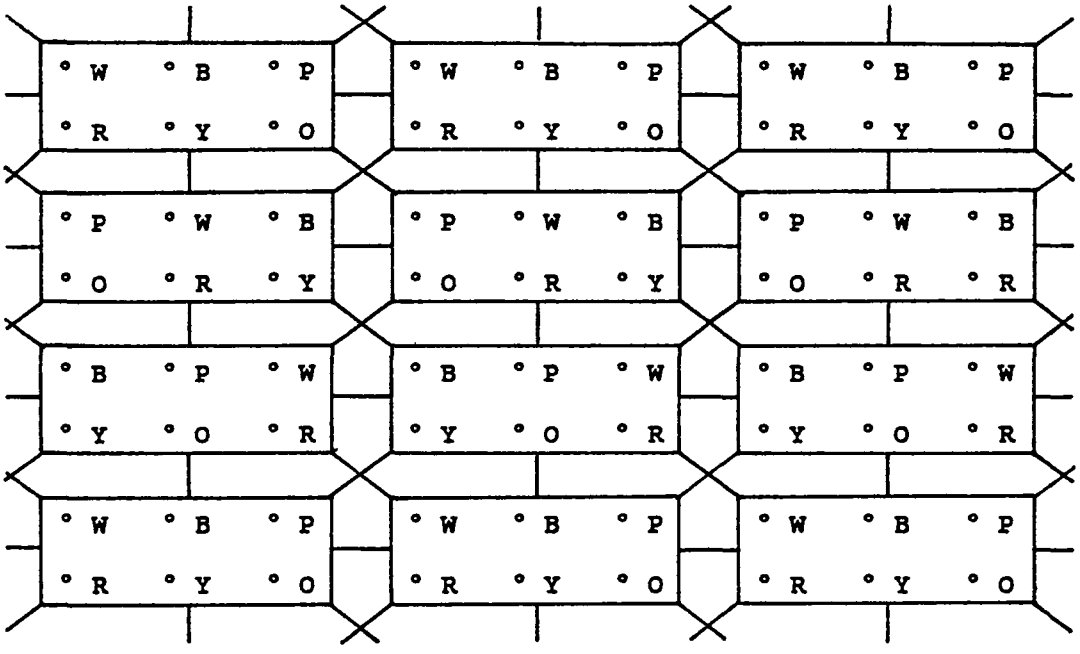


Figure 12. Processor Assignment for Figure 11.

We now consider rectangular domains that have been discretized by finite elements. Triangular elements with associated piecewise continuous (C^0) linear basis functions defined at the three vertices and their associated gridpoint stencil are shown in Figure 13.

Figure 13. Linear Triangular Element and Grid Point Stencil

The center point of the stencil is connected to the six points that share a common triangle. For this discretization, the grid points can be partitioned into three disconnected subsets by using the colors Red, Black, and Green as shown in Figure 14.

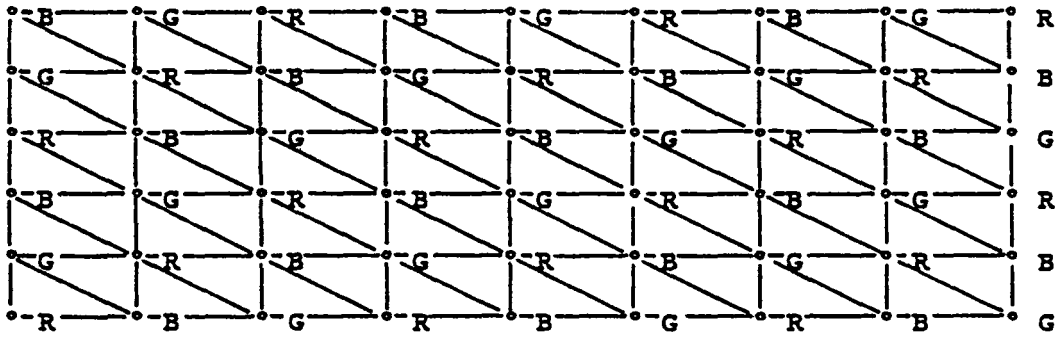


Figure 14. 3-Coloring for Figure 13.

The grid points are then assigned to processors in blocks of size $k \times 3$ with the minimum block being of size 1×3 as shown in Figure 15.

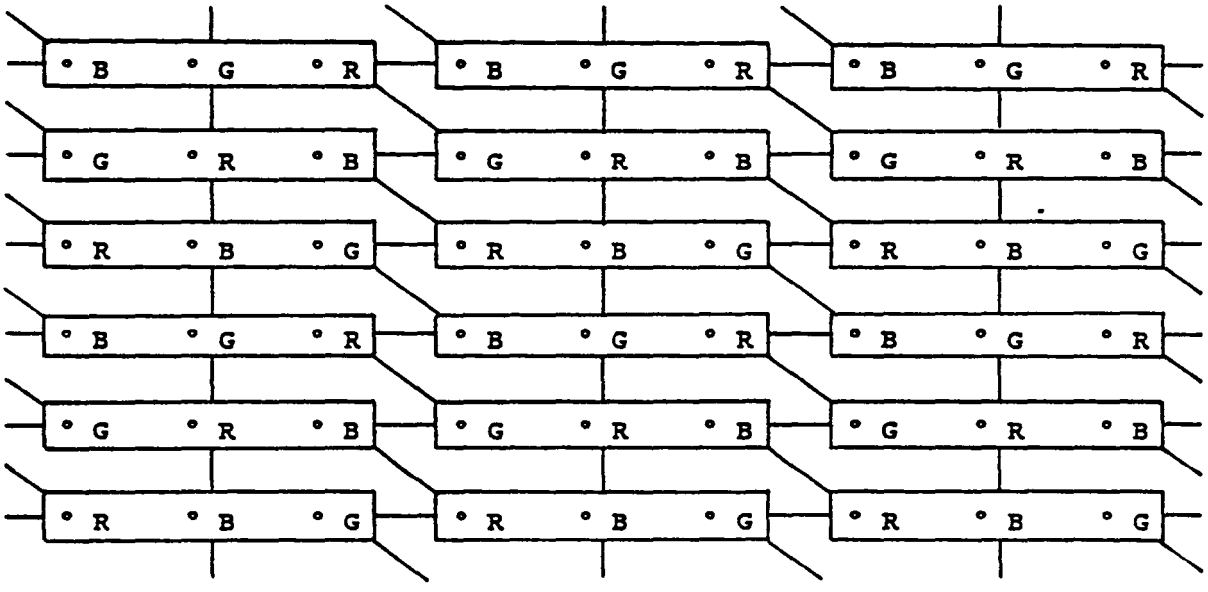
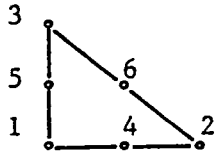


Figure 15. Processor Assignment for Figure 14.

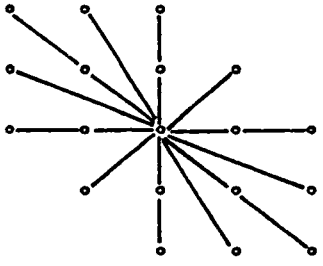
The local communication links of each FEM processor that are needed for this assignment are the North, South, East, West, Northwest, and Southeast links.

Next, consider a triangular element with C^0-piecewise quadratic basis functions defined at the vertices and midpoints of the triangle

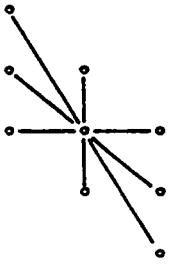
This element and its associated grid point stencil are illustrated in Figure 16.



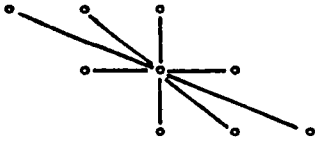
Quadratic Element



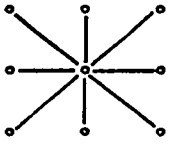
Stencil for Nodes 1,2, and 3



Stencil for Node 4



Stencil for Node 5



Stencil for Node 6

Figure 16. Quadratic Element and Grid Point Stencils

For this stencil, the gridpoints may be partitioned into six disjoint subsets with the colors Red, Black, Green, Orange, Yellow, and Purple as shown in Figure 17.

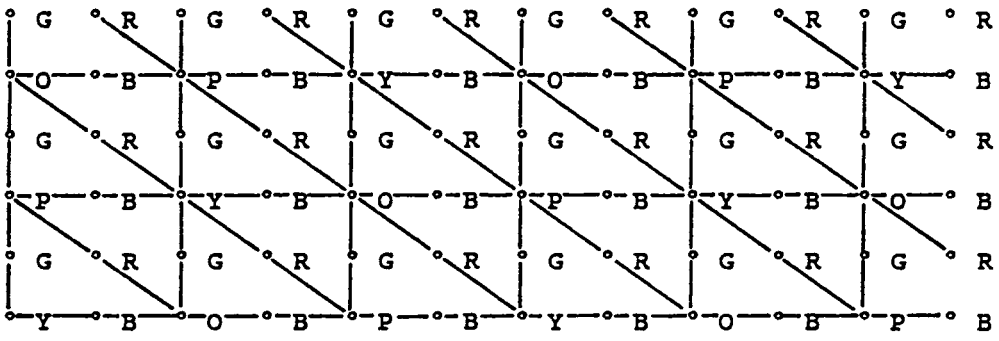


Figure 17. 6-Coloring for C^0 Quadratic Elements

The grid points are then assigned to processors in blocks of size $2k \times 6l$ with the minimum block of size 2×6 as shown in Figure 18 All eight local links are required for this processor assignment.

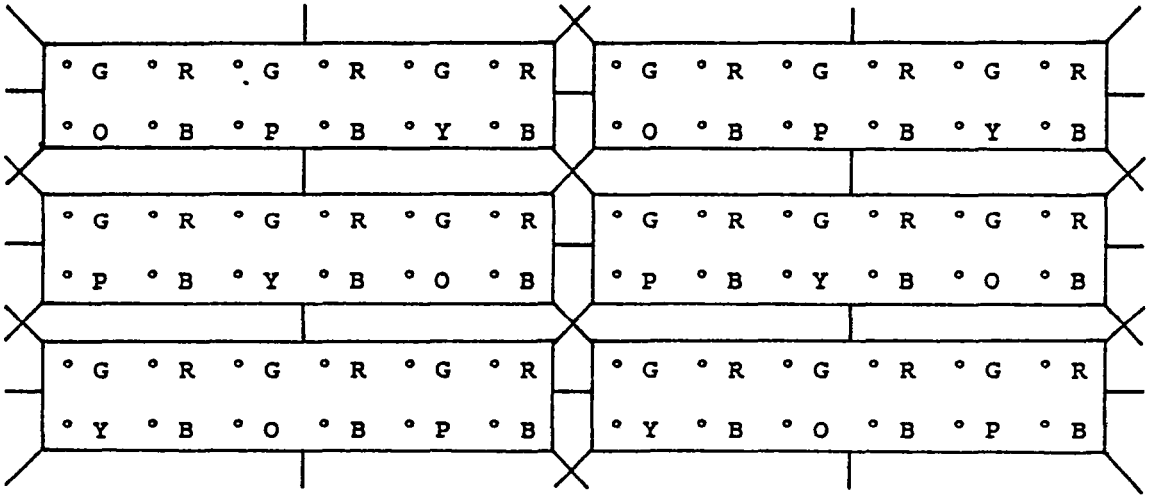


Figure 18. Processor Assignment for Figure 17.

We now consider two examples of higher order finite elements that are used to discretize 4th order partial differential equations. The first example is the C^1 (function and its first partials are continuous) bi-cubic rectangle (see Becker and Oden[1981]) A cubic in x and y can be uniquely

determined by 16 constants. Therefore, if we prescribe the values of the unknown at a grid point, u_h, its partials in x and y, $\frac{\partial u_h}{\partial x}$, and $\frac{\partial u_h}{\partial y}$ and its second partial $\frac{\partial^2 u^h}{\partial x\,\partial y}$ at the four corners of the rectangular element as shown in Figure 19.

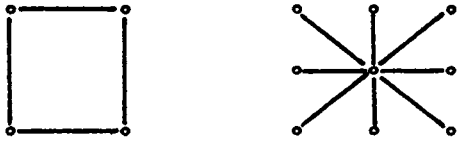


Figure 19. Bi-Cubic Rectangle and Grid Point Stencil

the basis functions at each grid point will be bi-cubic polynomials which will have continuous partials $\frac{\partial u^h}{\partial n}$ across element boundaries where n is the normal to a common side. The stencil in Figure 19 is the same stencil as the stencil of Figure 4, therefore Figures 5 and 6 give the appropriate coloring and processor assignment for grids that are discretized by this element.

Lastly, we consider the C^1 quintic triangle, Oden [1981], which is shown in Figure 20.

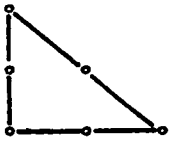


Figure 20. Quintic Triangle

A quintic basis function is defined at each grid point by specifying 21 values: the six values u_h, $\frac{\partial u_h}{\partial x}$, $\frac{\partial u_h}{\partial y}$, $\frac{\partial^2 u^h}{\partial x^2}$, $\frac{\partial^2 u_h}{\partial y^2}$, $\frac{\partial^2 u_h}{\partial x\,\partial y}$, and $\frac{\partial^2 u_h}{\partial y^2}$ at

each vertex of the triangle and the value of the normal derivative $\frac{\partial u_h}{\partial n}$ at the midpoints of each side of the triangle. The nodes in Figure 20 have the same connectivity as those of the C^0 quadratic triangular element and the stencils of Figure 16. Therefore, Figures 17 and 18 give the appropriate coloring and processor assignment for grids that are discretized by this element.

5.2.3. Comparison to Existing Theory

In this section we explain what is meant by a p-Colored matrix and show how matrices of this type relate to the consistently ordered (CO), the q-r consistently ordered (CO(q,r)), and the q-r generally consistently ordered (GCO(q,r)) matrices of Young[1971] and the p-cyclic matrices of Varga[1962]. For these matrices of Young and the p-cyclic consistently ordered matrices of Varga a well known theory exists for determining the optimum relaxation factor ω for the associated SOR iterative method. We show, *in general*, that p-Colored matrices do not fit into any of these classifications; however, CO, CO(q,r), and certain p-cyclic matrices can easily be permuted into a p-Colored matrix.

The notion of a p-Colored matrix is directly related to that of a p-partite graph as was given by Definition 1. Recall that nodes within the same subset of a p-partite graph are not connected by an edge; however, no restrictions are made on the number of nodes in a subset S_l that can have edge connections to nodes in subset S_j.

By numbering the equations associated with nodes in subset S_1, in any order, followed by the equations associated with nodes in subset S_2, S_3, ... , and finally subset S_p, the result is a *p-Colored* matrix K which has the following form:

$$K = \begin{bmatrix} D_{11} & X_{12} & . & . & X_{1p} \\ X_{21} & D_{22} & . & . & X_{2p} \\ . & . & . & . & . \\ . & . & . & . & . \\ X_{p1} & X_{p2} & . & . & D_{pp} \end{bmatrix} \tag{5.19}$$

where the D_{ii} are block diagonal matrices.

$$D_{ii} = \begin{bmatrix} E_1 & & & & \\ & E_2 & & & \\ & & . & & \\ & & & . & \\ & & & & E_{nc_i} \end{bmatrix}$$

with each entry E_i being a square matrix representing the equations at only one grid point of the associated problem domain that has been colored with p colors and nc_i nodes of color i

For the special case of one equation per grid point, say Laplace's equation for example, the D_{ii} will be diagonal matrices. As was noted in Chapter 2, the plane stress problem has two unknowns per grid point; consequently for this problem, the E_i will be 2x2 matrices and the 2 equations at the same node will have the same color.

We now compare p-Colored matrices with diagonal D_{ii} blocks to the CO(q,r) matrices of Young. Now, a test for determining whether a matrix K is a $CO(q,r)$ matrix is given by the following definitions and theorem by Young.

Definition 2. (Young)

For given positive integers q and r, the matrix K of order N is a (q,r)-consistently ordered matrix ($CO(q,r)$-matrix) if for

some t, there exists disjoint subsets $S_1, S_2, \cdots, S_t$ of $W=\{1,2,\ldots,N\}$ such that $\sum_{k=1}^{t} S_k = W$ and such that: if $k_{ij} \neq 0$ and $i<j$, then $i \in S_1 + S_2 + \cdots + S_{t-r}$ and $j \in S_{k+r}$, where S_k is the subset containing i; if $k_{ij} \neq 0$ and $i>j$, then $i \in S_{q+1} + S_{q+2} + \cdots + S_t$ and $j \in S_{k-q}$ where S_k is the subset containing i.

Definition 3. (Young)

The vector $\gamma=(\gamma_1, \gamma_2, \cdots, \gamma_n)^T$, where γ_i, $i=1,2,\ldots,N$ are integers, is a (q,r) compatible ordering vector for K if for any i and j such that $k_{ij} \neq 0$ then

$$\gamma_j - \gamma_i = r \qquad \text{if } i<j$$

and

$$\gamma_j - \gamma_i = -q \qquad \text{if } i>j$$

A CO(1,1) matrix is called CO, or consistently ordered.

Theorem 1. (Young)

The matrix K is a CO(q,r) matrix if and only if there exists a compatible odering vector for K

By using Theorem 1 it is very easy to conclude that the p-Colored matrix in (5.19) is in general not a CO(q,r) matrix. In particular, let

$$\begin{aligned} X_{12} &\neq 0 \\ X_{13} &\neq 0 \\ X_{23} &\neq 0 \end{aligned} \tag{5.20}$$

Then if a compatible ordering vector exists for (5.19), we must have

from Theorem 1 that

$$\gamma_2 - \gamma_1 = r$$

$$\gamma_3 - \gamma_1 = r$$

or

$$\gamma_3 - \gamma_2 = 0 \tag{5.21}$$

But, since $X_{32} \neq 0$, we must require

$$\gamma_3 - \gamma_2 = -q \tag{5.22}$$

Since (5.22) conflicts with (5.21), (5.19) is in general not a CO(q,r) matrix. The same technique can be used to show that the 4-Colored matrix for Figure 6, the 3-Colored matrix for Figure 7, the 6-Colored matrix for Figure 11, the 3-Colored matrix for Figure 14, the 6-Colored matrix for Figure 17, the 4-Colored matrix for the stencil of Figure 19, and the 6-Colored matrix for the stencil of Figure 22 are not CO(q,r) matrices.

On the other hand, if the matrix K is a CO(q,r) matrix, we show in the next theorem that K is permutationally similar to a p-Colored matrix. Before proving the theorem, we recall the following definitions of Young and Varga

Definition 4 (Young)

Given the positive integers q, r, and t, the matrix K is a $T(q,r,t)$ matrix if it can be partitioned into the txt block form $K=(K_{ij})$ where, for each i, $K_{ii}=D_i$ is a square diagonal matrix and where all other blocks vanish except possibly for the blocks $K_{i,i+r}$, $i=1,2,\ldots,t-r$, and $K_{i,i-q}$, $i=q+1,q+2,\ldots,t$

The matrix in (5.23) is an example of a $T(1,s,t)$ matrix.

$$\begin{bmatrix} D_1 & & H_{1+s} & & \\ K_1 & D_2 & & H_{2+s} & \\ & K_2 & \ddots & & \ddots \; H_t \\ & & \ddots & & \\ & & & K_{t-1} & D_t \end{bmatrix} \tag{5.23}$$

where the K_{ii} matrices are diagonal matrices.

Definition 5. (Varga)

Let K be partitioned as

$$K = \begin{bmatrix} K_{11} & K_{12} & \cdot & K_{1p} \\ K_{21} & K22 & \cdot & K_{2p} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ K_{p1} & K_{p2} & \cdot & K_{pp} \end{bmatrix} \tag{5.24}$$

If the Jacobi matrix $B=I-(diag\,(K))^{-1}K$ is permutationally similar to a T(1,p-1,p) matrix, then K is p-cyclic relative to the partitioning (5.24).

Theorem 2.

Let K be a CO(q,r) matrix and let $p'=(q+r)/d$ where d is the largest common factor of q and r. Then there exists a permutation matrix P such that $P^{-1}KP$ is a

(1) p'-Colored matrix

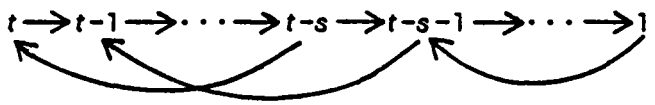
(2) 2-Colored matrix if p' is even

(3) 3-Colored matrix if p' is odd

Proof.

Since K is a CO(q,r) matrix, K has Property $A_{q,r}$ (see Young).

Therefore K also has Property $A_{1,p'}-1$. Now, there exists a permutation matrix P such that $P^{-1}KP$ is a CO(1,p'-1) matrix. Furthermore, Young shows that a CO(1,s) matrix is also permutationally similar to a T(1,s,t) matrix with possibly certain rows and corresponding columns of blocks deleted. Now, the adjacency graph associated with the Jacobi matrix for the T(1,s,t) matrix of (5.23) is shown below, where we denote all the variables associated with D_1 by 1, the variables associated with D_2 by 2,..., and finally those associated with D_t by t.



If we color these t blocks with p' colors from right to left as $C_1/C_2/\cdots/C_{p'}/C_1/C_2/\cdots/C_{p'}/C_1/$ etc. and group together all the blocks of the same color and then order the matrix by groups, the resulting matrix will have the form

$$K = \begin{bmatrix} D_1 & & & & X_{1p'} \\ X_{21} & D_2 & & & \\ & X_{32} & \ddots & & \\ & & \ddots & & \\ & & & X_{p',p'-1} & D_{p'} \end{bmatrix}$$

which is easily seen to be p'-Colored as well as p-cyclic and (1) follows.

We next prove statement (2) of the theorem. If s is odd, these t blocks can be colored R/B/R/B.. from right to left All the R blocks can be grouped together and the same for the B blocks

so that the resulting matrix has the form

$$K = \begin{bmatrix} D_1 & X_{12} \\ X_{21} & D_2 \end{bmatrix}$$

which shows that K is 2-Colored.

If s is even, the t blocks are colored R/B/G/R/B/G.. from right to left. Furthermore, if p' is a multiple of three, K has the form

$$K = \begin{bmatrix} D_1 & & X_{13} \\ X_{21} & D_2 & \\ & X_{32} & D_3 \end{bmatrix}$$

which is 3-Colorable and also 3-cyclic whereas, if p' is not a multiple of three, K has the form

$$K = \begin{bmatrix} D_1 & & X_{13} \\ X_{21} & D_2 & X_{23} \\ & X_{32} & D_3 \end{bmatrix}$$

which is not 3-cyclic but is 3-Colored. Hence, statement (3) of the theorem follows

We now compare p-Colored matrices to the p-cyclic matrices of Varga for the case where the D_{ii} matrices in (5.19) are diagonal. From the form of a T matrix given in (5.23), it is readily seen that a p-Colored matrix is not in general p-cyclic. On the other hand, if the matrix of (5.24) is p-cyclic it is also p-Colored In fact, we can use Theorem 1 to show that a p-cyclic matrix is permutationally similar to either a 2 or a 3 Colored matrix

Corollary 1.

Let K be a p-cyclic matrix with the K_{ll} matrices being diagonal matrices. Then there exists a permutation matrix P such that $P^{-1}KP$ is a

(1) 2-Colored matrix if p is even

(2) 3-Colored matrix if p is odd

Proof:

Since K is p-cyclic it is permutationally similar to a T(1,p-1,p) matrix. The conclusion follows directly from the proof of Theorem 2 after noting that $s=p-1$ and $t=p$.

Corollary 1 implies that p-cyclic matrices for which the diagonal blocks are diagonal can be reordered to yield 2 or 3 diagonal blocks on the diagonal. This means that for a vector implementation of the Multi-color SOR method, the associated vector $\underline{u}$ for the solution of (5.1) can be partitioned into 2 or 3 long vectors rather than p shorter ones. However, we note that the resulting 3-Colored matrix in (2) of Corollary 1 may not be 3-cyclic and hence no known theory exists to aid in the selection of the optimal relaxation factor ω. This fact is possibly offset by the much longer vectors that will result if $p>>3$. Barlow and Evans[1982] mentions that p-cyclic matrices may be colored with p colors but does not mention the possibility of fewer colors.

Lastly, we discuss the relationship of p-Colored matrices (again with the D_{ll} in (1) being diagonal blocks) to Young's generally consistently ordered, GCO(q,r), matrices. First, we give the definition of a GCO(q,r) matrix.

Definition 6 (Young)

A matrix K is a GCO(q,r) matrix if

$$det(\alpha^{q}L+\alpha^{-r}U-kD)$$

is independent of α for all $\alpha\neq 0$ and for all k where $D,-L,-U$ are the diagonal, strictly lower and strictly upper parts of K respectively

Definition 7. (Young)

A real matrix K of order N is an L- *matrix* if

$$k_{i,i}>0. \qquad i=1,2,.....N$$

and

$$k_{i,j}\leq 0. \qquad i\neq j. \qquad i,j=1,2,.....N$$

Young also gives the relationship between GCO(q,r) and CO(q,r) matrices in the following theorem

Theorem 3 (Young)

If K is an irreducible GCO(q,r) matrix which is an L matrix then K is a CO(q,r) matrix.

Hence, matrices which are both L and GCO(q,r) matrices are permutationaly similar to either a 2 or a 3-Colored matrix by Theorem 2 The 2-Colored matrix will be consistently ordered but the 3-Colored matrix may not be q-r consistently ordered as was shown in the proof of Theorem 2.

Since the matrix K is symmetric for our problems, we are interested in the relationship of symmetric GCO matrices to CO(q,r) matrices.

Lemma 1

If K is a symmetric CGO(q,r) matrix then $q=r$ and K is a GCO(1,1) matrix.

Proof:

Since K is a CGO(q,r) matrix, $\det(\alpha^q L+\alpha^{-r}U-kD)$ is independent of α for all $\alpha\neq 0$ and for all k. Recall that the determinant of an NxN matrix is the sum of N! terms of the form

$$t(\sigma)=s(\sigma)k_{1,\sigma(1)}k_{1,\sigma(2)}\cdots k_{N,\sigma(N)} \qquad (5\ 25)$$

where $s(\sigma)$ is 1 if the sequence $\sigma(1),\sigma(2),\ldots\sigma(N)$ can be put in the form $1,2,\ldots,N$ by an even number of interchanges of any pair of elements in the sequence and -1 otherwise.

Now, all the terms that are multiplied by α^{q-r} are of the form

$$s(\sigma)\alpha^{q-r}(-kd_1)(-kd_2)\ldots k_{ij}\ldots k_{ji}\ldots(-kd_n) \qquad (5\ 26)$$

where d_i is the ith entry of the diagonal of K and only k_{ij} and k_{ji} need to be interchanged for the sequence $\sigma(1)\sigma(2)\ldots\sigma(N)$ to be in the order 1,2,....,N. Hence, all these terms have $s(\sigma)=-1$. In addition, since $k_{ij}=k_{ji}$, all these terms are of the same sign and their sum can only be independent of α if α^{q-r} is independent of α which is true only if $q=r$. Now, $\det(\alpha^r L+\alpha^{-r}U-kD)$ can be written as

$$\det((\alpha^r)^1 L+(\alpha^r)^{-1}U-kD)$$

and is also independent of α for all $\alpha\neq 0$ and for all k. Therefore, we conclude that K is a GCO(1,1) matrix.

Definition 8.

A symmetric GCO(1,1) matrix is an SGCO matrix.

Next, we give the relationship between SGCO matrices and 2-Colored matrices.

Lemma 2.

Let K be an irreducible L matrix. If K is an SCGO matrix then there exists a permutation matrix P so that $P^{-1}KP$ is a 2-Colored matrix.

Proof:

From Theorem 3 it follows that K is a CO(1,1) or equivalently a consistently ordered (CO) matrix. It is well known that any CO matrix can be permuted to the R/B or 2-Colored form and the theorem follows.

The contrapositive of Lemma 2 states that if K is a symmetric L matrix that is not consistently ordered it can not be generally consistently ordered. This means that we can not simplify the determinant in Definition 6 for symmetric L matrices that are not consistently ordered in order to relate the eigenvalues of the Jacobi and SOR iteration matrices associated with K, and hence determine the optimum relaxation factor ω for the SOR iteration method. However, Lemma 2 only gives sufficient conditions for an SCGO matrix to be consistently ordered and it remains to be determined whether the requirement that K be an L matrix is necessary

5.2.4. Comparison with Rowwise Ordering

The Multi-color and lexiographical (rowwise) orderings were shown in the last section in general not be consistent orderings, therefore, we can not conclude that the eigenvalues of the respective SOR matrices are the same. The question then arises as to whether one ordering gives faster convergence than another However, we note that some degradation in the convergence rate of the Multi-color ordering can be permitted since it can be implemented effectively on a parallel machine whereas the rowwise ordering can not.

The Multi-color and rowwise orderings were compared experimentally for three problems. The first problem was the five-star discretization of Laplace's equation on a rectangular grid with 768 unknowns. The results for the R/B and rowwise ordering of the grid points are given in Table 1 Both these orderings are consistent and the results are included here for comparison with the next two example problems.

	Iterations	
ω	Red/Black	Rowwise
1.00	470	542
1.74	73	82
1.76	56	83
1.80	65	85

Table 1. Laplace's Equation (5-Star Discretization)
$\epsilon=10^{-6}$

The second problem was a finite element discretization of Laplace's equation. The finite elements were triangular with quadratic basis functions defined at the vertices and midpoints as shown in Figure 16 The width of each triangle was taken to be $h=1/12$ so that the resulting sys-

tem has $(23)^2$ equations. Table 2 gives the results for the 6-color ordering of Figure 17 and the rowwise ordering.

	Iterations		
ω	Rowwise	6-Color	5-Star (Rowwise)
1.00	563	561	463
1.20	394	392	324
1.40	266	264	218
1.60	161	158	132
1.70	113	109	91
1.76	83	76	64
1.77	76	69	57
1.78	69	60	57
1.79	62	54	59
1.80	68	58	59
1.82	75	66	66
1.84	83	73	75
1.86	94	82	97
1.88	109	97	98
1.90	117	121	117
1.92	161	147	146
1.94	194	197	195
1.96	291	302	293

Table 2. Laplace's Equation (Quadratic Elements)
$\epsilon=10^{-6}$, $(23)^2$ unknowns

For this problem, the 6-color ordering and the rowwise ordering for the finite element discretization give very similar results and the optimal values of ω are the same in both cases. In fact, near the optimum value of ω both the finite element discretizations require almost the same number of iterations as the 5-star finite difference discretization which is consistently ordered.

The third problem was the plane stress problem described in Chapter 2. The plate was discretized by linear triangular elements as shown in Figure 14. Table 3 gives results for the Red/Black/Green ordering of Figure 14 and the rowwise ordering of the gridpoints.

ω	Iterations R/B/G	Rowwise
1.4	349	347
1.5	265	263
1.6	169	167
1.61	153	152
1.62	131	128
1.621	129	126
1.622	127	124
1.623	142	140
1.63	149	148
1.64	147	145
1.65	141	138
1.66	135	133
1.67	156	154
1.68	155	154
1.69	153	150
1.7	150	148
1.8	233	232

Table 3. Plane Stress
ϵ=10-6, 60 unknowns

Note from Table 3 that the optimum value of ω is 1.622 for both orderings. Also, note that the number of iterations for ω>1.622 behaves differently than was seen from Table 2. For example, Table 2 showed that for $\omega>\omega_{opt}$ the number of iterations was strictly increasing whereas in Table 3 the graph of ω versus the number of iterations has relative minima at ω=1 66 for example

5.3. Multi-Color SSOR

In this section, we describe a Multi-color SSOR method, give an efficient implementation of this method on vector computers or parallel arrays, and give numerical comparisons to an SSOR method without multi-coloring for an example problem.

5.3.1. Description

The SSOR iterative method for solving (5.1) can be written as the forward SOR iteration followed by the backward SOR iteration

$$(D-\omega L)\underline{u}^{k+\frac{1}{2}} = [\omega U+(1-\omega)D]\underline{u}^{k} + \underline{b}$$

$$(D-\omega U)\underline{u}^{k+1} = [\omega L+(1-\omega)D]\underline{u}^{k+\frac{1}{2}} + \underline{b} \qquad (5.27)$$

The basic convergence theorem for SSOR iterative method (Young[1971]) is stated below.

SSOR Convergence Theorem

> If K is a symmetric matrix with positive diagonal elements, the SSOR method converges if and only if K is positive definite and $0<\omega<2$.

The SSOR method is therefore convergent for symmetric and positive definite matrices K. Even so, this method has been found to have a slower convergence rate than the SOR method for 2-Colored matrices. Therefore, our interest in this method is as a preconditioner for a parallel conjugate gradient method, as will be described in Chapter 6, and not as a stand alone linear stationary method. However, even for our purposes, a parallel implementation of this method is necessary.

5.3.2. Parallel SSOR Implementation

To solve (5.27) on a vector computer or a parallel array the equations are first ordered so that K is a p-Colored matrix with colors C_1, C_2, ... , and C_p. Then the Multi-color SOR method is first applied to

K in a forward fashion, starting with the updating of color C_1, followed by C_2, until the equations of color C_p are updated. Next, the Multi-color SOR method is applied to K in a reverse fashion starting with color C_p, followed by C_{p-1} until the equations of color C_1 are updated. After the reverse SOR pass is completed, and hence one SSOR iteration, if the convergence test is met, the iteration stops, otherwise, the process is repeated. For parallel arrays, after the values of each color C_i are updated on both the forward and reverse pass they must be communicated between neighbor processors. The Multi-color SSOR algorithm is given below:

For k=1,2.....k_{max} do

(1) For c=1,2.....nc do

(1) Solve for $\underline{u}_{c,p}^{k+1}$

(2) Send necessary portion of $\underline{u}_{c,p}^{k+1}$ to logical neighbors

(3) Receive $\underline{u}_{c,n}^{k+1}$ from logical neighbors.

(2) For c=nc,nc-1.....1 do

(1) Solve for $\underline{u}_{c,p}^{k+1}$

(2) Send necessary portion of $\underline{u}_{c,p}^{k+1}$ to logical neighbors.

(3) Receive $\underline{u}_{c,n}^{k+1}$ from logical neighbors.

(3) If $\left\| \underline{u}_{c,p}^{k+1} - \underline{u}_{c,p}^{k} \right\|_{\infty} < \epsilon$ set the convergence flag.

(4) If all processors have convergence flag set then stop.

Algorithm 4 Multi-color SSOR

Each iteration of the Multi-color SSOR method can be computationally expensive since it is comprised of two Multi-color SOR iterations

We now describe how to save 50% of this computational effort in the solution of (5.27) by using an auxiliary storage vector. This observation is due to Conrad and Wallach [1979] Recall that the equations to be solved to carry out one SSOR iteration (with $\omega=1$ and $D=I$ for simplicity) are

$$(I-L)\hat{\underline{u}}^{(j+\frac{1}{2})} = U\hat{\underline{u}}^{(j)} + \underline{b}$$

$$(I-U)\hat{\underline{u}}^{(j+1)} = L\hat{\underline{u}}^{(j+\frac{1}{2})} + \underline{b} \qquad (5.28)$$

The algorithm of Conrad and Wallach for doing multiple steps of SSOR is given below.

(1) Form $U\hat{\underline{u}}^{(o)}$ and store in $\underline{y}$.
(This takes zero operations if the initial guess is zero.)

(2) For $k=1,2,\ldots,k_{max}$

(3) Solve $(I-L)\hat{\underline{u}}^{(k+\frac{1}{2})} = \underline{y}+\underline{b}$ as a forward Multi-color SOR pass.

Store $L\hat{\underline{u}}^{(k+\frac{1}{2})}$ in $\underline{y}$.

(4) Solve $(I-U)\hat{\underline{u}}^{(k+1)} = \underline{y}+\underline{b}$ as a backward Multi-color pass.

Store $U\hat{\underline{u}}^{(k+1)}$ in $\underline{y}$.

If α and β denote the number of nonzeroes in L and U respectively then (3) requires α multiplications and (4) requires β multiplications. If η represents the maximum number of nonzeroes per row of K,

and m represents the number of multiplications per iteration of SSOR, then $\alpha+\beta<(\eta-1)N$ and

$$\begin{aligned} m &< (\eta-1)N \qquad \text{if } \omega=1 \\ m &< (\eta+1)N \qquad \text{if } \omega\neq 1 \end{aligned} \tag{5.29}$$

5.3.3. Comparison with Rowwise Ordering

It is well known, see Young[1971], that the SSOR method applied to a 2-Colored matrix has optimum relaxation factor $\omega=1$, whereas, if the grid points are ordered rowwise from bottom to top, left to right, the SSOR method converges faster for some $\omega_{opt}\neq 1$. It is an interesting question whether the same behavior will be seen for p-Colored matrices. We solved the plane stress problem of Chapter 2 (60 equations) with the R/B/G ordering of Figure 14 as well as the rowwise ordering. The results are in Table 4

Iterations		Iterations	
ω	Rowwise	ω	R/B/G
.90	589	.950	762
1.00	530	.990	759
1.20	467	.993	759
1.25	463	.994	758
1.30	463	.995	758
1.35	469	.997	758
		.998	758
		1.000	758
		1.050	761
		1.100	772
		1.200	815

Table 4. SSOR Results (R/B/G and Rowwise Orderings)
Plane Stress Problem (60 equations)

Table 4 shows that $.994<\omega<1.01$ produces the best results for the R/B/G ordering for the SSOR method, whereas, $\omega=1.25$ gave optimal results for the rowwise ordering. This suggests that the SSOR iterative

method for p-Colored matrices has optimal relaxation factor ω=1 as is true for 2-Colored matrices. However, this conjecture has yet to be proved or verified experimentally with more examples of p-Colored matrices. We note that even if this were true, the Multi-color SSOR method, with ω=1, can be implemented effectively on vector computers and parallel arrays whereas the rowwise ordering can not. In addition, using ω=1 alleviates the need to estimate the value of ω which may be a time consuming process since little theory exists to aid in this choice for matrices that are not 2-Colored.

5.4. Parallel Block Iterative Methods

In this section we consider the implementation of block iterative methods on vector computers and parallel arrays. In Section 5.4.1 we describe the implementation of the Block Jacobi iterative method. In Section 5.4.2 we discuss the difficulties in implementing the Block SOR method and in Section 5.4.3 we generalize the Multi-color orderings of Section 5.2.2 and the p-Colored matrices of Section 5.2.3 to Block Multi-color orderings and p-Block Colored matrices. Lastly, we compare the p-Block Colored matrices to the π-consistently ordered (π-CO) matrices of Young[1971] and the p-cyclic matrices of Varga[1962].

5.4.1. The Block Jacobi Method

Let K be a pxp block matrix as shown in (5.24) and let the vectors $\underline{u}$ and $\underline{f}$ be partitioned as $\underline{u}=(\underline{u}_1,\underline{u}_2,\cdots,\underline{u}_p)^T$ and $\underline{f}=(\underline{f}^1,\underline{f}_2,\cdots,\underline{f}_p)^T$ respectively. Furthermore, let

$$D = \begin{bmatrix} K_{11} & & & \\ & K_{22} & & \\ & & \ddots & \\ & & & K_{pp} \end{bmatrix} \tag{5 30}$$

$$-L = \begin{bmatrix} 0 & & & & \\ K_{21} & 0 & & & \\ \vdots & \vdots & 0 & & \\ \vdots & \vdots & \vdots & 0 & \\ K_{p1} & K_{p2} & \cdots & K_{p,p-1} & 0 \end{bmatrix} \quad -U = \begin{bmatrix} 0 & K_{12} & \cdots & \cdots & K_{1p} \\ & 0 & \ddots & \ddots & \vdots \\ & & 0 & \ddots & \vdots \\ & & & 0 & K_{p-1,p} \\ & & & & 0 \end{bmatrix}$$

Then the Block Jacobi method for solving (5.1) is

$$D\underline{u}^{k+1} = (L+U)\underline{u}^{k} + \underline{f} \tag{5 31}$$

or

$$\underline{u}^{k+1} = B\underline{u}^{k} + \underline{c} \tag{5.32}$$

where

$$B = D^{-1}(L+U)$$
$$\underline{c} = D^{-1}\underline{f}$$

and B is called the Block Jacobi iteration matrix. The Block JOR method is iteration (5.32) with B replaced by B_ω where

$$B_\omega = \omega B + (1-\omega)I \tag{5 33}$$

Now, the iteration (5.31) can be written in implementation form as

$$K_{ii}\underline{u}_i^{k+1} = \underline{f}_i - [\sum_{j=1}^{i-1} K_{ij}\underline{u}_j^{k} + \sum_{j=i+1}^{p} K_{ij}\underline{u}_j^{k}] \qquad (5.34)$$

Note that if the K_{ii} are diagonal matrices, (5.34) is just the Jacobi iteration method (5.3), but if the K_{ii} are not diagonal, p systems of equations must be solved each iteration, one for each $\underline{u}_i, i=1,2,\ldots,p$. However, these systems completely uncouple and hence can be solved simultaneously on parallel architectures.

On vector computers, the right hand side of (5.34) can be formed as matrix vector products and vector additions and the solution of the p systems of equations is vectorizable (Buzbee,Boley,Parter[1979]) with the vector length equal to p. On arrays with p processors, (5.34) is easily implemented by assigning processor i to the calculation of $\underline{u}_i$. Once $\underline{u}_i^{k+1}$ is calculated, the appropriate components are sent to neighbor processors and the appropriate components of $\underline{u}^{k+1}$ are received from neighbors for use in the next iteration. The p processors then complete the calculation of one iteration in the time it takes the processor with the most unknowns to complete its calculation. If each processor has the same speed and the same number of unknowns, $O(p)$ speedup can be achieved with this approach.

5.4.2. The Block SOR Method

The Block SOR method for solving (5.1) is

$$\frac{1}{\omega}(D-\omega L)\underline{u}^{k+1} = \frac{1}{\omega}[\omega U+(1-\omega)D]\underline{u}^{k} + \underline{f} \qquad (5.35)$$

or

$$\underline{u}^{k+1} = L_\omega \underline{u}^{k} + \underline{c} \qquad (5.36)$$

where

$$L_\omega = (D-\omega L)^{-1}[\omega U+(1-\omega)D]$$
$$\underline{c} = \omega(D-\omega L)^{-1}\underline{f}$$

and L_ω is the Block SOR iteration matrix.

The implementation form of (5.35) is given by

$$K_{ll}\underline{u}_l^{k+1} = \underline{f}_l - [\sum_{j=1}^{l-1} K_{lj}\underline{u}_j^{k+1} + \sum_{j=l+1}^{p} K_{lj}\underline{u}_j^{k}] \qquad (5.37)$$

and $\underline{u}_p^{k+1}, l=1,2,....,p$ is solved in sequence, first $\underline{u}_1$, followed by $\underline{u}_2$ ..... and finally $\underline{u}_p$.

The algorithm given by (5.37) is sequential and can not be completely vectorized or implemented on parallel arrays. However, it is well known that for some discretizations of partial differential equations a reordering of the grid points results in a block matrix for which the equations in (5.37) uncouple. In particular, consider the grid point stencil of Figure 4. If we color the even rows of points Red and the odd rows of points Black as shown in Figure 21.

```
° R     ° R     ° R     ° R
° B     ° B     ° B     ° B
° R     ° R     ° R     ° R
° B     ° B     ° B     ° B
° R     ° R     ° R     ° R
° B     ° B     ° B     ° B
```

Figure 21. Line Red/Black Ordering

group all points in a given row into one block, and then number the red blocks first from bottom to top, followed by black blocks, the matrix

K for Figure 21 has the form

$$K = \left[\begin{array}{ccc:ccc} K_{11} & & & X & X & \\ & K_{22} & & & X & X \\ & & K_{33} & & & X \\ \hdashline X & & & K_{44} & & \\ X & X & & & K_{55} & \\ & X & X & & & K_{66} \end{array}\right] \tag{5.38}$$

The SOR iteration (5.37) is the classical Red/Black line SOR which is composed of two block Jacobi sweeps, one on the Red blocks, followed by one for the Black blocks. The implementation of this method on vector computers is discussed by several authors (Buzbee,Boley,Parter[1979],Nolen[1979],Parter and Steuerwalt[1980], Saad and Sameh[1981]). For parallel arrays, every $2k$ rows of points are assigned to each processor as shown in Figure 22 for $k=1$.

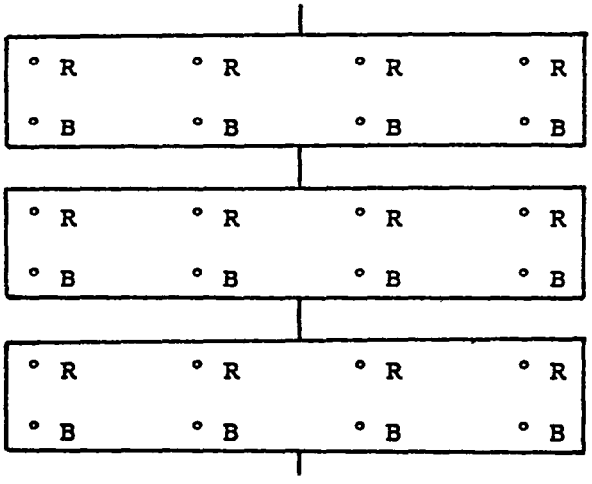


Figure 22. Processor Assignment for Figure 21.

For the assignment in Figure 22, processor i first updates the Red block

of unknowns, communicates these values to processor $i+1$, updates the Black block of unknowns and communicates these values to processor $i-1$ and then checks the convergence of the process. This algorithm is executed in all processors with slight modifications in processor 1 and processor p. If the grid contains p rows, a speedup of $O(p/2)$ is achieved by this scheme.

The same Red/Black line SOR method can be used for the linear triangular finite element discretization of Figure 13 and the bi-cubic rectangle of Figure 19. However, for the 9-point discretization in Figure 7, the 13-point discretization of Figure 16 and the quintic triangle in Figure 20, a Red/Black 2-line SOR method can be used. For this scheme, we color the first bottom two rows Black, the next two rows Red, etc. as shown in Figure 23.

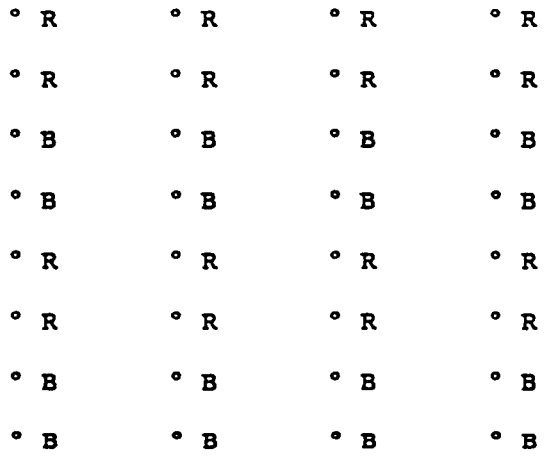


Figure 23. Red/Black 2-line Ordering

and then group every two rows of Red points into one block and the same for the Black points. If the Red blocks are numbered from bottom to top followed by the Black blocks, the resulting matrix K for Figure 23

will have the form (5.38) with 4 diagonal blocks instead of 6. The p rows of the problem grid are assigned to $p/4$ processors as shown in Figure 24

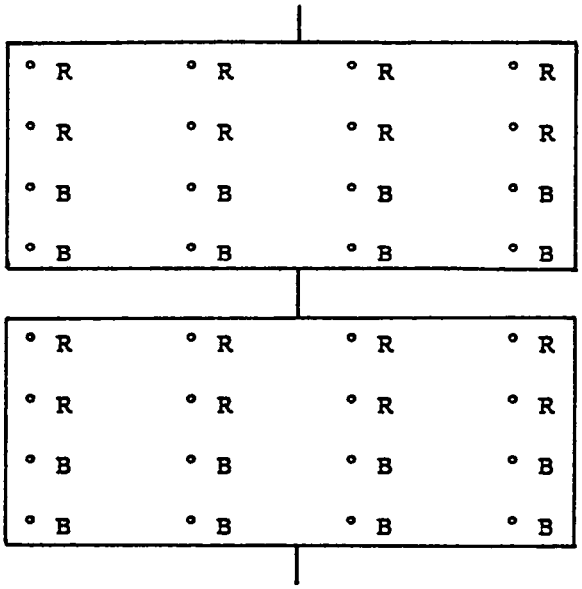


Figure 24. Processor Assignment for Figure 23.

With this assignment, a speedup of $O(p/4)$ is obtained

It is well known that the K matrix assoicated with the Red/Black k-line orderings is π-consistently ordered and has the form (5.39)

$$K = \begin{bmatrix} D_r & X_{12} \\ X_{21} & D_b \end{bmatrix} \tag{5.39}$$

where D_r and D_b represent the connectivity of the Red points to each other and the connectivity of the Black points to each other respectively (see Young[1971]) In this case, there is a theory for the selection of the relaxation parameter ω for the associated Block SOR method which is briefly summarized below

Definition 9. (Young)

Let K be partitioned as in (5.24) and define a pxp matrix Z with elements z_{ij} by

$$z_{ij} = \begin{cases} 0 & \text{if } k_{ij}=0 \\ 1 & \text{if } k_{ij}\neq 0 \end{cases}$$

Then K is π-consistently ordered (π-CO) if Z is consistently ordered.

Theorem 4. (Young)

Let K be a positive definite π-CO matrix. Then

(1) $\rho(B^{(\pi)}) < 1$

(2) $\omega_b^{\pi} = 2/[1-(1-\rho(B^{(\pi)})^2]^{\frac{1}{2}}$

where $B^{(\pi)}$ is the Jacobi iteration matrix associated with the partitioning in (5.24).

5.4.3. The Block Multi-Color SOR Method

In the last section we showed how to implement either a 1 or a 2 line SOR method on parallel arrays for all the discretizations in Section 5.2.2. This algorithm has the advantages that a theory exists for determining the optimum relaxation factor ω even though in practice the spectral radius of the Block Jacobi method may not be known in advance. A major drawback of this implementation arises when the number of processors p greatly exceeds $n/2$ and $n/4$ for the 1 and 2 line methods respectively where n represents the number of rows in the problem grid. In particular, these speedups are only $n/2$ and $n/4$, or

equivalently, $\sqrt{N}/2$ and $\sqrt{N}/4$ when the number of unknowns is $N=n^2$. In this section, we propose an alternative blocking of the grid points that will give much better speedup results on a parallel array.

5.4.3.1. Block Multi-Color Orderings

As a first example, we consider the 9-point stencil of Figure 4. If we color the problem grid into Red/Black/White/Orange blocks as shown in Figure 25,

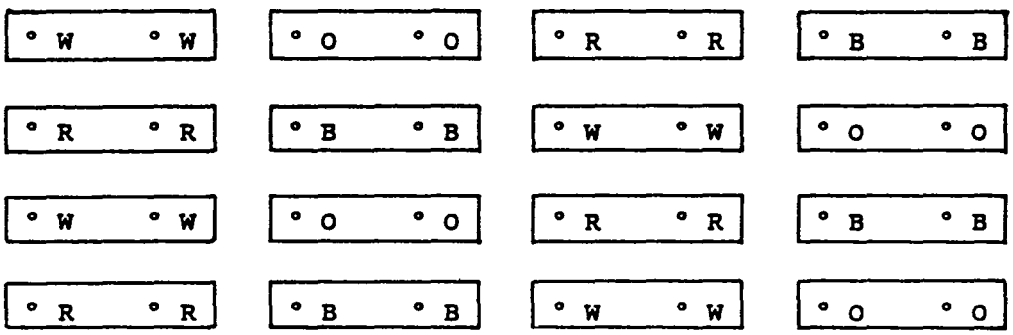


Figure 25. 4-Block Coloring for Figure 4

two blocks of the same color are not adjacent and hence the solution for blocks of unknowns of the same color in (5.37) completely uncouple. The blocks are assigned to processors in sizes $2k \times 4l$ so that each processor has the same number of blocks of each color as shown in Figure 26.

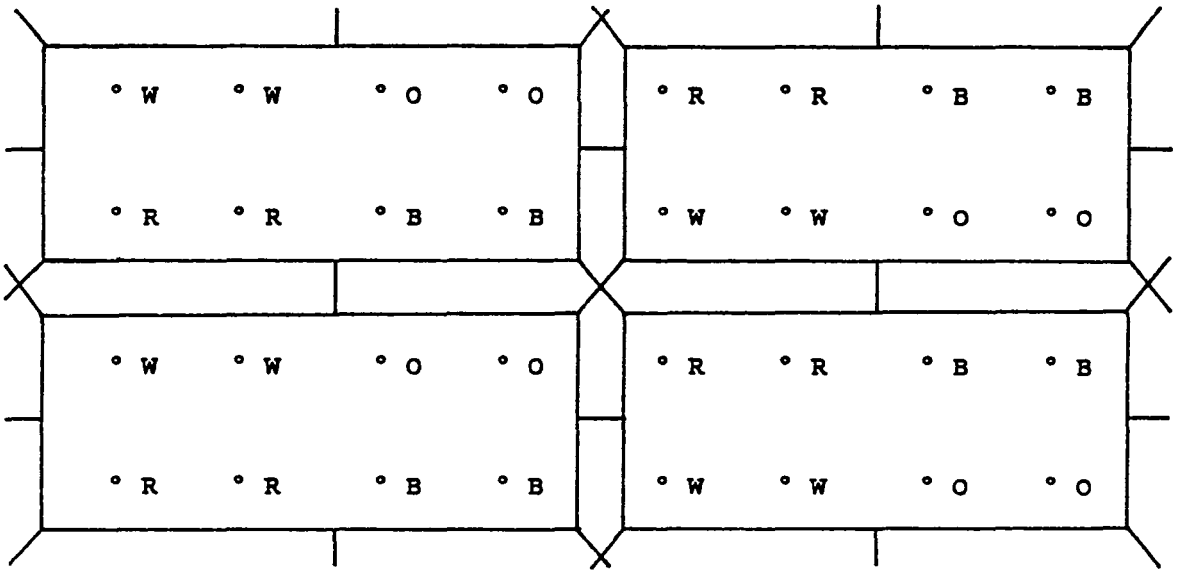


Figure 26. Processor Assignment for Figure 25.

The color pattern and processor assignment repeats beyond the subregion shown. For this assignment with n^2 grid points, $p=n^2/8$ and the maximum speedup that can be achieved is $n^2/8$.

If the Red blocks in Figure 25 are numbered first, followed by the Black blocks, then the Orange and finally the White blocks, the matrix K will have the form

$$K = \begin{bmatrix} D_{11} & X_{12} & X_{13} & X_{14} \\ X_{21} & D_{22} & X_{23} & X_{24} \\ X_{31} & X_{32} & D_{33} & X_{34} \\ X_{41} & X_{42} & X_{43} & X_{44} \end{bmatrix} \tag{5.40}$$

where the matrix D_{ii} is a $nc_i x nc_i$ block diagonal matrix of the form

$$D_{ll} = \begin{bmatrix} D_{l,1} & & & \\ & D_{l,2} & & \\ & & \ddots & \\ & & & D_{l,nc_l} \end{bmatrix} \tag{5.41}$$

and nc_l is the number of blocks of color l and $D_{l,j}$ represents the connectivity of nodes of the jth block of color l to each other. Note that nodes in two distinct blocks of the same color are not connected; whereas, nodes in the same block may be connected. Matrices which have the form (5.40) and (5.41) will be called *p-Block Colored matrices.*

As a second example, consider the 9-point discretization of Figure 7. The points are colored into Red/Black/Green blocks as shown in Figure 27.

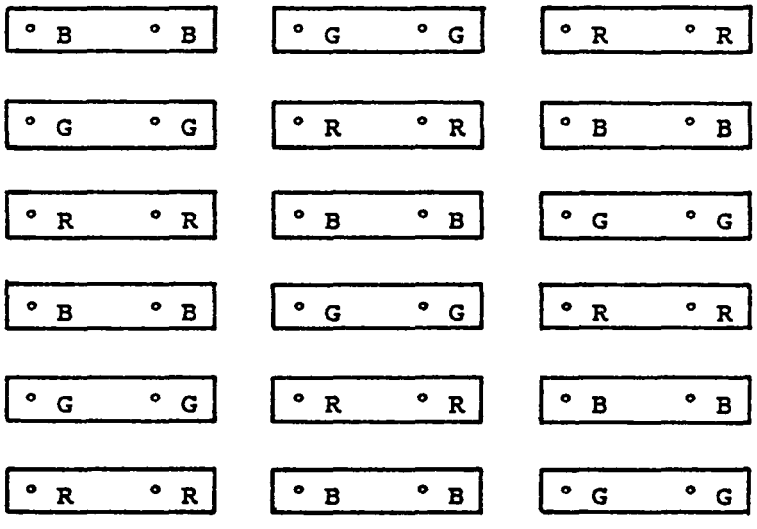


Figure 27. 3-Block Coloring for Figure 7.

The processors are assigned in blocks of size $3k \times 2j$ as shown in Figure 28 for $k=1$ and $j=1$ and a speedup of $O(n^2/6)$ is expected. Note that only four local communication links are used for each processor;

whereas for the coloring and assignment of Figure 8 and 9 respectively, four links plus four more for the next North, next South, next East, and next West processors was required to implement the point R/B/G SOR method.

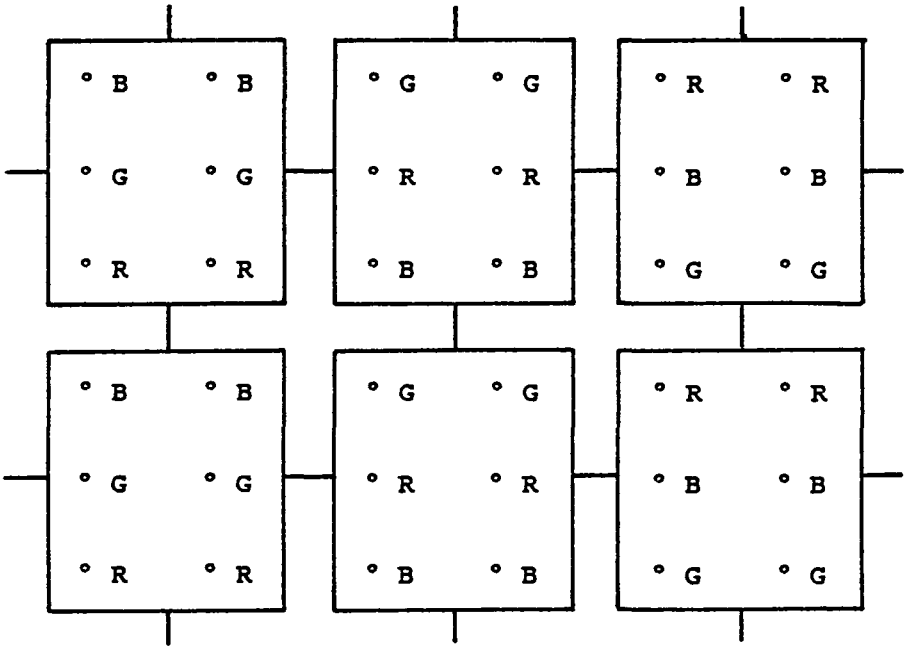


Figure 28. Processor Assignment for Figure 27.

As a last example of finite difference discretizations, consider the 13-point discretization of Figure 10. The points are colored into blocks with six colors, Red/Black/White/Orange/Purple/Yellow as shown in Figure 29

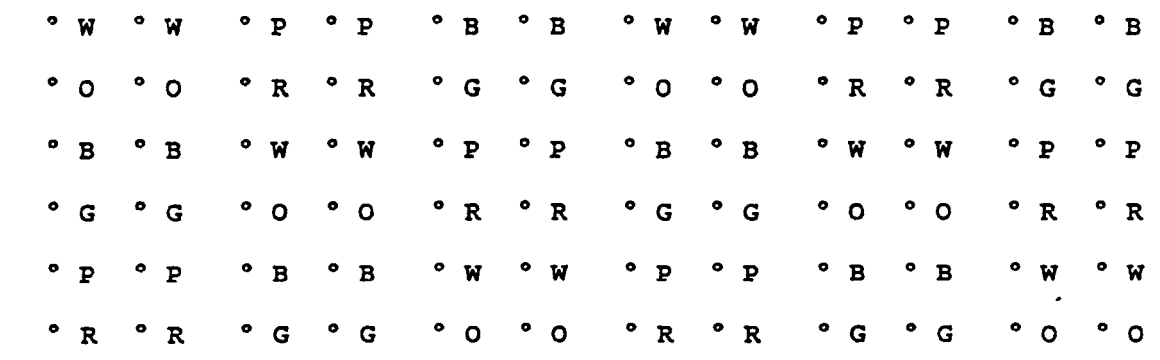


Figure 29. 6-Block Coloring for Figure 10.

The blocks are assigned to processors in sizes of $2k \times 6j$ as shown for $k=1$ and $j=1$ in Figure 30 and a speedup of $O(n^2/12)$ is expected. The coloring and processor assignment repeats beyond the subregion shown.

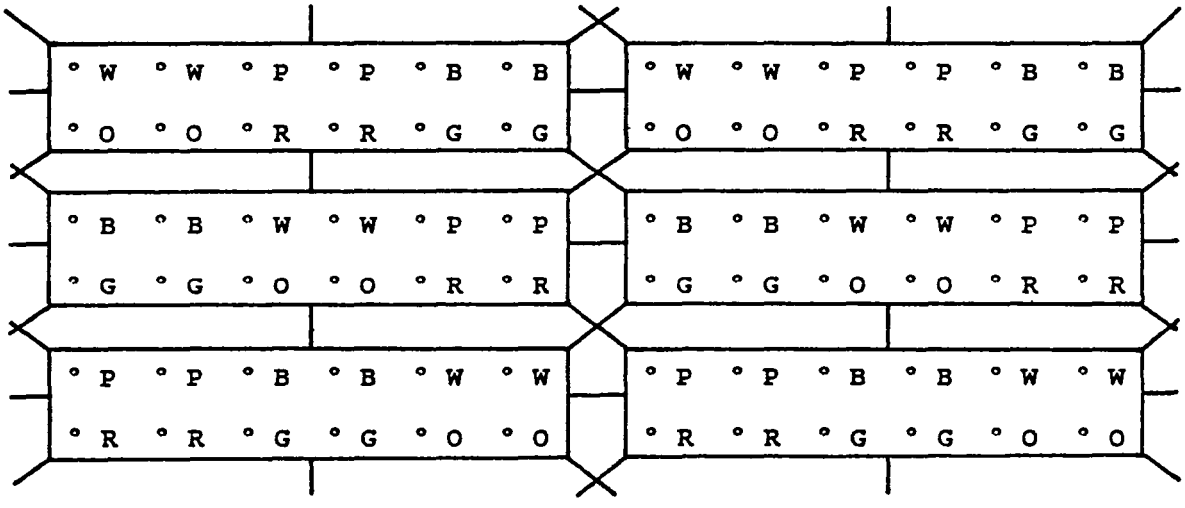


Figure 30. Processor Assignment for Figure 29.

We now consider the block orderings and processor assignments for the finite element discretizations of Section 5.2.2. The linear triangular element discretization of Figure 13 can be colored into Red/Black/Green blocks as shown in Figure 27 with the associated processor assignment of Figure 28. The quadratic triangular element discretization of Figure

16 can be colored with six colors. Red/Black/Green/White/Orange/Purple as shown in Figure 31.

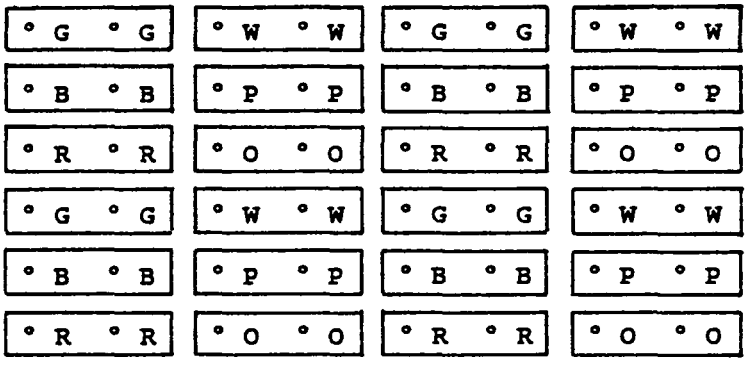


Figure 31. 6-Block Coloring for Figure 16.

The blocks are assigned to the processors in sizes $3k \times 4l$ as shown in Figure 32 for $k=1$ and $l=1$ and a speedup of $O(n^2/12)$ is expected. Note that only six local communication links for the interior processors are used for this implementation.

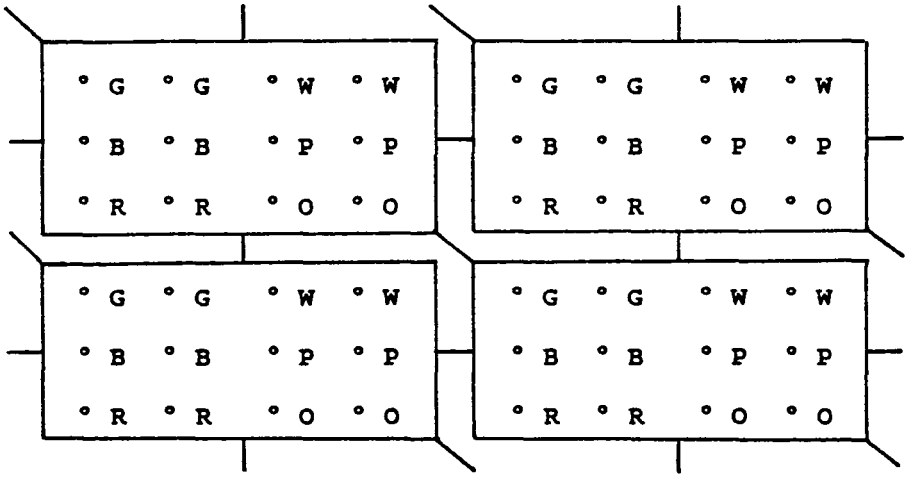


Figure 32. Processor Assignment for Figure 31.

All the examples of block colorings in this section lead to K p-Block Colored matrices. From (5.40) and (5.41) we can easily see that in general, p-Block Colored matrices are not π-CO matrices. On the other hand, it is a trivial observation that π-CO matrices are always permutationally similar to a 2-Block Colored matrix.

It is also easy to conclude that p-Block Colored matrices are not, in general, p-cyclic (relative to the partitioning (5.24)) matrices of Varga. On the other hand, it is an immediate generalization of Corollary 1 that p-cyclic matrices relative to (5.24) are permutationally similar to either a 2-Block or 3-Block Colored matrix.

We acknowledge that in general no theory exists as of yet to help in determining the relaxation factor ω for p-Block Colored matrices when $p>2$, but the extra parallelism that can be obtained over a k-line SOR method may far outweigh this disadvantage.

CHAPTER 6

Parallel Conjugate Gradient Methods

6.1. The Conjugate Gradient Method

The conjugate gradient (CG) method was proposed in 1952 by Hestenes and Stiefel[1952] as a method for solving a symmetric positive definite *NxN* system of linear equations. Although it is an iterative method in nature, it will converge in at most *N* steps in the absence of rounding error and hence may be viewed as a direct method.

In practice, however, the method was found to take many more than *N* steps due to this rounding error and was not competitive with Gaussian elimination But in 1971, Reid[1971] showed that the method could sometimes be used effectively as an iterative procedure for large sparse systems since suitable convergence may occur in far fewer than *N* steps. Several derivations and descriptions of this procedure appear in the literature: see for example, Chandra[1978] who studied the method for both finite element and finite difference discretizations of elliptic partial differential equations, and Schultchen and Kostem[1973] who recommend the method for solving the linear systems that arise from finite element discretizations Schreiber[1983] discussed the implementation of the CG method for vector computers and Podsiadlo and Jordan[1981] describe its implementation on the FEM. We give the algorithm below and review some of its implementation considerations on an array processor such as the FEM

(1) Choose $\underline{u}^0$

(2) $\underline{r}^0 = \underline{f} - K\underline{u}^0$

(3) $\underline{p}^0 = \underline{r}^0$

(4) $k = 0$

(5) For $k = 0, 1, \ldots, k_{max}$

(1) $\alpha = \dfrac{(\underline{r}^k, \underline{r}^k)}{(\underline{p}^k, K\underline{p}^k)}$

(2) $\underline{u}^{k+1} = \underline{u}^k + \alpha \underline{p}^k$

(3) *If* $\| \underline{u}^{k+1} - \underline{u}^k \|_\infty < \epsilon$ *then stop, otherwise continue.*

(4) $\underline{r}^{k+1} = \underline{r}^k - \alpha K \underline{p}^k$

(5) $\beta = \dfrac{(\underline{r}^{k+1}, \underline{r}^{k+1})}{(\underline{r}^k, \underline{r}^k)}$

(6) $\underline{p}^{k+1} = \underline{r}^{k+1} + \beta \underline{p}^k$

Algorithm 1. Conjugate Gradient Algorithm

In the above, (x, y) denotes the inner product $\underline{x}^T \underline{y}$

This algorithm can be implemented on an array computer with p processors like the FEM by partitioning the K matrix by rows into p portions, where each portion consists of at most $[\frac{N}{p}]$ rows. The vectors $\underline{u}$, $\underline{r}$, $\underline{p}$, and $\underline{f}$ are likewise partitioned by rows in the same manner The ith portion of each data structure is assigned to processor i as illustrated below for $p = 3$.

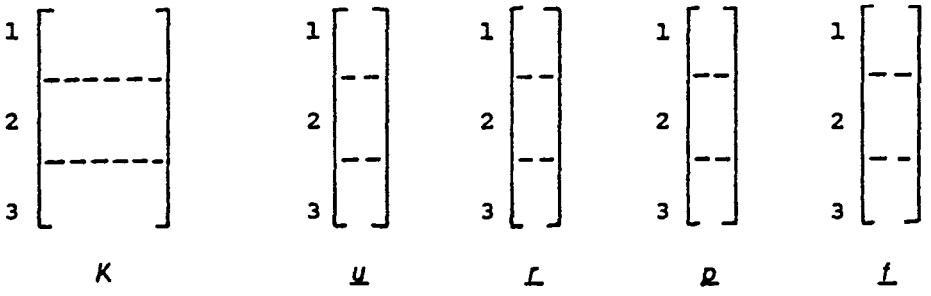


Figure 1. Data Assignment to 3 Processors

An examination of the CG algorithm as described above leads to the following observations:

(1) Once α is known, all processors can calculate their portion of $\underline{u}^{k+1}$ simultaneously with no communication required.

(2) Once β is known, all processors can calculate their portion of $\underline{p}^{k+1}$ simultaneously with no communication required

(3) Once $K\underline{p}^{k}$ and α are calculated, all processors can calculate their portion of $\underline{r}^{k+1}$ simultaneously with no communication required.

(4) Some components of $\underline{p}^{k}$ residing in other processors will be needed for the calculation of $K\underline{p}^{k}$ This means that the values of $\underline{p}^{k}$ for the non-interior nodes must be communicated between processors. This corresponds to the communication of the $\underline{u}^{k}$ values during a Jacobi or Multi-color SOR iteration as described in Chapter 5.

(5) The calculations of α and β require inner products to be formed globally over the array computer. Each processor can calculate the partial sum that corresponds to its portion of rows, but these partial sums must then be added together. If each processor

were to broadcast its partial sum to every other processor, the number of values received by a single procesor is $O(p-1)$ for one inner product alone.

This aspect of the CG algorithm was realized by Jordan[1979] to be detrimental to the performance of CG on an array computer and as a result the sum/max hardware circuit discussed in Chapter 3 was designed for the FEM to perform sums over the p processors. With this hardware, one inner product can be performed in $O(\log_2 p)$ operations since each processor will load its partial sum onto the circuit, and the circuit will perform the sum in a binary tree fashion and then return the complete sum to each processor.

6.2. Preconditioned Conjugate Gradient Methods

6.2.1. The PCG Algorithm

The condition number of any nonsingular matrix K with respect to a given norm is

$$\kappa(K) = \|K\| \ \|K^{-1}\| \tag{6.1}$$

In particular, if K is symmetric with eigenvalues λ_i then in the spectral (i.e. l_2) norm

$$\kappa(K) = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|} \tag{6.2}$$

The standard analysis of the conjugate gradient method, Chandra[1978], shows that the error in the ith iterate is bounded by

$$\| \underline{u}-\underline{u}^{i} \|_2 < 2\sqrt{\kappa(K)}\left(\frac{1-\alpha}{1+\alpha}\right)^{i} \| \underline{u}-\underline{u}^{o} \|_2 \tag{6.3}$$

where $\alpha=\frac{1}{\kappa(K)}$

This bound shows that the error is a decreasing function of the condition number of K. Hence, the conjugate gradient method applied to a system $\hat{K}\hat{\underline{u}}=\hat{\underline{f}}$ where $\kappa(\hat{K})<\kappa(K)$ will converge in fewer steps than the conjugate gradient method applied to $K\underline{u}=\underline{f}$. This observation is the motivation for the preconditioned conjugate gradient method Instead of solving $K\underline{u}=\underline{f}$, we choose to solve

$$\hat{K}\hat{\underline{u}}=\hat{\underline{f}} \tag{6.4}$$

where

$$\hat{K}=Q^{-1}KQ^{-T}$$
$$\hat{\underline{u}}=Q^{T}\underline{u}$$
$$\hat{\underline{f}}=Q^{-1}\underline{f}$$

and Q is a nonsingular matrix chosen so that $\kappa(\hat{K})<\kappa(K)$ Since Q is nonsingular, we can define

$$M=QQ^{T} \tag{6.5}$$

and M will be symmetric and positive definite. In terms of M, $\hat{K}$ can be written as

$$\hat{K}=Q^{T}M^{-1}KQ^{-T} \tag{6.6}$$

from which it can be seen that the eigenvalues of $\hat{K}$ and $M^{-1}K$ are the same. The introduction of M into the expression for $\hat{K}$ allows the standard conjugate gradient algorithm to be written for the solution of $\underline{u}$ directly in terms of M without explicitly forming Q This algorithm is described in Chandra[1978] and is given below

(1) Choose $\underline{u}^0$

(2) $\underline{r}^0 = \underline{f} - K\underline{u}^0$

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

(4) $\underline{p}^0 = \hat{\underline{r}}^0$

(5) k=0

(6) For $k=0,1,\ldots,k_{max}$

(1) $\alpha = \dfrac{(\hat{\underline{r}}^k, \underline{r}^k)}{(\underline{p}^k, K\underline{p}^k)}$

(2) $\underline{u}^{k+1} = \underline{u}^k + \alpha\underline{p}^k$

(3) *If* $\left\|\underline{u}^{k+1} - \underline{u}^k\right\|_\infty < \epsilon$ *then stop. otherwise continue.*

(4) $\underline{r}^{k+1} = \underline{r}^k - \alpha K\underline{p}^k$

(5) $M\hat{\underline{r}}^{k+1} = \underline{r}^{k+1}$

(6) $\beta = \dfrac{(\hat{\underline{r}}^{k+1}, \underline{r}^{k+1})}{(\hat{\underline{r}}^k, \underline{r}^k)}$

(7) $\underline{p}^{k+1} = \underline{r}^{k+1} + \beta\underline{p}^k$

Algorithm 2. Preconditioned Conjugate Gradient Algorithm

The only difference in the implementation of Algorithm 2 and Algorithm 1 is the solution of a system of the form $M\hat{\underline{r}} = \underline{r}$ during each iteration. The considerations in choosing an M and in implementing the

corresponding system on a parallel computer are discussed in the next section

6.2.2. Implementation of Preconditioners

The preconditioned conjugate gradient algorithm of the last section requires a symmetric and positive definite preconditioning matrix M to be specified or computed. The question arises as how to choose M so that the condition number of $\hat{K}=Q^T M^{-1} K Q^{-T}$,

$$\kappa(\hat{K})=\frac{\max_i \lambda_i}{\min_i \lambda_i}$$

where λ_i are the eigenvalues of $\hat{K}$, or equivalently $M^{-1}K$, is as small as possible.

The best choice for M in the sense of minimizing $\kappa(\hat{K})$ is $M=K$ but this gains nothing since $K\hat{\underline{r}}=\underline{r}$ is just as difficult to solve as $K\underline{u}=\underline{f}$. The approach that has been taken in the literature is to choose M to be a symmetric and positive definite approximation of K. If we write K as

$$K=M-R \tag{6.7}$$

then

$$M^{-1}K=I-M^{-1}R \tag{6.8}$$

where R can be regarded as a remainder term. Concus, Golub, and O'Leary[1976] give the following three criteria for M to be an effective preconditioner:

(1) $M\hat{\underline{r}}=\underline{r}$ is easily solved

(2) $M^{-1}R$ has small or nearly equal eigenvalues, or

(3) $M^{-1}R$ has small rank

A fourth criteria that is a major consideration on a parallel computer is

(4) M is easily formed.

One class of preconditioners, incomplete Cholesky Conjugate Gradient, (ICCG), (see Manteuffel[1979] for example) chooses M to be an incomplete Cholesky factorization of the matrix K. That is, $M=LL^T$ where

$$K = LL^T - R \tag{6.9}$$

and L is a lower triangular matrix and R is the remainder term. The matrix L in (6.9) and hence the matrix R will vary as different rules are used to create the incomplete factorization. For example, one rule may restrict L to have the same sparsity structure as the lower part of K, whereas, another rule may allow fill-in within the band in some special fashion. In either case the system $M\hat{\underline{r}}=\underline{r}$ will be solved by forward and backward substitutions on the triangular systems

$$\begin{aligned} L\underline{y} &= \underline{r} \\ L^T\hat{\underline{r}} &= \underline{y} \end{aligned} \tag{6.10}$$

respectively.

The formation of M and the solution of the systems in (6.10) can be easily implemented on a sequential computer; however, an efficient parallel implementation on an array or vector machine may be difficult to devise. In particular, the formation of M as an incomplete Cholesky factorization may be difficult to implement in parallel. In addition, if L does not have a special structure, the forward and backward substitutions

will be inherently sequential processes although Sameh and Kuck[1978] and van der Vorst[1981] have discussed the parallel solution of triangular systems and we address this issue in more depth later in this section. However, in general, tridiagonal and banded matrices are not well suited for preconditioning matrices for a conjugate gradient method to be implemented on parallel computers.

Another class of preconditioners that appears to be more easily implemented on parallel computers arises by choosing M to be a splitting of K that describes a linear stationary iterative method. As a first example, let D be the diagonal or (block diagonal) of K and choose $M=D$. We note that in most cases $M=D$ will not closely approximate K. Furthermore, the choice $M=D$ corresponds to a diagonal (block diagonal) scaling of K. That is,

$$\begin{aligned} \hat{K} &= D^{-1/2} K D^{-1/2} \\ \hat{\underline{u}} &= D^{-1/2} \underline{u} \\ \hat{\underline{f}} &= D^{-1/2} \underline{f} \end{aligned} \tag{6.11}$$

and in practice this scaling would be done a priori and $M\hat{\underline{r}}=\underline{r}$ would not be solved on each iteration. That is, the standard conjugate gradient method would be applied to (6.11) each iteration.

As a second example of a preconditioner that arises from a splitting for an iterative method, consider the SSOR splitting of $K\hat{\underline{r}}=\underline{r}$ which is

$$\frac{\omega}{2-\omega}\left[\left(\frac{1}{\omega}D-L\right)D^{-1}\left(\frac{1}{\omega}D-U\right)\right]\hat{\underline{r}} = \frac{1}{2-\omega}\left[\frac{(1-\omega)^2}{\omega}D+(1-\omega)(L+U)+\omega L D^{-1} U\right]\hat{\underline{r}}+\underline{r} \tag{6.12}$$

where D, $-L$, and $-U$ are the diagonal, strictly lower, and strictly upper parts of K respectively. If we choose $\hat{\underline{r}}^{(0)}=0$ and take one step of the

SSOR method applied to $K\hat{r}=r$, the resulting $\hat{r}^{(1)}$ will be the exact solution to the system $M\hat{r}=r$ where the matrix M is given by (6.13).

$$M = \frac{\omega}{2-\omega}(\frac{1}{\omega}D-L)D^{-1}(\frac{1}{\omega}D-U) \tag{6.13}$$

We now consider the parallel implementation of the solution of $M\hat{r}=r$ when M is given by (6.13). If the matrix K is ordered by the Multi-color ordering, then the solution to the triangular systems

$$\begin{aligned}(\frac{1}{\omega}D-L)y &= \frac{2-\omega}{\omega}Dr \\ (\frac{1}{\omega}D-L)^T\hat{r} &= y\end{aligned} \tag{6.14}$$

can be efficiently implemented on parallel computers as one Multi-color SSOR iteration applied to $K\hat{r}=r$ with initial guess $\hat{r}^{(0)}=0$.

Systems like (6.14) can be solved as Multi-color SSOR implementations even if $-L$ does not have the same elements as K as long as the sparsity structure of $(D-L)$ corresponds to some Multi-color ordering. We note that being able to solve these systems efficiently on an array computer would allow ICCG methods that require the factors of M to have the same sparsity structure as K or that correspond to some Multi-color scheme to be implemented on parallel computers provided an efficient algorithm could be found to do the incomplete factorization in parallel.

We next show for Laplace's equation that the above implementation of the SSOR preconditioning matrix (with $\omega=1, D=I$) for a Multi-colored grid achieves more accuracy with less computation than an implementation described by van der Vorst[1981] for a natural ordering of the grid.

Let $I-E-F$ denote the lower triangular part of the matrix that results from a 5-star discretization of Laplace's equation where the grid is ordered by the natural ordering. Let the matrices $-E$ and $-F$ contain the first and second nonzero subdiagonals of the matrix K respectively. Then in its block tridiagonal form, the matrix K can be written as

$$K = \begin{bmatrix} T_1 & -F^T_1 & & & \\ -F_1 & T_2 & -F^T_2 & \ddots & \\ & -F^T_2 & \ddots & \ddots & -F^T_{N-1} \\ & & \ddots & F_{N-1} & T_N \end{bmatrix} \tag{6.15}$$

where the matrix F is partitioned into the nxn diagonal submatrices F_i and the matrix E is partitioned into the nxn E_i submatrices where E_i is the lower triangular part of the symmetric tridiagonal matrix T_i which has been scaled to have unit diagonal. Recall, that the system of equations that must be solved each iteration to implement the preconditioner is,

$$(I-E-F)(I-E-F)^T\hat{r} = r \tag{6.16}$$

Now, van der Vorst suggests approximating the forward substitution

$$(I-E-F)y = r \tag{6.17}$$

or equivalently the partitioned systems

$$(I-E_i)y_i = r_i + F_i y_{i-1}$$

by

$$y_i = (I+E_i+E_i^2+\ldots+E_i^m)(r_i+F_i y_{i-1}) \tag{6.18}$$

where m terms of the series for $(I-E_i)^{-1}$ are taken and a similar

expression is found for the approximation to the back substitution $(I-E-F)^T\hat{\underline{r}}=\underline{v}$. Therefore, his idea is to take enough terms to approximate M given by (6.16) and at the same time produce a preconditioner for a natural ordering of the grid that is vectorizable without being cost prohibative. Simple operation counts show the following number of multiplications are needed to implement this scheme.

(1) n to calculate $F_j\underline{v}_{j-1}$

(2) $\frac{2n(m-1)-m(m+1)+2}{2}$ for *finding* $I+E_j+\ldots+E_j^m$ and $I+E_j^T+\ldots+(E_j^m)^T$.
For $m=2$, this is $n-2$. For $m=3$, this is $2n-5$.

(3) $\frac{2mn-m^2-m}{2}$ for *multiplying* $(I+E_j+\ldots+E_j^m)(\underline{r}_j+F_j\underline{v}_{j-1})$
For $m=2$, this is $2n-3$. For $m=3$, this is $3n-6$.

The total number of multiplications for $m=2$ and $m=3$ are given below in (6.19)

$$\begin{array}{ll} 7N - 10\sqrt{N} & m=2 \\ 10N - 19\sqrt{N} & m=3 \end{array} \tag{6.19}$$

Now, if the grid points are ordered by the R/B ordering, the matrix will have the form

$$K = \begin{bmatrix} I_1 & -F_1^T \\ -F_1 & I_2 \end{bmatrix}$$

Note that the matrix E in (6.16) is now $\underline{0}$, so that the van der Vorst scheme in (6.18) reduces to

$$\underline{y}_j = \underline{r}_j + F_j \underline{y}_{j-1} \qquad \text{for } j=1,2$$

$$\hat{\underline{r}}_j = \underline{y}_j + F_j^T \hat{\underline{r}} \qquad \text{for } j=1,2$$

which is the R/B SSOR iteration on the equation $K\hat{\underline{r}}=\underline{r}$ with $\hat{\underline{r}}^{(0)}$. The number of multiplications required for this Multi-color SSOR implementation is found from (5.29) to be at most $4N$. Hence, by ordering the grid in a R/B fashion, $O(3N)$ and $O(6N)$ multiplications can be saved over the van der Vorst 2 or 3-term implementation respectively for the natural ordering. In principle, the van der Vorst scheme is more general since it can be applied to block matrices K regardless of the ordering of the unknowns, but the more dense the matrices T_j, the more expensive the scheme will be. We also note that the m-term approximation to $(I-E_j)^{-1}$ in (6.18) is not necessary if the grid is ordered R/B (also true for Multi-colored grids) since $E_j=0$ for *all* j. This means that the solution to (6.16) is exact for the R/B ordering, whereas, it is only approximate for the natural ordering whenever $m<n+1$. In addition, even if an exact solution to (6.16) could be obtained with a small value of m, (say 2 or 3), the number of iterations of the PCG method with the resulting preconditoner would have to be O(1.75) or O(2.5) times less (for $m=2,3$ respectively) than the number of iterations with the R/B PCG method to compensate for the increase in the computational work.

6.2.3. m-Step PCG Methods

6.2.3.1. Description

It was demonstrated in section 6.2.2 that taking one step of a linear stationary iterative method such as Jacobi or SSOR applied to $K\hat{\underline{r}}=\underline{r}$ with $\hat{\underline{r}}^{(0)}=0$ results in a preconditioner for the conjugate gradient method that

can be implemented on a vector or array computer. The question now arises whether it would be beneficial to take more than one step of a linear stationary iterative method to produce a preconditioner M that more closely approximates K.

We begin by deriving an expression for M. Let $K=P-Q$ be a splitting of K that is associated with the linear stationary iterative method with iteration matrix $G=P^{-1}Q$. Then the m-step iterative method applied to $K\hat{\underline{r}}=\underline{r}$ is

$$P(I+G+\ldots+G^{m-1})^{-1}\hat{\underline{r}}^{(m)} = [P(I+G+\ldots+G^{m-1})^{-1}-(P-Q)]\hat{\underline{r}}^{(0)}+\underline{r} \tag{6.20}$$

By choosing $\hat{\underline{r}}^{(0)}=0$, (6.20) becomes

$$P(I+G+\ldots+G^{m-1})^{-1}\hat{\underline{r}}^{(m)} = \underline{r} \tag{6.21}$$

Hence, the preconditioning matrix is

$$M = P(I+G+\ldots+G^{m-1})^{-1} \tag{6.22}$$

Now, M must be symmetric and positive definite to be considered as a preconditioner for the conjugate gradient method. Before we establish the necessary and sufficient conditions for M to satisfy these criteria, we prove the following lemma

Lemma 1.

If $A=BC$ is a symmetric positive definite matrix, B is symmetric, and C has positive eigenvalues, then B is positive definite.

Proof:

Let $C^{-1}\underline{x}=\lambda\underline{x}$, or equivalently,

$$A^{-1}B\underline{x}=\lambda\underline{x} \tag{6.23}$$

Multiply both sides by $A^{1/2}$ to get

$$(A^{-1/2}BA^{-1/2})(A^{1/2}\underline{x}) = \lambda A^{1/2}\underline{x} \tag{6.24}$$

or

$$R\underline{y}=\lambda\underline{y}$$

The proof is now by contradiction. Assume that B has a nonpositive eigenvalue. Then, since (6.24) is a congruency transformation of B, it follows that R has a nonpositive eigenvalue (see Gantmacher[1959]). But the spectrum of R is identical to that of C^{-1} and by hypothesis can not have a nonpositive eigenvalue. Hence B is positive definite.

The necessary and sufficient conditions for M to be positive definite are given in Theorem 1.

Theorem 1.

Let $K=P-Q$ be a symmetric positive definite matrix and let P be a symmetric nonsingular matrix. Then

(1) the matrix M of (6.22) is symmetric.

(2) for m odd, M is positive definite if and only is P is positive definite.

(3) for m even, M is positive definite if and only if $P+Q$ is positive definite.

Proof.

To prove symmetry, we write M^{-1} as

$$M^{-1}=P^{-1}+P^{-1}QP^{-1}+P^{-1}QP^{-1}QP^{-1}+\underbrace{\ldots+P^{-1}QP^{-1}Q\cdots P^{-1}}_{m-1 \text{ terms}} \qquad (6.25)$$

Now since P and K and hence Q are symmetric, each term in (6.25) is symmetric. Thus M^{-1} and therefore M are symmetric.

The matrix $G=P^{-1}Q$ can be expressed as $G=K^{-\frac{1}{2}}(I-K^{\frac{1}{2}}P^{-1}K^{\frac{1}{2}})K^{\frac{1}{2}}$. Since P^{-1} is symmetric with P, the eigenvalues of the congruence transformation $K^{1/2}P^{-1}K^{1/2}$ are real. Hence, the eigenvalues of G are real.

To prove (2), let m be odd. If g is any eigenvalue of G other than 1, the corresponding eigenvalue of

$$R=(I+G+\ldots+G^{m-1})$$

is

$$1+g+\ldots+g^{m-1}=\frac{1-g^m}{1-g}$$

which is positive since m is odd. If $g=1$, the corresponding eigenvalue of R is equal to m which is also positive.

Now, since $P=MR$ and M is symmetric and R has positive eigenvalues, it follows from Lemma 1 that if P is positive definite then M must also be positive definite. Conversely, M can be written as $M=PR^{-1}$. Since R^{-1} has positive eigenvalues and P is symmetric, we conclude from Lemma 1 that if M is posi-

tive definite then P is also positive definite.

Next, to prove (3) let m be even. It is sufficient to consider M^{-1} since any conclusions about the definiteness of M^{-1} will apply to M. Since m is even, M^{-1} from (6.22) can be written as

$$M^{-1} = P^{-1}(P+PG+PG^2+PG^3+\ldots+PG^{m-1})P^{-1}$$

or

$$M^{-1}=P^{-1}[(P+PG)+(P+PG)G^2+(P+PG)G^4+\ldots+(P+PG)G^{m-2})]P^{-1}$$

Now, since $PG=Q$, M^{-1} can be written as

$$M^{-1}=P^{-1}(P+Q)(I+G^2+G^4+\ldots+G^{m-2})P^{-1} \tag{6.26}$$

Now, since P is nonsingular and symmetric, M^{-1} is positive definite if and only if the symmetric matrix

$$S=(P+Q)(I+G^2+G^4+\ldots+G^{m-2}) \tag{6.27}$$

is positive definite

Assume $P+Q$ is positive definite. Since S is symmetric and the matrix $(I+G^2+G^4+\ldots+G^{m-2})^{-1}$ has positive eigenvalues, S is positive definite by Lemma 1.

Conversely, if S is positive definite, since $P+Q$ is symmetric and the series $I+G^2+G^4+\ldots+G^{m-2}$ has positive eigenvalues, $P+Q$ is positive definite by Lemma 1.

Dubois, Greenbaum, and Rodrique[1979] consider a truncated Neumann series for K^{-1} as a preconditioner This preconditioner is equivalent to that of (6.22) if $K=P-Q$ corresponds to a Jacobi splitting where $P=diag(K)$, but they do not consider more complicated splittings that result from iterative methods. Theorem 1 extends their main result. Under the hypothesis that K and P are both symmetric and positive definite matrices and $\rho(G)<1$, they prove that M is symmetric and positive definite for all m Note that for odd m the condition that $\rho(G)<1$ is not needed and for even m, the matrix P is only required to be symmetric. The relationship between the condition $\rho(G)<1$ and the positive definiteness of $P+Q$ is given later in this discussion in Theorem 2.

Theorem 1 is helpful in choosing a splitting of K that will produce an m-step preconditioner that is symmetric and positive definite For example, if the Jacobi splitting of K ($P=D$ and $Q=D-K$ where D is the diagonal of K) were considered, part (3) of the theorem says that if m is even, $P+Q$ must be positive definite. We know from the Jacobi Convergence Theorem (see, e g. Young[1971]),

Jacobi Convergence Theorem

> Let $K=P-Q$ be a real, symmetric, and nonsingular matrix with positive diagonal elements Then the Jacobi method converges ($\rho(G)<1$) if and only if both $P-Q$ and $P+Q$ are positive definite.

that $P+Q$ and hence M will be positive definite only if the Jacobi method is convergent. For the problems of interest to us, the Jacobi method is not guaranteed to be convergent since we only know that K will be symmetric and positive definite. Therefore, for these problems, only odd values of m will yield m-step Jacobi preconditioning matrices

that are guaranteed to be positive definite.

For any splitting satisfying the hypothesis of Theorem 1, the question arises whether the same relationship exists between the positive definiteness of $P+Q$ and the convergence of the iterative method with iteration matrix $P^{-1}Q$ that is given in the Jacobi Convergence Theorem above for the Jacobi splitting. We answer this question with Theorem 2.

Theorem 2.

Let $K=P-Q$ be a symmetric positive definite matrix and let P be symmetric and nonsingular. Then $\rho(P^{-1}Q)<1$ if and only if $P+Q$ is positive definite.

Proof:

First, assume P+Q is positive definite. Since K is symmetric positive definite and P is nonsingular, K=P-Q is a p-regular splitting. Hence, from Ortega's p-regular splitting theorem, Ortega[1971], $\rho(P^{-1}Q)<1$.

We next note that $G=P^{-1}Q$ can be expressed as

$$G=K^{-1/2}(I-K^{1/2}P^{-1}K^{1/2})K^{1/2}$$

and the matrix $K^{1/2}P^{-1}K^{\frac{1}{2}}$ has real eigenvalues since K is symmetric positive definite and P is symmetric. Hence, G has real eigenvalues.

Now, assume that $\rho(G)<1$, then $(I-G)^{-1}$ exists and since G has real eigenvalues, it easily follows that the matrix H defined by

$$H=(I-G)^{-1}(I+G) \tag{6.28}$$

has real eigenvalues. But we know from a theorem by Young[1971] that H is N-stable. Hence H has positive eigenvalues. Now, we can write H as

$$H=K^{-1}(P+Q) \tag{6.29}$$

or equivalently,

$$K=(P+Q)H^{-1} \tag{6.30}$$

Finally, since K is symmetric and positive definite and H^{-1} has positive eigenvalues and $P+Q$ is symmetric, we conclude from Lemma 1 that $P+Q$ is positive definite.

Note that the requirement that P be symmetric is stated as a sufficient condition but not a necessary one. It remains to be proven whether or not the symmetry of P is necessary for M to be symmetric for odd $m>1$ or for $P+Q$ and hence M to be symmetric and positive definite for even m. However, in practice, it was observed that the number of iterations for convergence of the m-step PCG method with a nonsymmetric matrix P was extremely more than the number required by the standard conjugate gradient method ($m=0$). In particular, we solved the 60x60 plane stress problem which has a symmetric and positive definite coefficient matrix K with an m-step R/B/G SOR preconditioner. From the SOR convergence theorem stated in Chapter 5, we know that $\rho(G)<1$, but the SOR splitting matrix with $\omega=1$ for simplicity is $P=D-L$ and is not symmetric. The results are given in Table 1

m	m-step R/B/G SOR
0	49
1	200 +
2	200 +
3	200 +
4	200 +

Table 1. Number of m-step R/B/G SOR PCG Iterations
60x60 Plane Stress Problem

These results indicate that only symmetric splittings should be considered. In the next section, we include results of the m-step PCG method derived from the Jacobi and SSOR splittings which are both symmetric

6.2.3.2. Analysis of the Condition Number

In the last section, we gave conditions for M to be symmetric and positive definite and hence to be considered as a preconditioner for the conjugate gradient method In this section we determine if increasing m will in fact produce a better conditioned system. For this purpose, we now denote by M_m the matrix of (6.22).

As a first step towards answering this question, we derive an expression for $\kappa(\hat{K}_m)$ Recall from (6.6) that $\hat{K}$ is similar to $M_m^{-1}K$ so that $\kappa(\hat{K}_m)$ is the same as the ratio of the largest to smallest eigenvalues of $M_m^{-1}K$. An expression for $M_m^{-1}K$ as a polynominal in G is

$$M_m^{-1}K = (I+G+\ldots+G^{m-1})P^{-1}(P-Q) \tag{6.31}$$

or

$$M_m^{-1}K = I-G^m$$

where $G=P^{-1}Q$.

Since we wish to compare $\kappa(\hat{K}_m)$ to $\kappa(\hat{K}_{m+1})$, we will assume that both M_m and M_{m+1} are symmetric and positive definite. By Theorem 1, this implies that P and $P+Q$ are positive definite and thus by Theorem 2, $\rho(G)<1$. Therefore, since the proof of Theorem 1 showed that the eigenvalues λ_i of G are real, they can be ordered as

$$-1<\lambda_1<\lambda_2<\cdots<\lambda_n<1$$

Furthermore, let δ be the eigenvalue with the smallest absolute value Then the condition number of $\hat{K}_m$ is

$$\kappa(\hat{K}_m) = \begin{cases} \dfrac{1-\lambda_1^m}{1-\lambda_n^m} & \lambda_1 \geq 0 \text{ or } \lambda_1<0 \text{ and } m \text{ odd} \\ \dfrac{1-\delta^m}{1-\lambda_n^m} & \lambda_1<0,\ \lambda_n \geq |\lambda_1| \ .m \text{ even} \\ \dfrac{1-\delta^m}{1-\lambda_1^m} & \lambda_1<0,\ |\lambda_1| \geq |\lambda_n| \ .m \text{ even} \end{cases} \tag{6.32}$$

As can be seen from (6.32), the conditions for $\kappa(\hat{K}_{m+1})<\kappa(\hat{K}_m)$ depend upon the distribution of the eigenvalues λ_i of G. These conditions are given by Theorem 3 if $\lambda_1 \geq 0$, and by Theorem 4 if $\lambda_1<0$ and $\lambda_n \geq |\lambda_1|$. We note that (6.32) shows for both odd and even m that if $\lambda_1<0$ and $|\lambda_1| \geq |\lambda_n|$ it is impossible to conclude if $\kappa(\hat{K}_{m+1})<\kappa(\hat{K}_m)$ without knowledge of the values of λ_1, λ_n, and δ.

Theorem 3.

Let $K=P-Q$ and P be symmetric and positive definite with $\rho(G)<1$. Then if $\lambda_1 \geq 0$, $\kappa(\hat{K}_m)$ is a decreasing function for all m.

Proof:

We must show that $\kappa(\hat{K}_{m+1}) < \kappa(\hat{K}_m)$ By (6.32),

$$\kappa(\hat{K}_m) = \frac{(1-\lambda_1)(1+\lambda_1+\lambda_1^2+\ldots+\lambda_1^{m-1})}{(1-\lambda_n)(1+\lambda_n+\lambda_n^2+\ldots+\lambda_n^{m-1})} \tag{6.33}$$

Hence, we must show for $\lambda_1 \geq 0$ that

$$\frac{1+\lambda_1+\lambda_1^2+\ldots+\lambda_1^m}{1+\lambda_1+\lambda_1^2+\ldots+\lambda_1^{m-1}} < \frac{1+\lambda_n+\lambda_n^2+\ldots+\lambda_n^m}{1+\lambda_n+\lambda_n^2+\ldots+\lambda_n^{m-1}} \tag{6.34}$$

This inequality is true since $\lambda_n > \lambda_1$ and

$$f(x) = \frac{1+x+x^2+\ldots+x^m}{1+x+x^2+\ldots+x^{m-1}} \tag{6.35}$$

can easily be shown to be an increasing function of x for $x \geq 0$.

As an application of Theorem 3 consider the SSOR splitting of a symmetric and positive definite matrix. Recall from the basis convergence theorem for SSOR that was stated in Chapter 5, that if K is a symmetric matrix with positive diagonal elements, the SSOR method converges if and only if K is positive definite and $0<\omega<2$. Therefore, $\rho(G)<1$ for the SSOR splitting and from Young [1971] we know that all the eigenvalues of G are real and nonnegative. Hence $\lambda_1 \geq 0$. To satisfy the last hypothesis of Theorem 3 we prove that the matrix P for the SSOR splitting is symmetric and positive definite.

Lemma 2.

Let $K=P-Q$ be symmetric and positive definite. If P is the SSOR splitting matrix, then P and M_m are symmetric and positive definite.

Proof.

Now,

$$P=\frac{\omega}{2-\omega}[D^{-1/2}(\frac{1}{\omega}D-U)]^T\ [D^{-1/2}(\frac{1}{\omega}D-U)]$$

where $D=diag(K)$, and $-U$ is the strictly upper triangular part of K. Since the matrix $D^{-1/2}(\frac{1}{\omega}D-U)$ is upper triangular with positive diagonal elements and hence nonsingular, it follows immediately that P is symmetric and positive definite. Therefore by Theorems 1 and 2, M_m is also symmetric and positive definite for all m

The results of the m-step SSOR preconditioned conjugate gradient method on the 60x60 plane stress problem problem are given in Table 2. the results on the 1536x1536 plane stress problem are given in Table 3, and the results on a 768x768 matrix derived from the 5-star discretization of Laplace's equation are given in Table 4 For all three problems, the results are given for both the natural rowwise ordering and Multi-color ordering of the grid. The convergence criterion was $\|\underline{u}^{k+1}-\underline{u}^{k}\|_\infty<\epsilon$, where $\epsilon=10^{-6}$ for all three problems. The standard conjugate gradient results with no preconditioning are indicated by $m=0$.

m	R/B/G # Iterations (ω=1)	Natural # Iterations (ω=1)	Natural # Iterations (ω=1.2)
0	49	49	49
1	23	20	20
2	16	15	14
3	14	12	12
4	12	11	10

Table 2. m-step SSOR PCG for 60x60 Plane Stress Problem

m	R/B/G # Iterations (ω=1)	Natural # Iterations (ω=1)	Natural # Iterations (ω=1.6)
0	363	363	363
1	139	111	93
2	99	80	66
3	82	65	54
4	71	57	47

Table 3. m-step SSOR PCG for 1536x1536 Plane Stress Problem

m	R/B # Iterations (ω=1)	Natural # Iterations (ω=1)	Natural # Iterations (ω=1.8)
0	56	56	56
1	30	28	17
2	22	21	13
3	18	17	10
4	16	15	9

Table 4. m-step SSOR PCG for 768x768 Laplace's Equation

The results in Tables 2,3, and 4 show that the number of iterations is a decreasing function of m as was predicted by Theorem 3. The results also indicate that there will be an optimal value of m, say m_{opt}, since for $m > m_{opt}$, the reduction in the number of CG iterations is not enough to balance the increase in the number of iterations of the SSOR preconditioner For example, consider the R/B/G results in Table 3. The number of CG iterations and the number of steps of the SSOR

preconditioner as a function of m are summarized in the table below. The last two columns of the table give the total algorithm cost (in units of SSOR iterations) for the assumptions that one CG iteration is equivalent to one SSOR iteration and that one CG iteration is twice as expensive as one SSOR iteration respectively.

	Iterations		Total Cost	
m	CG	SSOR	CG=SSOR	CG=2(SSOR)
0	363	0	363	726
1	139	139	278	417
2	99	198	297	396
3	82	246	328	410
4	71	284	355	426

For this example, $m=1$ is optimal if one CG iteration costs the same as one SSOR iteration and $m=2$ is optimal if one CG iteration is twice as costly as one SSOR iteration. The actual relative cost of the CG and SSOR iterations on a parallel computer will be a factor of the amount of artithmetic and communication operations in each algorithm as well as the times to perform these operations on the machine. These issues will be discussed in more detail in Chapter 7

We now prove Theorem 4.

Theorem 4

Let $K=P-Q$ and P be symmetric and positive definite with $\rho(G)<1$. Then if $\lambda_n > |\lambda_1|$ and $\lambda_1<0$.

(1) for m odd, $\kappa(\hat{K}_{m+1}) < \kappa(\hat{K}_m)$.

(2) for m even, $\kappa(\hat{K}_{m+1}) < \kappa(\hat{K}_m)$ if and only if $(1+|\lambda_1|^{m+1})(1-\lambda_n^m) < (1-\delta^m)(1-\lambda_n^{m+1})$

Proof:

By (6.32), we must show that

$$\frac{1-\delta^{m+1}}{1-\lambda_n^{m+1}} < \frac{1+|\lambda_1|^m}{1-\lambda_n^m} \tag{6.36}$$

Since $\lambda_n < 1$ and $m+1$ is even, (6.36) is true because $1-\delta^{m+1} < 1+|\lambda_1|^m$ and $1-\lambda_n^{m+1} > 1-\lambda_n^m$

Statement (2) of the theorem follows from (6.32) directly since $\kappa(\hat{K}_{m+1}) < \kappa(\hat{K}_m)$ can be written as

$$\frac{1+|\lambda_1|^{m+1}}{1-\lambda_n^{m+1}} < \frac{1-\delta^m}{1-\lambda_n^m} \tag{6.37}$$

and $1-\lambda_n^{m+1} > 1-\lambda_n^m$ and $1+|\lambda_1|^{m+1} > 1-\delta^m$.

Observe from Theorem 4 that if $\lambda_n > |\lambda_1|$ a better conditioned system will result by increasing m from m (odd) to $m+1$ (even), whereas, this may not be the case if m is increased from m (even) to $m+1$ (odd).

As an example of the application of Theorem 4, we consider the Jacobi splitting of any symmetric and positive definite matrix K that has Property A (see Young [1971]). For this splitting, $P=D$ where D is the diagonal of K and therefore P is symmetric and positive definite. Now, since K has Property A, the eigenvalues λ_i of G occur in $\pm\lambda_i$ pairs and $\lambda_n=-\lambda_1$ and $\delta=0$ From (2) of the theorem, we conclude that going from an even to a consecutive odd number of steps is advantageous if

and only if

$$(1+\lambda_n^{m+1})(1-\lambda_n^m) < (1-\lambda_n^{m+1}) \tag{6.38}$$

or equivalently,

$$\lambda_n^{m+1} - 2\lambda_n + 1 > 0 \tag{6.39}$$

As m increases the inequality in (6.39) reduces asymptotically to

$$\lambda_n < \frac{1}{2} \tag{6.40}$$

For $m=2$ and $m=3$, the exact conditions are $\lambda_n<.62$ and $\lambda_n<.53$ respectively. But for problems of interest to us, λ_n will be closer to 1 and we can conclude that it is not advantageous to increase m from m (even) to $m+1$ (odd). This fact has been verified by numerical experiments for the m-step Jacobi preconditioner on an 89x89 symmetric and positive definite system that had Property A. The results are given in Table 5.

m	# iterations
0	45
1	45
2	23
3	36
4	21
5	30
6	18
7	26
8	16

Table 5. m-step Jacobi Results for 89x89 Problem

Note from Table 5 that increasing m from 2 to 3, from 4 to 5, and from 6 to 7 also increases the number of iterations from 23 to 36, from 21 to 30, and from 18 to 26 respectively. On the other hand, observe

that increasing m from an odd to a consecutive even number always reduces the number of iterations. Dubois, Greenbaum, Rodrique[1979] reported similar results for Poisson's equation but they explained the results by assuming that the eigenvalues of $M_m^{-1}K$ were near 0,1, and 2. Hence, our explanation given by Theorem 4 is more general.

Theorem 4 suggested that in certain instances it is better to take an even number of steps of the preconditioner. If this were done, the question would be to determine the conditions for $\kappa(\hat{K}_{m+2}) < \kappa(\hat{K}_m)$. These conditions are given in Theorem 5. Notice that the hypothesis of this theorem only requires P to be symmetric and nonsingular instead of positive definite.

Theorem 5.

Let $K=P-Q$ be symmetric and positive definite and let P be symmetric and nonsingular. If $\rho(G)<1$ and m is even then $\kappa(\hat{K}_{m+2}) < \kappa(\hat{K}_m)$.

Proof:

From (6.32) we must show that

$$\begin{aligned} \frac{1-\delta^{m+2}}{1-\lambda_1^{m+2}} &< \frac{1-\delta^m}{1-\lambda_1^m} \quad \text{for} \quad |\lambda_1| > |\lambda_n| \\ \frac{1-\delta^{m+2}}{1-\lambda_n^{m+2}} &< \frac{1-\delta^m}{1-\lambda_n^m} \quad \text{for} \quad \lambda_n > |\lambda_1| \end{aligned} \tag{6.41}$$

We rewrite (6.41) as

$$\frac{1-|\delta|^{m+2}}{1-|\delta|^{m}} < \frac{1-|\lambda_1|^{m+2}}{1-|\lambda_1|^{m}} \quad \text{for} \quad |\lambda_1| > |\lambda_n|$$

$$\frac{1-|\delta|^{m+2}}{1-|\delta|^{m}} < \frac{1-|\lambda_n|^{m+2}}{1-|\lambda_n|^{m}} \quad \text{for} \quad \lambda_n > |\lambda_1| \tag{6.42}$$

Since $\delta \leq |\lambda_1|$ and $\delta \leq |\lambda_n|$, (6.42) is true since

$$f(x) = \frac{1-x^{m+2}}{1-x^{m}} \tag{6.43}$$

is an increasing function of x for $x \geq 0$.

Hence, if we always take an even number of steps of the preconditioner, a better conditioned system will result as the number of steps increases.

So far we have only addressed the question of whether a better conditioned system results by increasing m. We now turn to the question of how much improvement over $m=1$ can be made by taking $m>1$ steps of the preconditioner. Dubois, Greenbaum, and Rodrique[1979] prove that the m-step PCG method can only reduce the number of iterations needed by the 1-step PCG method by a factor of m, that is,

$$\frac{\#\ \textit{iterations}\ \ 1\textit{-step}\ \ PCG}{\#\ \textit{iterations}\ \ m\textit{-step}\ \ PCG} \leq m \tag{6.44}$$

In practice, this theoretical bound may not be reached and for a given distribution of eigenvalues it may be sharper for some values of m than for others. The results of Dubois, et.al.[1979] show this for the m-step Jacobi PCG for Laplace's equation. Tables 2, 3, and 4 show for the m-step SSOR PCG method applied to both the plane stress problem and Laplace's equation that the bound is best for $m=2$. Table 5 shows that for the m-step Jacobi PCG applied to a problem with Property A that

the bound is extremely sharp for $m=2$ and extremely poor for odd values of m.

In order to determine the conditions under which the m-step PCG method gives the most improvement over the 1-step PCG method, we examine the ratio $\frac{\kappa(\hat{K}_1)}{\kappa(\hat{K}_m)}$ for both odd and even m with different assumptions about the distribution of the eigenvalues λ_i of G which are assumed to be ordered as $-1<\lambda_1<\lambda_2<\ldots<\lambda_n<1$ with $\delta=\min_i |\lambda_i|$. This ratio can easily be calculated from the equations of (6.32) and is summarized below for the various cases.

$$\frac{\kappa(\hat{K}_1)}{\kappa(\hat{K}_m)} = \begin{cases} \dfrac{1+\lambda_n+\lambda_n^2+\ldots+\lambda_n^{m-1}}{1+\lambda_1+\lambda_1^2+\ldots+\lambda_1^{m-1}} & \lambda_1 \geq 0 \\ \dfrac{(1+|\lambda_1|)(1+\lambda_n+\lambda_n^2+\ldots+\lambda_n^{m-1})}{1+|\lambda_1|^m} & \lambda_1<0, \lambda_n>0, \ m \text{ odd} \\ \dfrac{(1+|\lambda_n|^m)(1+|\lambda_1|)}{(1+|\lambda_1|^m)(1+|\lambda_n|)} & \lambda_1<0, \lambda_n<0, \ m \text{ odd} \\ \dfrac{(1+|\lambda_1|)(1+\lambda_n+\lambda_n^2+\ldots+\lambda_n^{m-1})}{(1-|\delta|^m)} & \lambda_1<0, \lambda_n \geq |\lambda_1| \ m \text{ even} \\ \dfrac{(1+|\lambda_1|)(1-|\lambda_1|^m)}{(1-\lambda_n)(1-|\delta|^m)} & \lambda_1<0, |\lambda_1| \geq |\lambda_n| \ m \text{ even} \end{cases} \tag{6.45}$$

Several observations can be made from (6.45) and are listed below.

(1) If $\lambda_1 \geq 0$, the maximum value of $\frac{\kappa(\hat{K}_1)}{\kappa(\hat{K}_m)}$ occurs as

$\lambda_1 \longrightarrow 0$ and $\lambda_n \longrightarrow 1$ and is equal to m.

(2) If $\lambda_1 < 0$ and $\lambda_n > 0$, and m is odd, the maximum value of $\frac{\kappa(\hat{K}_1)}{\kappa(\hat{K}_m)}$ occurs when $\lambda_n \longrightarrow 1$ and is equal to $m\left(\frac{1+|\lambda_1|}{1+|\lambda_1|^m}\right)$.

(3) The m-step PCG method ($m>1$) is more effective if $\lambda_n > 0$.

(4) If $\lambda_1 < 0$, and $\lambda_n \geq \lambda_1$, and m is even, the maximum value of $\frac{\kappa(\hat{K}_1)}{\kappa(\hat{K}_m)}$ occurs when $\lambda_n \longrightarrow 1$ and $|\lambda_1| = |\lambda_n|$ and is equal to $\frac{2m}{1-\delta^m}$. Note that the larger δ, the larger this ratio will be. Hence to achieve the maximum performance in this case, we would like the value of δ to be as close to that of λ_1 as possible. For K matrices with Property A, this is not possible since $\delta=0$ and the maximum ratio of the two condition numbers is $2m$.

In summary, the m-step PCG method gives more improvement over the 1-step PCG method when an even number of steps of the preconditioner are taken and the eigenvalues of the matrix G are distributed as described in (4) above.

6.2.4. m-step Extrapolated PCG Methods

6.2.4.1. Description

It was pointed out in the last section that the m-step methods perform better if the smallest eigenvalue λ_1 of G is negative and the largest eigenvalue λ_n is positive with $\lambda_n = |\lambda_1|$. Furthermore, the maximum value of $\frac{\kappa(\hat{K}_1)}{\kappa(\hat{K}_m)}$ was seen to be $2m$ if $\delta=0$ and greater than $2m$ otherwise. The purpose of this section is to demonstrate how to achieve this distribution of eigenvalues by using extrapolation.

We begin by recalling that the iteration matrix H for an extrapolated iterative method can be written as

$$H = (1-\gamma)I + \gamma G \tag{6.46}$$

where G is the associated iteration matrix for $\gamma=1$

This iterative method, corresponds to the splitting of K given by

$$K = \frac{1}{\gamma}P - (\frac{1-\gamma}{\gamma}P + Q) \tag{6.47}$$

where

$$K = P - Q \tag{6.48}$$

is the splitting of K that leads to the unextrapolated ($\gamma=1$) method with iteration matrix $G=P^{-1}Q$.

If we define

$$R = \frac{1}{\gamma}P \tag{6.49}$$

and

$$S = \frac{1-\gamma}{\gamma}P + Q \tag{6.50}$$

the preconditioning matrix $M_{m,\gamma}$ for the extrapolated m-step PCG method is

$$M_{m,\gamma} = R(I+H+\ldots+H^{m-1})^{-1} \tag{6.51}$$

The following Corollary gives the necessary and sufficient conditions for $M_{m,\gamma}$ to be symmetric and positive definite.

Corollary 1.

Let $K=P-Q$ be symmetric and positive definite and let P be symmetric and nonsingular. If $\gamma>0$, then

(1) $M_{m,\gamma}$ is symmetric.

(2) for odd m, $M_{m,\gamma}$ is positive definite if and only if P is positive definite.

(3) for even m, $M_{m,\gamma}$ is positive definite if and only if $g_n<1$ and $\gamma<\frac{2}{1-g_1}$ where the eigenvalues of G are $g_1<g_2<\ldots<g_n$.

Proof:

$R=\frac{1}{\gamma}P$ is symmetric since P is symmetric. It follows from Theorem 1. that $M_{m,\gamma}$ is symmetric.

Since $\gamma>0$, (2) follows from Theorem 1 since for odd m, $\frac{1}{\gamma}P$ is positive definite if and only if P is positive definite.

To prove (3) we note from Theorem 1 that for even m, $M_{m,\gamma}$ is positive definite if and only if $R+S$ is positive definite Since R is symmetric and nonsingular, we know by Theorem 3 that $R+S$ is positive definite if and only if $\rho(H)<1$. Therefore,

$M_{m,\gamma}$ is positive definite if and only if $\rho((1-\gamma)I+\gamma G)<1$ and this condition is met if and only if $1-\gamma+\gamma g_n<1$ and $-1<1-\gamma+\gamma g_1$ or equivalently $g_n<1$ and $\gamma<\frac{2}{1-g_1}$ and (3) follows.

The ratio of the largest to smallest eigenvalues of $M_{m,\gamma}^{-1}K$ where

$$M_{m,\gamma}^{-1}K = I-H^m \tag{6.52}$$

depends upon the distribution of the eigenvalues of H as was discussed in the last section and this distribution will be a function of γ. However, for the special case $m=1$,

$$M_{1,\gamma}^{-1}K = \gamma(I-G)$$

the ratio of the largest to smallest eigenvalues of $M_{1,\gamma}^{-1}K$ is independent of γ and extrapolation is not worthwhile in this case.

6.2.4.2. Choosing the Extrapolation Factor

We would like to choose γ so that the eigenvalues of H, $h_1<h_2<\ldots<h_n$ satisfy

$$|h_1| = h_n \tag{6.53}$$

in order to achieve the most improvement over the 1-step extrapolated (or unextrapolated for $m=1$) method. Since $h_i=1-\gamma+\gamma g_i$, (6.53) leads to the following choice for γ:

$$\gamma_{opt} = \frac{2}{2-g_1-g_n} \tag{6.54}$$

Note that if $g_1=-g_n$, γ will equal 1 and the matrix G already has the optimal distribution of eigenvalues and hence no extrapolation will be performed. This will be the case for the Jacobi splitting for Laplace's equation.

Extrapolation is therefore most useful if all the eigenvalues g_l of G are nonnegative and from (6.45) the maximum value of $\frac{\kappa(\hat{K}_1)}{\kappa(\hat{K}_m)}$ is

$$\frac{\kappa(\hat{K}_1)}{\kappa(\hat{K}_m)} < \begin{cases} \frac{2m}{1-g_1^m} & g_1 \geq 0,\ m \text{ even, extrapolation} \\ m & g_1 \geq 0,\ \text{no extrapolation} \end{cases} \tag{6.55}$$

To illustrate how effective extrapolation can be for $m>1$ we consider the m-step SSOR PCG method. The eigenvalues of the SSOR iteration matrix G are nonnegative and $\rho(G)<1$ for symmetric and positive definite matrices K so the hypotheses for m even in Corollary 1 are met if we take $\gamma<\frac{2}{1-g_1}$. The plane stress problem and Laplace's equation were solved with the m-step extrapolated SSOR PCG method for the Multi-color orderings of the respective grids. The results are given in parenthesis in Tables 6 and 7 respectively.

m	R/B/G	Natural	
	ω=1	ω=1	ω=1.6
0	363	363	363
1	139	111	93
2	99 (72)	80	66
3	82	65	54
4	71 (59)	57	47

Table 6. m-step SSOR (Extrapolated SSOR) 1536x1536 Plane Stress Problem (γ=1.95)

m	R/B/G ω=1	Natural ω=1	Natural ω=1.8
0	56	56	56
1	30	28	17
2	22 (17)	21	13
3	18	17	10
4	16 (14)	15	9

Table 7. m-step SSOR (Extrapolated SSOR)
768x768 Laplace's Equation
(γ=1.7)

Note that for both the plane stress problem and Laplace's equation, the extrapolated method for the Multi-colored grid required fewer iterations for convergence than the corresponding unextrapolated method but still required more iterations than the unextrapolated method applied to the natural ordering of the grid with optimal relaxation factor. The ratio of the number of iterations for the 1-step method to the number of iterations for the m-step method is given in Table 8 for the plane stress problem and in Table 9 for Laplace's equation

m	Unextrapolated	Extrapolated	Theoretical Maximum
2	1.40	1.93	2.00
4	1.96	2.36	4.00

Table 8. Ratio of 1-step to m-step R/B/G SSOR PCG
Plane Stress Problem
(γ=1.95)

2	1.36	1.76	2.00
4	1.88	2.14	4.00

Table 9. Ratio of 1-step to m-step R/B SSOR PCG
Laplace's Equation
(γ=1.70)

Tables 8 and 9 show that for both problems, the extrapolated method with $m=2$ gives results closer to the theoretical maximum than does $m=4$.

The implementation of the extrapolation method takes little extra computational effort each iteration but does require the storage of an auxiliary vector of length equal to the number of unknowns. However, the major consideration in the use of extrapolation is the determination of the extrapolation factor γ. As is seen from (6.54), the optimal value of γ depends on prior knowlege of the largest and smallest eigenvalues of G which may not be known in practice.

6.2.4.3. Comparison to the PPCG Method

Johnson, Micchelli, and Paul [1982] have suggested symmetrically scaling the matrix K to have unit diagonal and then taking m terms of a parametrized Neumann series for $K^{-1} = (I-G)^{-1}$ as the value for M_m^{-1}. They call the resulting method the PPCG(m-1) method to mean m-step Parametrized Preconditioned Conjugate Gradient Method. This corresponds to a preconditioning matrix that is a polynominal of degree $m-1$ in G.

$$M_m^{-1} = \alpha_0 I + \alpha_1 G \quad \alpha_2 G^2 + .. + \alpha_{m-1} G^{m-1} \tag{6.56}$$

derived from the Jacobi splitting

$$K = I - G \tag{6.57}$$

and the solution to $M_m \hat{\underline{r}} = \underline{r}$ can be implemented by taking m steps of the Jacobi iterative method applied to $K\hat{\underline{r}} = \underline{r}$ with initial guess $\hat{r}^{(o)} = \underline{0}$.

Since K, I, and hence G are symmetric, clearly M_m is symmetric. Now, $M_m^{-1}K$ can be written as a polynominal in K,

$$M_m^{-1}K=[\alpha_o I+\alpha_1(I-K)+\alpha_2(I-K)^2+...+\alpha_{m-1}(I-K)^{m-1}]K \tag{6.58}$$

and Johnson, et.al., guarantee that M_m will be positive definite by choosing the α_i's so that the eigenvalues, $y(\lambda)$, of $M_m^{-1}K$ and hence those of M_m are positive on the interval $[\lambda_1,\lambda_n]$ that contains the eigenvalues of K. Hence, the idea of the parametrization is to choose the α_i's so that the eigenvalues of $M_m^{-1}K$ are positive on $[\lambda_1,\lambda_n]$ and are as close to 1 as possible in some sense such as the min-max or the least squares criteria.

We now show how to generalize this idea for any splitting of the matrix K. If we let

$$K = P - Q \tag{6.59}$$

and $G=P^{-1}Q$ then from (6.22), the inverse of the m-step preconditioner is

$$M_m^{-1} = (I+G+G^2+...+G^{m-1})P^{-1} \tag{6.60}$$

We parametrize this series as,

$$M_{m,\alpha}^{-1} = (\alpha_o+\alpha_1G+\alpha_2G^2+...+\alpha_{m-1}G^{m-1})P^{-1} \tag{6.61}$$

and note from Theorem 1 that $M_{m,\alpha}$ will be symmetric if P is symmetric since the α's do not affect the proof of symmetry.

The expression for $M_{m,\alpha}^{-1}K$ is given by

$$M_{m,\alpha}^{-1}K=[\alpha_0+\alpha_1(I-P^{-1}K)+\ldots+\alpha_{m-1}(I-P^{-1}K)^{m-1}]P^{-1}K \qquad (6.62)$$

and is seen to be a polynominal in $P^{-1}K$ rather than in K as in (6.58).

This means that the values of α_i should be chosen so that the eigenvalues $y(\lambda)$ of $M_{m,\alpha}^{-1}K$ are positive on the interval $[\lambda_1,\lambda_n]$ that contains the eigenvalues of $P^{-1}K$ and are as close to 1 as possible in some sense such as the min-max or least squares criteria.

We now consider the special case of $m=2$ in equation (6.61) for the general splitting $K=P-Q$. The matrix $M_{2,\alpha}$ is then

$$M_{2,\alpha} = \frac{1}{\alpha_1}P\left(\frac{\alpha_0}{\alpha_1}I + G\right)^{-1} \qquad (6.63)$$

and the 2-step extrapolated preconditioner matrix for the same splitting of K is seen by (6 51) to be

$$M_{2,\gamma} = \frac{1}{\gamma^2}P\left(\frac{2-\gamma}{\gamma}I + G\right)^{-1} \qquad (6.64)$$

It is known, see Chandra[1978], that the same iterates of the PCG method will be obtained with M and any positive constant multiple of M Hence, the extrapolated preconditioner of (6.64) yields the same results as the parametrized preconditioner of (6 63) if

$$\frac{2-\gamma}{\gamma} = \frac{\alpha_0}{\alpha_1} \qquad (6.65)$$

or equivalently,

$$\gamma = \frac{2}{\alpha_0/\alpha_1+1} \qquad (6 66)$$

We note that for $m>2$ such a relationship will not exist between $M_{m,\alpha}$ and $M_{m,\gamma}$ and more research will be required to determine if the parametrized preconditioner is better than extrapolation for these cases.

CHAPTER 7

Parallel Algorithm Analysis

7.1. Execution Time Model

The method of comparing algorithms for serial machines is the standard complexity analysis of the number of arithmetic operations required for completion of the algorithm. For iterative methods this can be broken into the number of operations per iteration times the number of iterations necessary for convergence.

It has been pointed out repeatedly in the literature, see Ortega and Voigt[1977] Buzbee[1978], Grosch[1979], Jones[1980], Hockney[1982], for examples, that this standard complexity analysis is not sufficient to compare parallel algorithms. Additional factors such as data transmissions between processors, processor synchronizations, and global decision making among processors add to the execution time of a parallel algorithm. The number of these overhead operations vary with each algorithm and the number of processors used to solve the problem. In addition, the time required per operation may be a function of the number of processors as well as the hardware/software implementation of the operation on the parallel machine. These considerations suggest that the analysis of a parallel algorithm's performance on a particular machine should include a model for its execution time.

In this chapter an execution time model is developed for analyzing the parallel algorithms in Chapters 5 and 6. The number of arithmetic operations, data transmissions, synchronizations, and flag checks per

iteration are multiplied by the total number of iterations to yield the number of operations of each type. These numbers are then multiplied by the time cost for the respective operation to obtain the total execution time. This execution time is measured in units of one multiplication/addition pair. A detailed description of the model follows:

Let

a = number of multiplication/addition pairs per iteration

b = number of barrier synchronizations per iteration

c = number of colors

d = number of divisions per iteration

e = number of equations of each color per processor

f = number of global flag checks per iteration

g = number of global transmissions per iteration

l = number of interior equations per processor

m = number of steps of m-step PCG

p = number of processors

r = number of receives per iteration

s = number of sum/max circuitry uses per iteration

t = number of local transmissions per iteration

v = number of local convergence tests per iteration

η = maximum number of nonzero entries per row of K

l = number of iterations required for convergence

N = number of equations to be distributed to p processors

The following are the costs of each operation in units of one single precision floating point multiplication/addition pair.

B = cost per barrier synchronization

D = cost per division

F = cost per global flag check

G = cost per global send

R = cost per receive (both local and global)

S = cost per sum/max usage

T = cost per local transmission (send)

V = cost per local convergence test

The formula for the execution time (E) is given by

$$E = I[a + dD + vV + rR + tT + gG + bB + fF + sS] \tag{7.1}$$

This formula can also be used to determine the execution time of a sequential algorithm by setting r, t, g, b, f, and s to zero.

The values of v, b, and f are the same for all the iterative algorithms considered in Chapters 5 and 6. A description of how these values are determined will now be given. The value of $\left\| \underline{u}^k - \underline{u}^{k-1} \right\|_\infty$ must be determined at the kth iteration. If this value is less than a prescribed tolerance ϵ, the iteration terminates, if not, the next iteration is begun. This estimate is determined in two steps. First, each processor compares its portion of $\underline{u}^k$, say $\underline{u}_p^k$, with the corresponding portion $\underline{u}_p^{k-1}$ obtained on iteration $k-1$. If $\left\| \underline{u}_p^k - \underline{u}_p^{k-1} \right\|_\infty < \epsilon$ the processor raises its convergence flag. For a sequential algorithm this convergence criterion requires N comparisons each iteration; whereas, for a parallel algorithm, an equal partitioning of $\underline{u}$ to the p processors allows these comparisons to be performed simultaneously in the processors. Hence the value of v is given by

$$\begin{aligned} v &= N \quad (sequential) \\ v &= N/p \quad (parallel) \end{aligned} \tag{7.2}$$

We note that the complexity analysis of an algorithm on a sequential computer rarely includes the operation counts for the convergence test. However, if $p=O(N)$, (7.2) implies that $v=O(1)$ for a parallel implementation. This can cause a significant reduction in execution time of the parallel algorithm if the convergence test is a costly operation or if the number of iterations is large.

Secondly, the processors must be synchronized at the end of each iteration for the purpose of checking the convergence flags of all the processors. That is,

$$\begin{aligned} b &= 1 \\ f &= 1 \end{aligned} \tag{7.3}$$

On the Finite Element Machine this synchronization is implemented as a barrier whereby each processor uses the signal flag hardware circuit to monitor the synchronization flag on all processors. When this flag is set in all processors, the barrier is lowered and the processors continue with the next instruction which is the global convergence test. To perform this test, each processor uses the signal flag hardware to check the convergence flag in all processors. If all processors have set their flag, the algorithm terminates; if not, the next iteration is begun.

We note that other norms could be used to estimate the error. In particular, the 2-norm would require $(\underline{u}^k-\underline{u}^{k-1})^T(\underline{u}^k-\underline{u}^{k-1})$ to be formed. This would require N multiplications and $N-1$ additions (N additions if the sum is initally set to zero) for a sequential algorithm For a parallel algorithm, N/p multiplications can be done simultaneously by the p

processors and these partial sums loaded onto the sum/max circuit. The circuit would calculate and then return the complete sum (inner product) to each processor. Since the sum/max circuit is not yet operational on FEM and the actual programs run on FEM used the ∞-norm test, the 2-norm will not be considered in the model.

To determine l in (7.1), for the plane stress problem for example, let x and y denote the number of rows and columns of problem nodes assigned to each processor as shown in Figure 1 where lines between processors represent the local links that are used during computation.

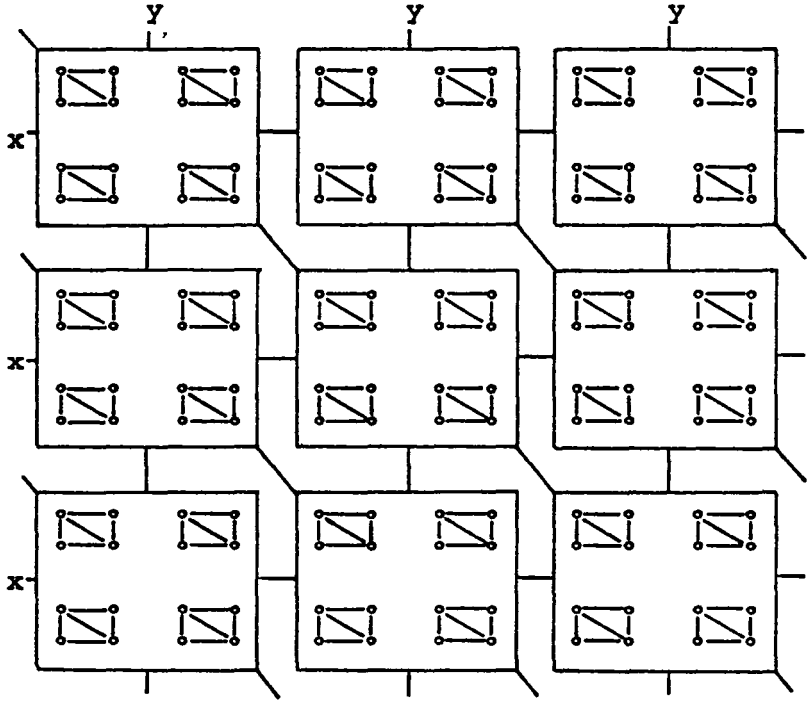


Figure 1. Problem Node Assignment

Furthermore, let d represent the number of equations at each problem node, and assume that the grid is discretized by linear triangular finite elements so that each interior node is on a common finite element with six other nodes (East, West, North, South, Northwest, Southeast).

The number of equations within each processor that correspond to the $\underline{u}$ values that are not communicated to other processors during computation (interior equations) is given by

$$I = \begin{cases} d(x-1)(y-1) & p<9 \\ d(x-2)(y-2) & p\geq 9 \end{cases} \tag{7.4}$$

where we are assuming that the processors are connected in an array fashion like the FEM so that a completely interior processor does not occur until $p=9$.

The values of a, g, s, t and r in (7.1) depend on the particular iterative algorithm used to solve the problem and will be discussed separately in the following sections for the Multi-Color SOR, the Conjugate Gradient, and the m-step (SSOR) Preconditioned Conjugate Gradient algorithms.

7.1.1. Execution Time for Multi-Color SOR

To solve $K\underline{u}=\underline{f}$ by the Multi-Color SOR method given by Algorithm 3 in Chapter 5, at most $\eta-1$ multiplications/addition pairs for each of the N rows of K are required to produce the next iterate and 2 additional multiplications/additions per row to do the over-relaxation. That is,

$$a = N(\eta+1) \tag{7.5}$$

If I represents the number of SOR iterations, the total execution time for the sequential Multi-Color SOR algorithm is given by

$$E = I[N(\eta+1)+NV] \tag{7.6}$$

Now suppose that p processors are available and that p evenly divides N. Then, the arithmetic in (7.5) is divided by p to give

$$a = N(\eta+1)/p \tag{7.7}$$

Chapter 5 showed how to map problems on rectangular grids onto an array of processors so that the Multi-color SOR method only requires a given processor to communicate with at most eight of its nearest neighbors. For example, an interior processor will communicate with its four nearest neighbors (North,South,East,West) during the solution of Laplace's equation if the region is discretized by the usual five-star discretization, and during the solution of the plane stress problem with the domain discretized by linear triangular finite elements, an interior processor will communicate with its North, South, East, West, Northwest, Southeast, neighbors as shown in Figure 1. This means that the global bus and the sum/max circuitry are not required for these discretizations and

$$g = 0 \tag{7.8}$$

$$s = 0 \tag{7.9}$$

The number of local sends for each processor will equal the number of non-interior equations:

$$t = ce - i$$

or equivalently,

$$t = \begin{cases} d(x+y-1) & p<9 \\ d(2x+2y-4) & p \geq 9 \end{cases} \tag{7.10}$$

and the number of values received by each processor per iteration will be the number of non-interior equations for all six neighboring processors:

$$r = \begin{cases} 2d(\alpha+y+1) & p \geq 9 \\ d(\alpha+y+1) & p < 9 \end{cases} \tag{7.11}$$

Thus, the total parallel execution time for Red/Black/Green SOR for the plane stress problem is

$$E = I[(N(\eta+1)+NV)/p + tT + rR + B + F] \tag{7.12}$$

where t, and r are given by (7.10) and (7.11) respectively.

7.1.2. Execution Time for Conjugate Gradient

To solve $K\underline{u}=\underline{f}$ by the conjugate gradient method given by Algorithm 1 in Chapter 6, the following number of multiplications and additions are required: at most ηN multiplications and $\eta(N-1)$ additions for forming $K\underline{p}$, N multiplications and $N-1$ additions for doing each of the inner products $\underline{p}^T K\underline{p}$ and $\underline{r}^T\underline{r}$, N multiplications and N additions for each of the computations $\underline{u}^k+\alpha\underline{p}^k$, $\underline{r}^k-\alpha K\underline{p}^k$, and $\underline{r}^{k+1}+\beta\underline{p}^k$ respectively. In additon, 2 divisions for the calculation of α and β are required. Hence, the total number of arithmetic operations per iteration is given by (7.13)

$$\begin{aligned} a &\leq \eta N + 5N \\ d &= 2 \end{aligned} \tag{7.13}$$

and the total execution time for the sequential conjugate gradient is bounded by

$$E \leq I[\eta N + 5N + 2D + NV] \tag{7.14}$$

Now, suppose that p processors are available and that p evenly divides N. Then the number of multiplication/addition pairs in (7.13) is divided by p to give

$$a = (\eta N + 5N)/p \tag{7.15}$$

but the two divisions must be done by all p processors.

The values of t, r, g, and s in (7.1) depend on the mechanism used for doing the inner products $\underline{r}^T\underline{r}$ and $\underline{p}^T K\underline{p}$. We first give these values for the plane stress problem if a bus (such as the global bus on FEM) alone is used. The values that each processor must communicate during each iteration of the conjugate gradient algorithm are the non-interior $\underline{p}$ values, one partial sum for $\underline{r}^T\underline{r}$, and one partial sum for $\underline{p}^T K\underline{p}$. The non-interior $\underline{p}$ values are sent to the six neighboring processors for a total of $ce-i$ transmissions as given by (7.10). In addition, the partial sums for $\underline{r}^T\underline{r}$ and $\underline{p}^T K\underline{p}$ are sent to the eight local neighbor processors and broadcast over the global bus to the remaining $p-9$ processors. The totals for t and g are given in (7.16) and (7.17) respectively.

$$t = ce - i + 2 \tag{7.16}$$

$$g = \begin{cases} 0 & p \leq 9 \\ 2 & p > 9 \end{cases} \tag{7.17}$$

The non-interior values of $\underline{p}$ from the six local neighbors must be received each iteration. Also the two partial sums $\underline{r}^T\underline{r}$ and $\underline{p}^T K\underline{p}$ must be received from $p-1$ processors and added to the accumulating global sum. If q denotes the ratio of the time to receive and add one number to this global sum to the time to receive the number, the value of r is given by (7.18).

$$r = \begin{cases} 2d(x+y+1) + 2(p-1)q & p \geq 9 \\ d(x+y+1) + 2(p-1)q & p < 9 \end{cases} \tag{7.18}$$

The total execution time for the parallel conjugate gradient algorithm that uses the global bus for inner products is given in (7.19).

$$E = I[(\eta N + 5N + NV)/p + 2D + tT + rR + gG + B + F] \tag{7.19}$$

where t, g, and r are given by (7.16), (7.17), and (7.18) respectively.

Secondly, we assume that a special hardware circuit such as the sum/max circuit on the FEM is available for performing the inner products. Then the partial sums for $\underline{L}^T\underline{L}$ and $\underline{Q}^T K \underline{Q}$ are calculated simultaneously in the processors and then loaded onto the sum/max circuit, summed by the circuit, and the result placed into a special receive buffer in each processor. In particular, the values of t, r, g, and s are:

$$t = ce - I \tag{7.20}$$

$$r = \begin{cases} 2d(x+y+1) & p \geq 9 \\ d(x+y+1) & p < 9 \end{cases} \tag{7.21}$$

$$g = 0 \tag{7.22}$$

$$s = 2 \tag{7.23}$$

Note that the number of receives, r, is no longer $O(p)$ since the global bus is not used. The total execution time for the parallel conjugate gradient algorithm that uses this special hardware for the inner products is given in (7.24).

$$E = I[(\eta N + 5N + NV)/p + 2D + tT + rR + 2S + B + F] \tag{7.24}$$

where t and r are given by (7.20) and (7.21) respectively.

7.1.3. Execution Time for m-Step (SSOR) PCG

The number of multiplication/addition pairs per iteration is equal to the number for standard conjugate gradient plus m times the number for one step of SSOR. Each step of SSOR can be implemented as discussed in Chapter 5 and will require $N(\eta+1)$ multiplications if either overrelaxation or extrapolation is used. The total execution time for the sequential m-step SSOR PCG method is given in (7.25).

$$E = \eta N+5N+2D+NV+mN(\eta+1) \tag{7.25}$$

The addition of the m-step SSOR preconditioner to the standard conjugate gradient algorithm adds extra arithmetic and local send and receive operations. It is important to note that the SSOR algorithm can be implemented as a forward and backward Multi-Color SOR method as described in Chapter 5 and hence no global communication between processors is required for rectangular grids.

The number of multiplications, local sends, and receives required by the m-step preconditioner alone will now be described. The arithmetic in (7.25) due to SSOR is distributed among p processors and is given by (7.26).

$$a = mN(\eta+1)/p \tag{7.26}$$

The only communication that the m-step preconditioner adds to the PCG algorithm is the local communication of the $\hat{r}$ values during the m-step iterative solution of $K\hat{r}=r$. These values must be sent to the six neighbor processors twice for every step of the preconditioner, once for the forward SOR pass and once for the reverse pass as indicated by (7.27).

$$t = 2m(ce-l) \tag{7.27}$$

Likewise, values of $\hat{L}$ must be received from the six neighbor processors for both the forward and reverse SOR pass for every step of the preconditioner. Hence, the number of receives is given by

$$r = \begin{cases} 2d(x+y+1)m & p<9 \\ 4d(x+y+1)m & p\geq 9 \end{cases} \tag{7.28}$$

The total execution time of the parallel m-step PCG algorithm depends on the implementation of the inner products. First, if the partial sums of the inner products are communicated over the global bus, the execution time for the parallel m-step PCG method is given by (7.29).

$$E = C + l[mN(\eta+1)/p + tT + rR] \tag{7.29}$$

where t and r are given by (7.27) and (7.28) respectively and C is the execution time of the parallel conjugate gradient method given in (7.19).

Secondly, we assume a special hardware circuit such as the sum/max circuit on FEM is used to perform the inner products. Then the execution time for the m-step SSOR PCG method is given by (7.29) where, in this case, C represents the execution time of the parallel conjugate gradient method given in (7.24).

7.2. Model Validation

To fully validate the model, problems would need to be solved on a p-processor array such as the Finite Element Machine so that an algorithm's execution time dependence on p could be determined. At this writing, only four processors are operational on the FEM; therefore,

the model can only be partially validated. For instance, algorithms that use only four processors will make use of the local communication links and the signal flag network but will not use the sum/max hardware or the global bus.

The plane stress problem of Chapter 2 was chosen to validate the model on the 4-processor FEM. The plate was discretized into 50 triangular finite elements and the basis functions were chosen to be piecewise linear polynomials. As a result, the displacements were calculated at the vertex nodes shown in Figure 2.

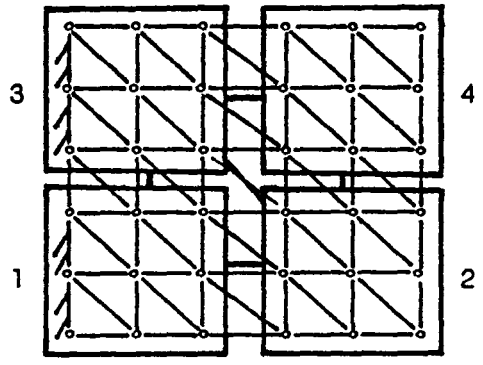


Figure 2. Four Processor Assignment Plane Stress Problem

Furthermore, the plate is constrained on the left edge so that the displacements at the six nodes along this edge are zero. The calculation of the displacements u and v at the remaining 30 nodes must be partitioned to the four processors. Processors 1 and 3 in Figure 2 are assigned 6 nodes, or 12 equations each; whereas, processors 2 and 4 will each solve 18 equations. The speed of the 4-processor FEM is then governed by processors 2 and 4 since more work is assigned to them and as a result we can consider each of the four processors to be solving 18 equations. Therefore, the best possible speedup for this

problem on the 4-processor machine is

$$\frac{60}{72}(4) = 3.33 \tag{7.30}$$

which corresponds to an efficiency of 83%.

The parameters for the current 4-processor FEM that were used in the model were obtained from Tom Crockett at NASA and the arithmetic speeds were gotten from the specifications for the AMD 9512 floating point chip[1979].

1 mult/add pair	= 844μs		
1 division	= 472μs	or	D=0.5592
1 barrier	= 185μs	or	B=0.2192
1 flag check	= 156μs	or	F=0.1848

In addition, the times to send and receive values on the local links were determined by Loendorf and Smith[1982] to be

1 local receive	= 1500μs	or	R=1.7730
1 local send	= 1240μs	or	T=1.4692

and the time for a local convergence check was assumed to equal the time for a multiplication/addition pair (V=1). The speedups obtained on the 4-processor FEM for the 3-Color SOR and standard conjugate gradient algorithm as well as those predicted by the model are given in Table 1.

Method	FEM Speedup	Model Speedup	FEM Efficiency	Model Efficiency
R/B/G SOR	2.84	2.93	71%	73.3%
CG	2.82	2.90	71%	72.5%

Table 1. 4-FEM and Model Results

The efficiency of 71% in Table 1 indicates that the parallel overhead due to communication and synchronization between processors was only 12% of the execution time since 17% efficiency was lost because the number of equations was not evenly divisible by the number of processors. We note that the efficiencies predicted by the model agree very closely with the 4-processor FEM results.

7.3. Model Results

The questions that we will answer with the model are itemized below:

(1) What ratio, α, of communication time to arithmetic time must the array computer have to efficiently support the implementation of an algorithm, and how does this ratio change as p increases?

(2) For a given algorithm and ratio α, what is the maximum number of processors that should be used to achieve a given efficiency level?

(3) If the global bus is used to do the inner products, will the m-step PCG(SSOR) be a more efficient algorithm than the standard conjugate gradient method ($m=0$)?

These questions can be answered by analyzing the speedup of

a p-processor algorithm over its corresponding single processor version and will be discussed in section 7.3.1.

(4) How does an algorithm's performance on an array such as FEM with certain hardware speeds compare with its performance on a benchmark hardware? A measure of this performance will be called the para-efficiency and will be discussed in section 7.3.2.

(5) For a given α, what is the best algorithm for solving the problem as p increases?

(6) For a given algorithm, how does the execution time change as α changes? In particular, below what α level will the execution time fail to decrease significantly?

(7) For a given α, if the global bus is used for the inner products, for what values of m will the execution time of m-step PCG(SSOR) be less than that of standard conjugate gradient ($m=0$)?

The answers to these questions are found by examining the execution times as a function of p and α and are given in section 7.3.3.

(8) What is the tradeoff between a decrease in speed and an increase in the chance of machine failure as p increases? This issue of reliability is discussed in section 7.3.4.

(9) For a given algorithm and ratio α, how many processors are necessary to be competitive with a conventional machine and

problem solver? This question is the topic of section 7.3.5.

The model was used to answer the above questions for the algorithms of Chapter 5 and 6 as applied to the following two test problems. This first problem is Laplace's equation on a rectangular domain with Dirichlet boundary conditions. The domain is discretized into 18 rows and 50 columns of nodes so that the values at the 16 by 48 grid of interior nodes are to be found. The resulting stiffness matrix K has dimension 768 by 768. If the nodes are ordered by the classical Red/Black scheme, the problem can be solved on a machine consisting of the following number of processors which are assigned the corresponding grid sizes.

Processors	Grid size/processor
1	16x48
4	8x24
16	4x12
64	2x 6
128	2x 3
384	1x 2

Table 2. Processor Assignments
Laplace's Equation

The second problem is the plane stress problem of Chapter 2. A rectangular plate is discretized by linear triangular finite elements as shown in Figure 2 so that the displacements, u and v, at 16 rows and 48 columns of nodes must be found. The resulting stiffness matrix K has dimension 1536 by 1536. These nodes are colored Red/Black/Green as described in Chapter 5. This problem can be solved on a machine with the following number of processors by assigning the corresponding grid of nodes to each processor.

Processor	Grid size/processor
1	16x48
4	8x24
16	4x12
64	2x 6
256	1x 3

Table 3. Processor Assignments
Plane Stress Problem

We note that the execution time per iteration of an algorithm is a function of the number of equations per processor as well as the number of processors. If we fix the size of the problem, and allow the number of processors and hence the number of equations per processor to vary, we get results like those in the next five sections. Although not done here, the size of the problem, N, could be varied and the model used to predict the dependence on N for a fixed number of processors. However, for algorithms like Multi-color SOR which do not require any global communication or summation, the execution time is only a function of the number of equations per processor and remains constant as N increases.

7.3.1. Speedup Results

The speedup as a function of the number of processors,

$$Speedup\,(p) = \frac{Execution\ \ time\,(1)}{Execution\ \ time\,(p)}$$

for the Multi-Color SOR, the standard conjugate gradient, and the m-step (SSOR) preconditioned conjugate gradient methods of Chapters 5 and 6 can be predicted by the model for a Finite Element Machine with particular arithmetic and hardware/software communication times. Since speedup

is a measure of how well the architecture of the machine can support the implementation of an algorithm, or conversely, how well an algorithm performs on a particular machine. these arithmetic and communication times can be viewed as design variables for the machine and by changing these times we can determine what ratio of communication to arithmetic the machine must have to efficiently support a given algorithm.

Since the barrier operation and the flag checks occur only once per iteration, the send and receive operations comprise the majority of the communication between processors. Therefore, to analyze the performance of an algorithm as the ratio of communication to arithmetic time changes, we choose to vary T, G, and R in (7.1) and let the values of B, F, S, and D remain constant. Although T, G, and R can have different values, we analyze the case where these values are equal and denote this value by α. We will refer to α as the ratio of communication to arithmetic time.

The model was run with four values of α: namely, 10^{-2}, 10^{-1}, 1, and 10 in order to determine an algorithm's performance over a wide range of values and to aid in machine design. Once this is done, the results can be used to determine a smaller interval for additional model runs. Table 4 gives the four sets of model values where each set is regarded as describing a particular Finite Element Machine.

FEM	Machine	T	G	R	B	F	S	D	V
	1	10.00	10.00	10.00	1.85	1.56	*	.5592	1
	2	1.00	1.00	1.00	1.85	1.56	*	.5592	1
	3	.10	.10	.10	1.85	1.56	*	.5592	1
	4	.01	.01	.01	1.85	1.56	*	.5592	1

Table 4. Four Sets of Model Costs
* (see 7.3.4)

The times to perform a barrier synchronization and a global flag test were taken to be the times for these operations on the current machine. An approximate time for one use of the sum/max circuit was given by Jordan, et.al.[1979] to be

$$48 + \log_2 p \quad \mu s \tag{7.3.3}$$

To this was added the time to place a value on the circuit (assumed to be one send) and the time to read the sum from the circuit (assumed to be one receive). For example, if the time for 1 multiplication/addition pair is 10^{-4} seconds, the value of S is given by

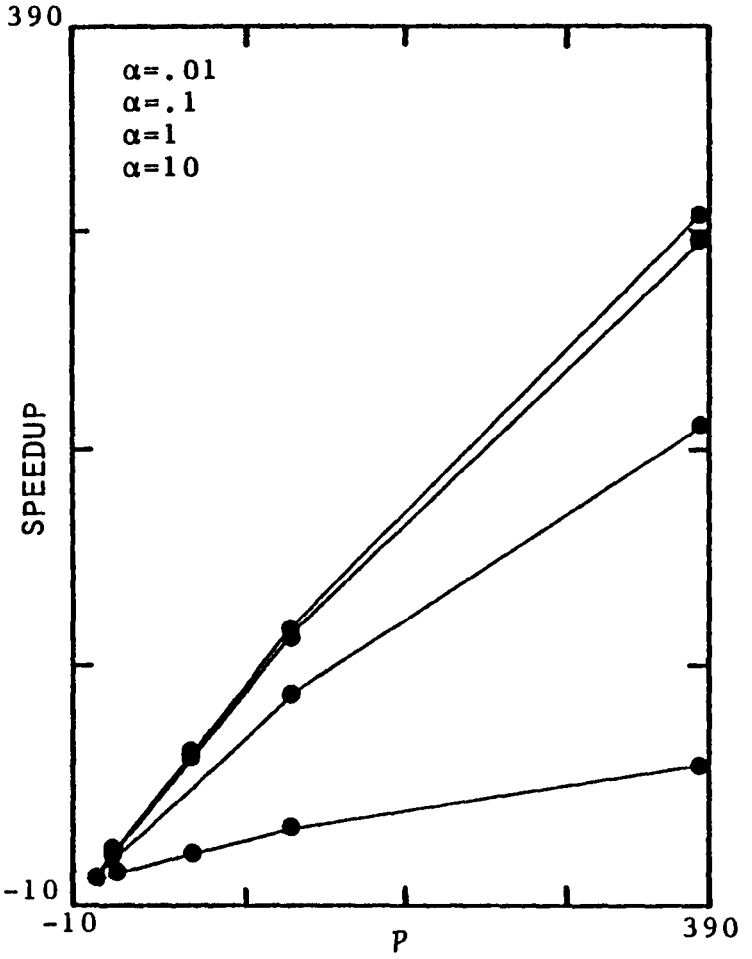
$$S = T + (0.01)(48+\log_2 p) + R \tag{7.3.4}$$

For each of the machines in Table 4, Speedup(p) was determined for the R/B SOR, standard conjugate gradient (Global Bus), standard conjugate gradient (Sum/Max), and R/B 1,2-step SSOR PCG (Bus) and (Sum/Max) methods for the solution of Laplace's equation. These speed-ups are given in Table 5.

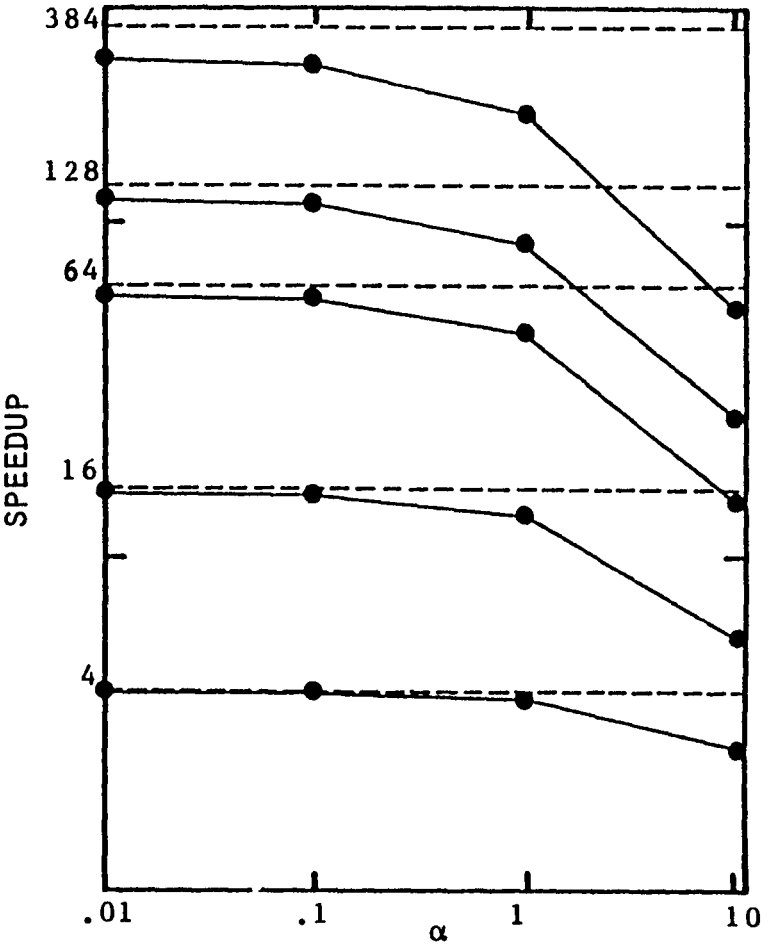
			PCG(SSOR) (Bus)			PCG(SSOR) (S/M)		
α	p	R/B SOR	m=0	m=1	m=2	m=0	m=1	m=2
	4	2.7	3.0	2.4	2.5	3.0	2.4	2.5
	16	5.7	5.7	4.0	4.5	7.2	4.5	4.8
10	64	14.6	4.9	5.0	6.7	18.5	10.8	11.7
	128	26.2	3.0	3.7	5.6	31.2	18.8	20.6
	384	55.2	1.1	1.4	2.5	57.7	36.6	41.6
	4	3.8	3.9	3.7	3.8	3.9	3.7	3.7
	16	13.5	13.5	12.3	12.7	14.2	12.7	12.9
1	64	46.6	28.8	29.0	34.7	50.1	42.3	43.9
	128	87.5	24.6	28.8	41.2	93.3	78.6	82.8
	384	211.0	10.5	13.9	23.9	219.2	184.2	202.2
	4	4.0	4.0	4.0	4.0	4.0	4.0	4.0
	16	15.6	15.6	15.5	15.6	15.7	15.5	15.6
.1	64	59.6	55.9	56.3	58.5	60.5	59.6	60.5
	128	114.4	87.3	92.9	104.1	116.5	115.5	118.6
	384	295.2	81.9	102.1	149.8	304.5	308.5	329.5
	4	4.0	4.0	4.0	4.0	4.0	4.0	4.0
	16	15.8	15.9	15.9	15.9	15.9	15.9	15.9
.01	64	61.3	61.7	62.1	62.8	61.8	62.2	62.9
	128	117.9	117.1	119.4	122.9	119.5	121.2	123.9
	384	307.4	254.6	278.6	316.5	316.9	330.8	351.6

Table 5. Speedups for Laplace's Equation

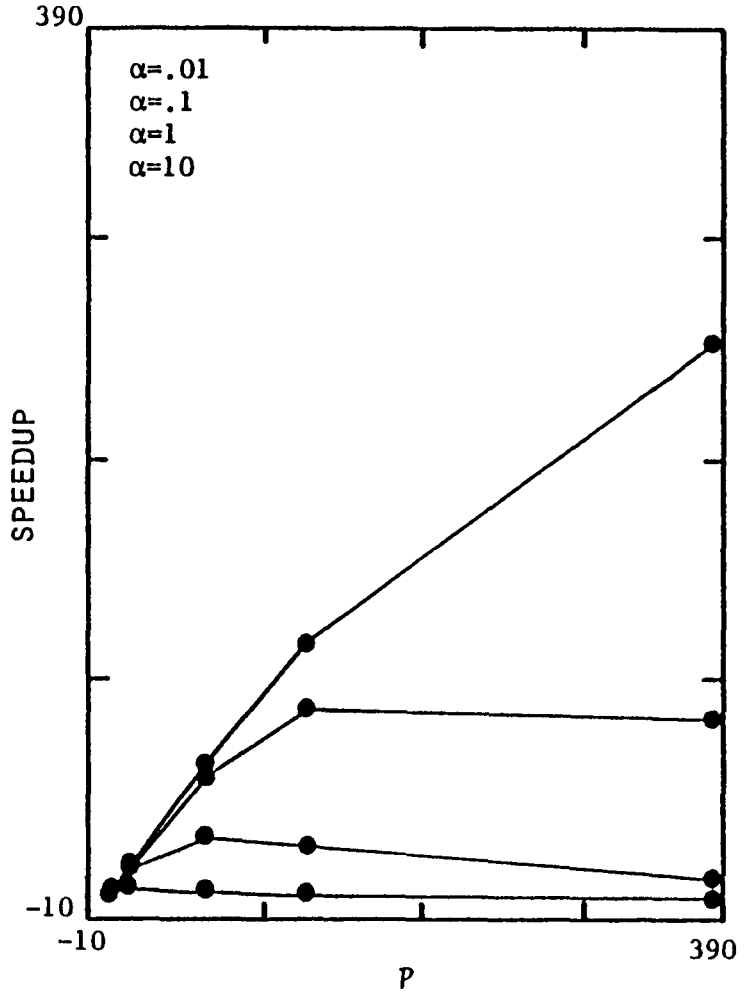
Graphs 1A, 2A, and 3A show speedup as a function of p, the number of processors, for the R/B SOR, the CG(Bus), and the R/B 2-step SSOR PCG(Bus) algorithms respectively. Each of these graphs shows the cases α=.01, α=.1, 1 and 10 from top to bottom of the graph respectively. Graphs 1B, 2B, and 3B show the speedup as a function of the ratio α for the same algorithms. Each of these graphs shows the cases of p=4,16,64,128, and 384 and the five dotted lines indicate perfect speedup for these values of p. Graphs 1A and 1B show for R/B SOR that p=384 processors solve the problem in the shortest time. Also note from Graph 1A that only a slight improvement in speedup is seen for α=0.01 over α=0.1; whereas, a large drop in speedup can be seen by



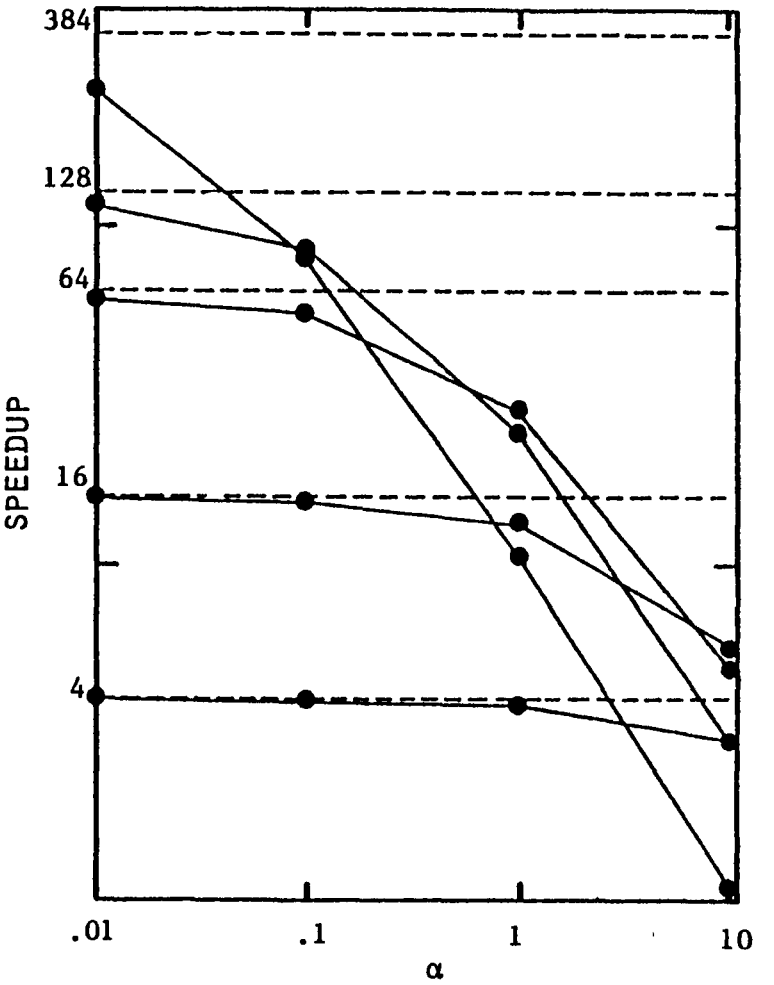
GRAPH 1A. R/B SOR



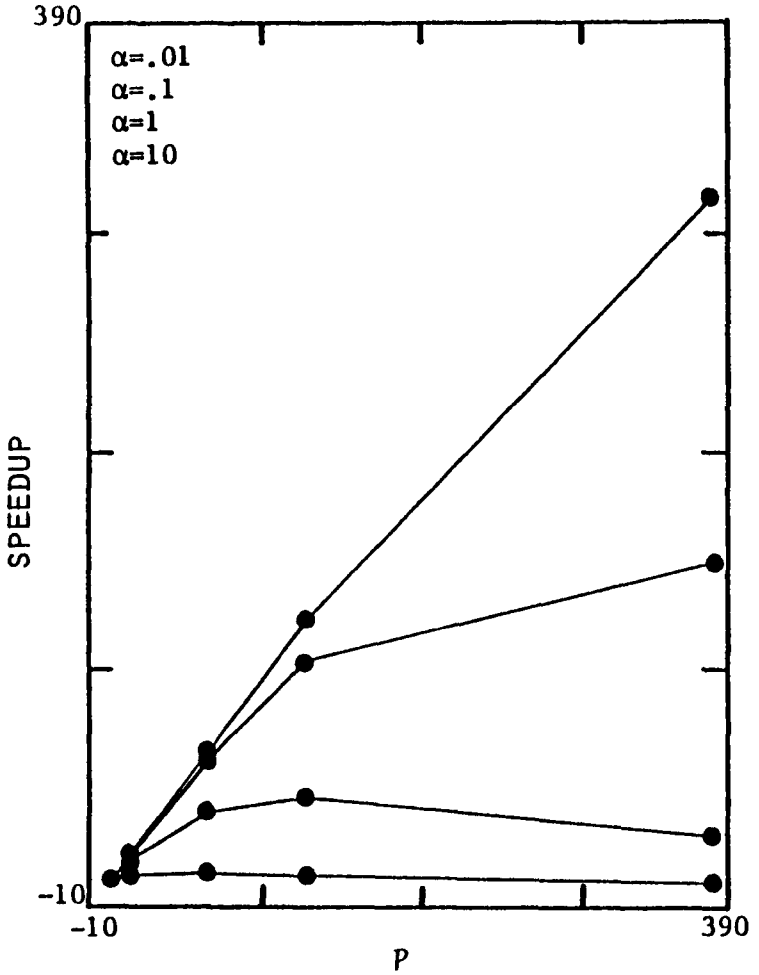
GRAPH 1B. R/B SOR



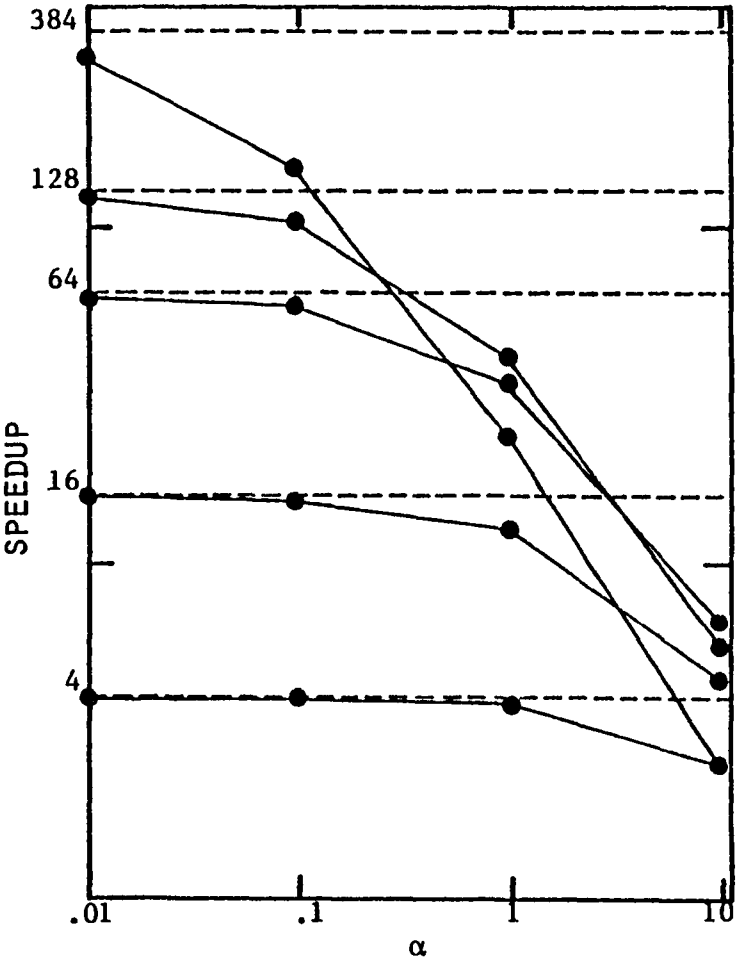
GRAPH 2A. CG(BUS)



GRAPH 2B. CG(BUS)



GRAPH 3A. R/B SSOR 2-PCG(BUS)



GRAPH 3B. R/B SSOR 2-PCG(BUS)

increasing α from 1 to 10.

However, for CG(Bus), a different story can be seen from Graph 2A and 2B. The number of processors that solves the problem in the shortest time is not constant and in fact varies drastically with α. Note that the largest speedup is obtained with p=384 only for α=0.01 while p=128 gives the largest speedup only for α=0.1. At the level α=1, 64 processors gives the maximum speedup and at α=10, the speedup decreases for more than 16 processors. Also note that for p=4 and p=16, the efficiency is quite good for α=1; whereas for p=64 and 128, α=0.1 is strongly preferred over α=1 and 384 processors should only be considered if α is 0.01 or less.

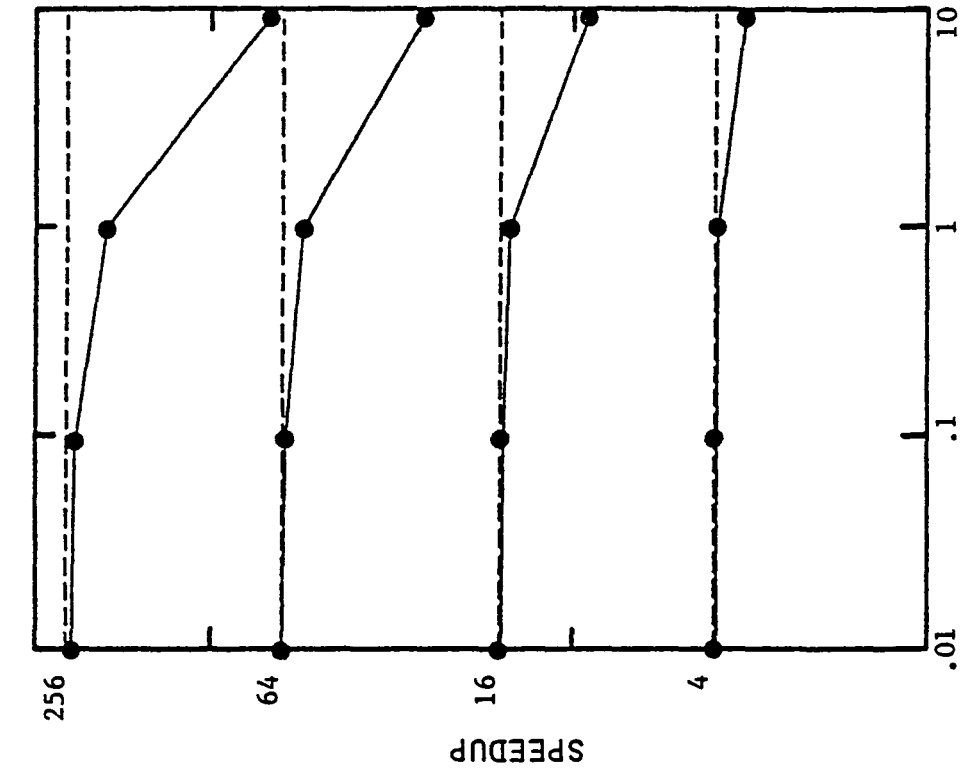
By comparing Graphs 3A and 3B with Graphs 2A and 2B, we see that some improvement over CG(Bus) can be achieved by using 2 steps of SSOR PCG(Bus) and m=2 is seen to be best the value of m from Table 5. In particular, p=384 gives the best efficiency at the α=0.1 level for the 2-step SSOR PCG(Bus) algorithm whereas p=128 was best for CG(Bus). In addition, 64 processors give the best speedup at the α=10 level for the 2-step SSOR PCG(Bus) algorithm whereas p=16 gave the best speedup for this value of α for the *CG(Bus)* algorithm.

The speedups for the same algorithms for the plane stress problem are given in Table 6.

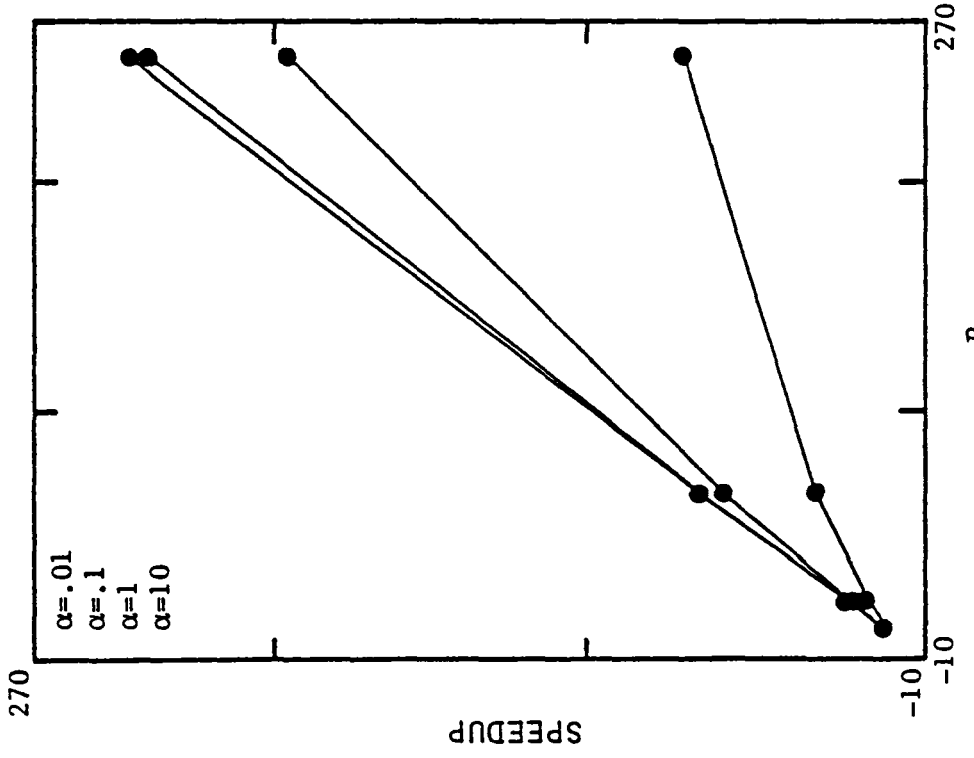
			PCG(SSOR) (Bus)			PCG(SSOR) (S/M)		
α	p	R/B/G SOR	m=0	m=1	m=2	m=0	m=1	m=2
	4	3.3	3.4	3.1	3.1	3.4	3.1	3.1
10	16	9.0	8.9	7.1	7.2	9.7	7.4	7.4
	64	25.9	13.1	13.4	15.4	28.3	20.1	20.0
	256	68.4	5.6	8.3	12.3	72.3	49.6	49.7
	4	3.9	3.9	3.9	3.9	3.9	3.9	3.9
1	16	14.8	14.8	14.2	14.3	15.0	14.3	14.3
	64	55.4	45.9	46.4	48.5	56.4	52.3	52.4
	256	196.0	46.3	63.9	85.4	198.8	178.2	179.3
	4	4.0	4.0	4.0	4.0	4.0	4.0	4.0
.1	16	15.8	15.8	15.8	15.8	15.9	15.8	15.8
	64	62.5	61.2	61.4	61.9	62.6	62.3	62.4
	256	240.9	173.2	194.5	211.7	240.9	240.5	242.6
	4	4.0	4.0	4.0	4.0	4.0	4.0	4.0
.01	16	16.0	16.0	16.0	16.0	16.0	16.0	16.0
	64	63.3	63.3	63.5	63.5	63.3	63.5	63.6
	256	246.6	238.5	244.5	248.5	246.1	249.3	251.4

Table 6. Speedups for the Plane Stress Problem

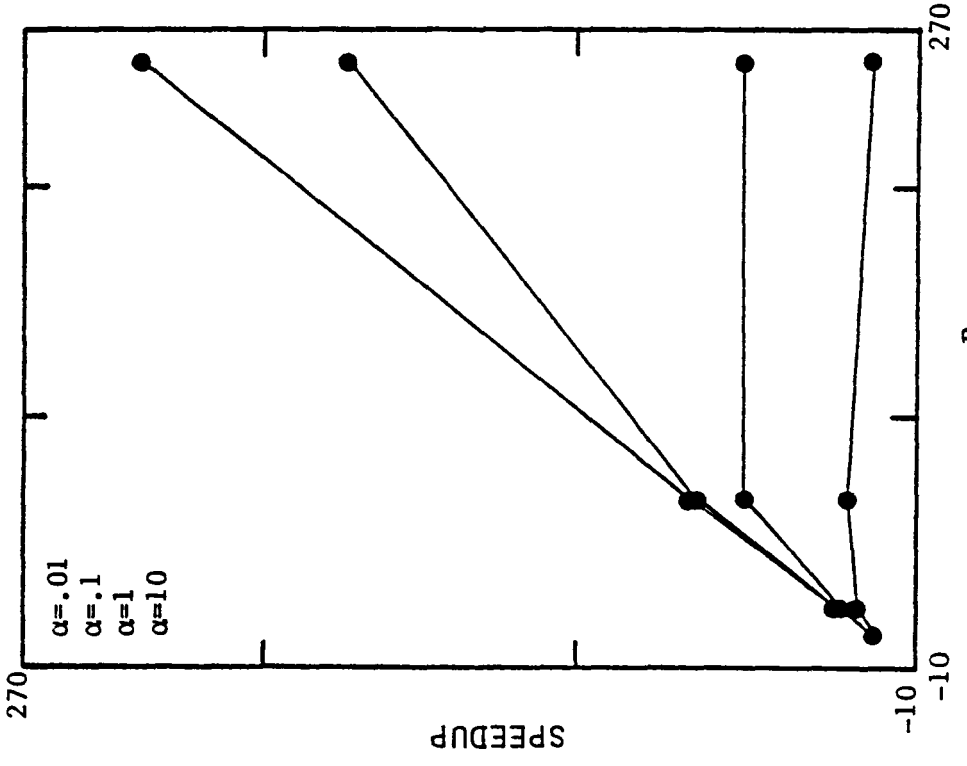
Graphs 4A, 5A, 6A, and 7A show speedup as a function of p, the number of processors, for the R/B/G SOR, the CG(Bus), the R/B/G 2-step SSOR PCG(Bus), and the R/B/G 2-step SSOR PCG(Sum/Max) algorithms respectively. Each of these graphs shows the cases α=.01, α=.1, α=1 and α=10 from top to bottom of the graph respectively. Graphs 4B, 5B, 6B, and 7B show speedup as a function of α for the same algorithms where the four dotted horizontal lines respresent perfect speedup for p=4, 16, 64, and 256. From Graphs 4A and 4B the R/B/G SOR algorithm is seen to be very efficient on machines with α as large as 1 for $p \leq 64$. Even though the speedup drops drastically for α=10, p=256 processors still solve the problem faster. From Graphs 5A and 5B and 6A and 6B, we see a situation similar to Graphs 2A and 2B and 3A and 3B for Laplace's equation; namely, if CG(Bus) is used, p=256 is



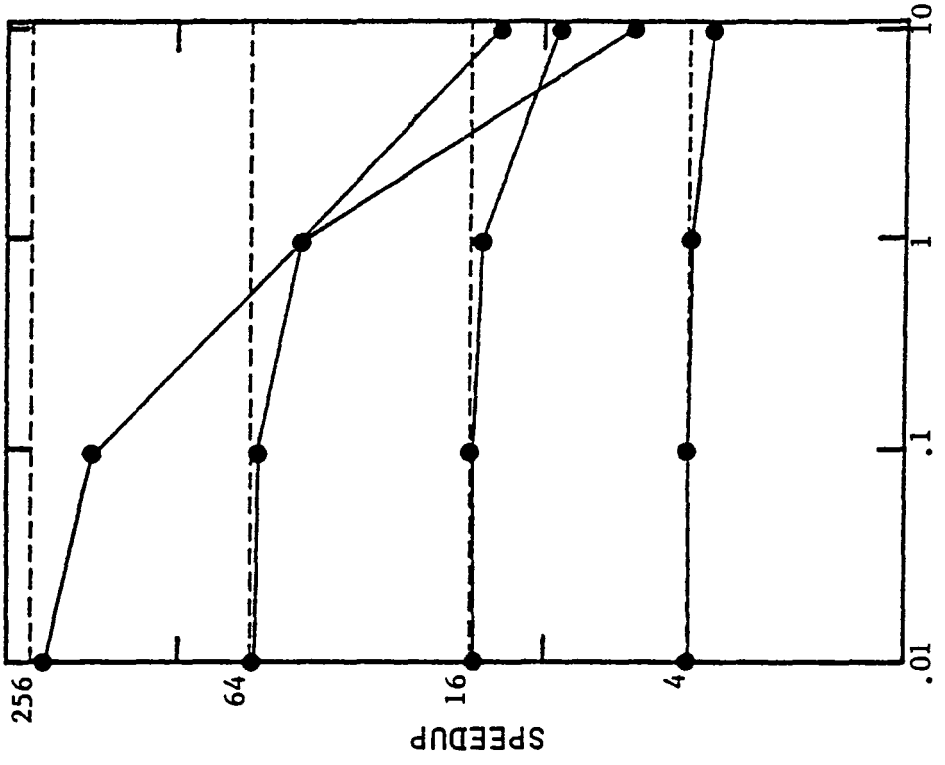
GRAPH 4B. R/B/G SOR



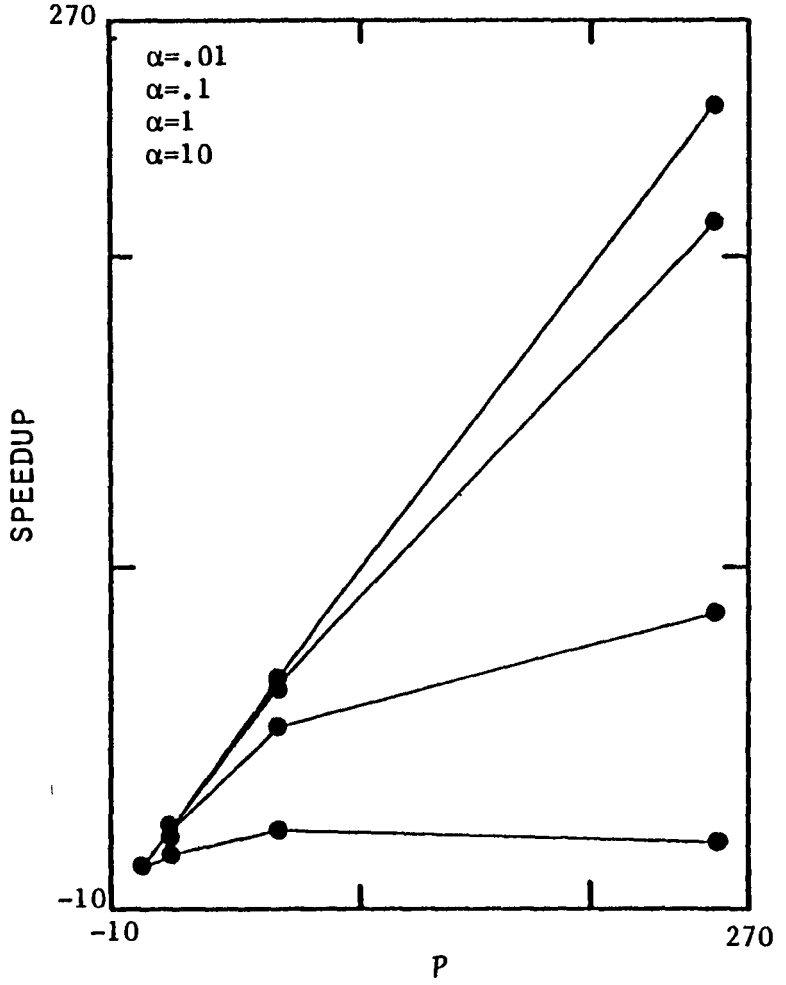
GRAPH 4A. R/B/G SOR



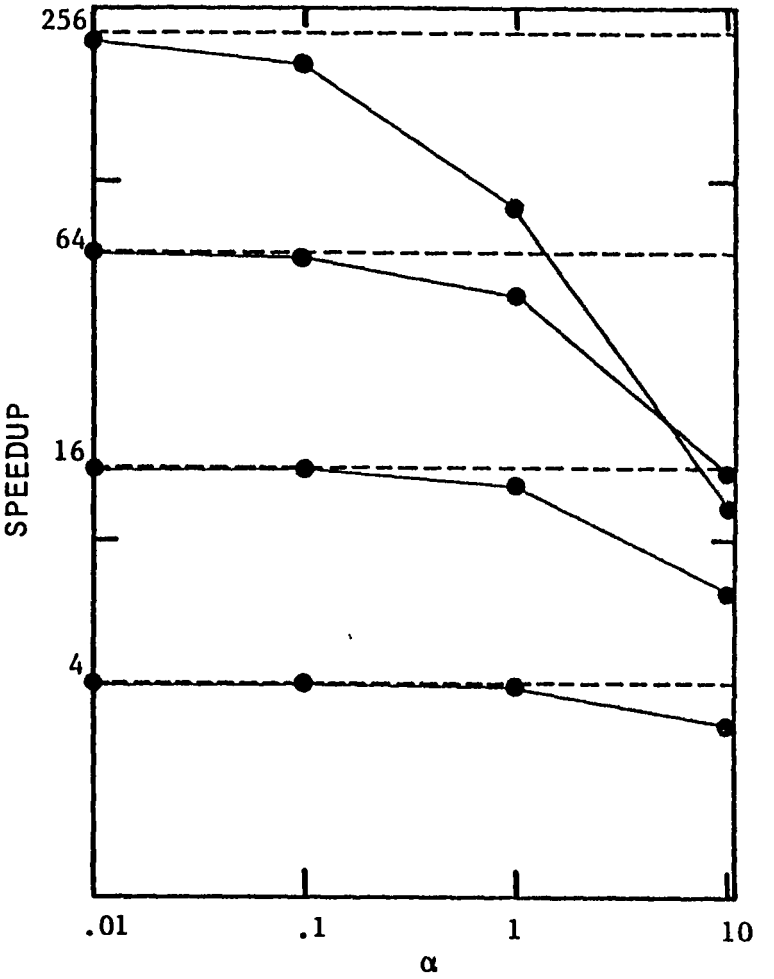
GRAPH 5A. CG(BUS)



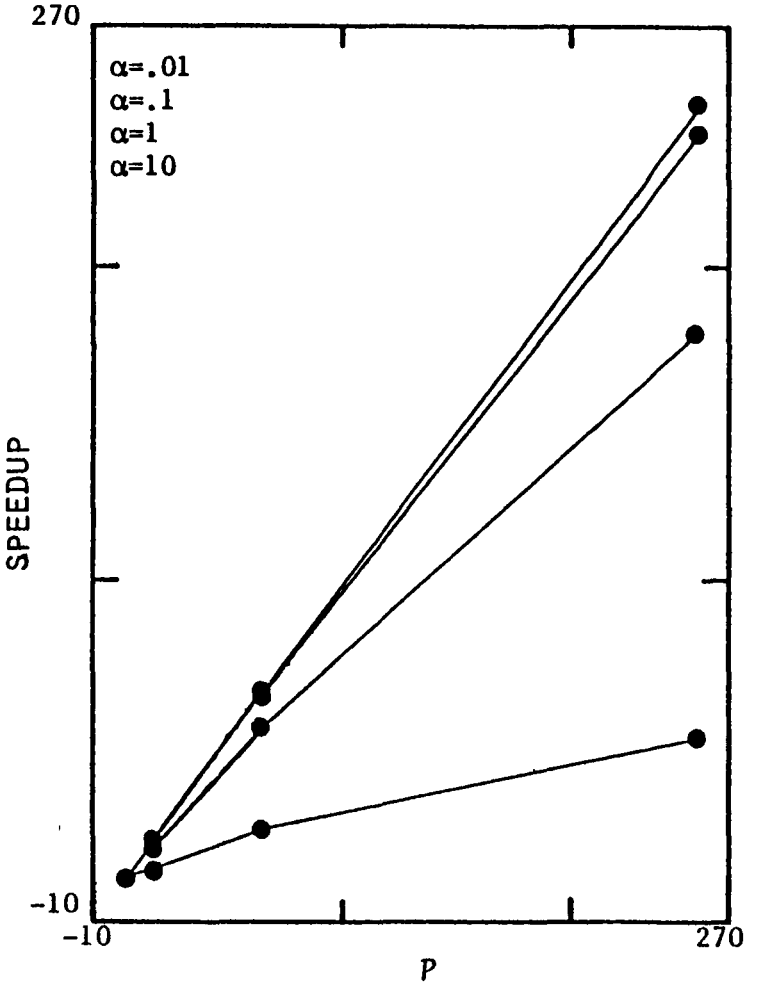
GRAPH 5B. CG(BUS)



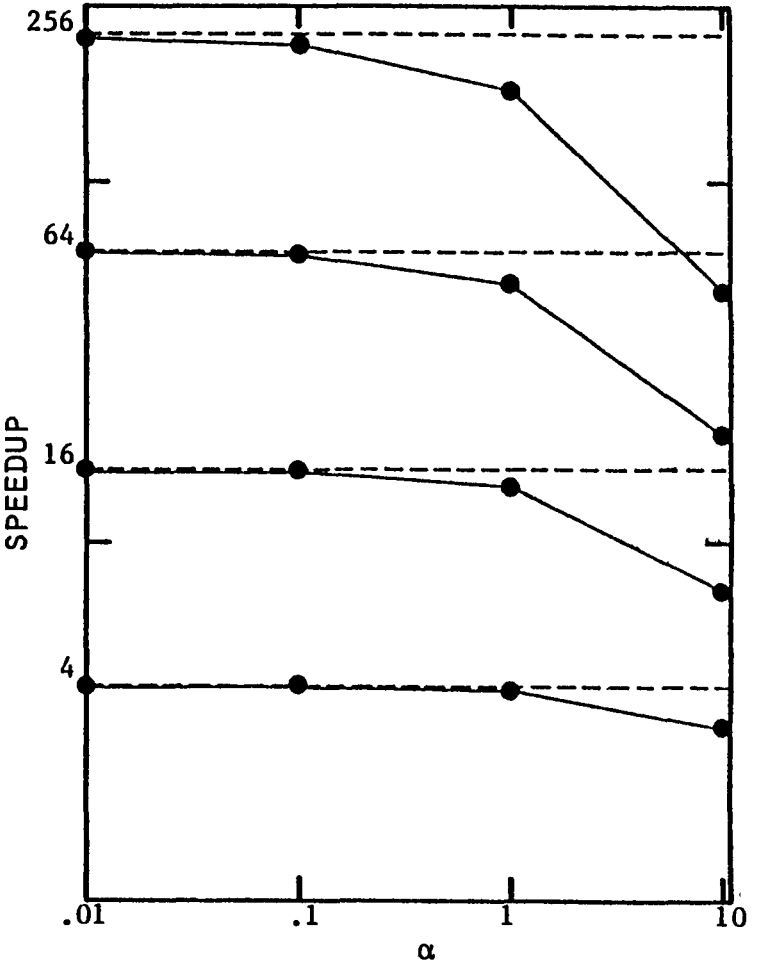
GRAPH 6A. R/B/G SSOR 2-PCG(BUS)



GRAPH 6B. R/B/G SSOR 2-PCG(BUS)



GRAPH 7A. R/B/G SSOR 2-PCG(SUM/MAX)



GRAPH 7B. R/B/G SSOR 2-PCG(SUM/MAX)

preferred at the α=.1 level, p=64 at the α=1 level (perhaps α=10 also). From Graph 6A and 6B, we see that with the use of 2-step SSOR PCG(Bus) algorithm, p=256 gives the largest speedup up to level α=1, but for α=10, p=64 is the maximum number of processors to use.

Graph 7A and 7B show that the sum/max hardware circuit greatly improves the efficiency of the 2-step SSOR PCG algorithm. In particular, p=256 solves the problem in the fastest time for all values of α.

It should be noted that speedup is not a viable measurement to compare parallel algorithms since the algorithm that can be implemented with the least parallel overhead (most efficient) may still take longer to execute due to extra arithmetic calculations. This was the case with R/B/G SOR applied to the plane stress problem. The algorithm is very efficient for all values of α but takes too long to converge to a solution to be competitive with any of the conjugate gradient type algorithms. Execution time comparisons are given in section 7.3.3.

7.3.2. Para-efficiency

Schwartz[1979] recommended the para-efficiency

$$PEFF(p) = \frac{E(p,H)}{E(p,p\text{-}array)}$$

as a good measure of an algorithm's performance on a particular hardware configuration H. The para-efficiency is the ratio of the execution time of an algorithm on a particular hardware, H, to the execution time on the p-array. The p-array is an unrealizable hardware configuration in which all p processors are connected to each other and a central shared memory. The time to write into and read from this memory is assumed to be negligible compared with arithmetic time. Also no

memory contention is assumed for the p-array. This p-array is envisioned as an ideal architecture for p processors since the overhead due to passing data between them is as small as possible.

The FEM can be envisioned as a p-array if the send time is equal to the time to write to memory and the receive time is equal to the time to read from memory. Note that FEM never has the problem of memory contention. Let us define our p-array, or benchmark computer, to be a FEM for which send and receive operations over the local links and the global bus takes $0.1\mu s$ each and the barrier and flag tests require the same time as given in Table 4. This definition views our benchmark computer to be nearly ideal in the sense of sending and receiving data but communication overhead can still occur in the form of barrier or flag checking operations. This is consistent with the assumption in Table 4 that the values of B, F, S, and D are constant and that the major overhead is due to the send and receive operations The $PEFF(p)$ values for the machine configurations of the last section are given in Table 7 for Laplace's equation.

α	p	R/B SOR	PCG(SSOR) (Bus) m=0	PCG(SSOR) (Bus) m=2	PCG (SSOR) (S/M) m=0	PCG (SSOR) (S/M) m=2
	4	1.5	1.3	1.6	1.3	1.6
	16	2.8	2.8	3.5	2.2	3.3
10	64	4.2	12.7	9.4	3.3	5.4
	128	4.5	40.3	22.3	3.8	6.0
	384	5.6	298.0	142.5	5.5	8.5
	4	1.0	1.0	1.1	1.0	1.0
	16	1.2	1.2	1.3	1.1	1.2
1	64	1.3	2.2	1.8	1.2	1.5
	128	1.4	4.9	3.0	1.3	1.5
	384	1.5	30.7	14.9	1.5	1.8
	4	1.0	1.0	1.0	1.0	1.0
	16	1.0	1.0	1.0	1.0	1.0
.1	64	1.0	1.1	1.1	1.0	1.1
	128	1.0	1.4	2.1	1.0	1.1
	384	1.1	3.9	2.4	1.0	1.1
	4	1.0	1.0	1.0	1.0	1.0
	16	1.0	1.0	1.0	1.0	1.0
.01	64	1.0	1.0	1.0	1.0	1.0
	128	1.0	1.0	1.0	1.0	1.0
	384	1.0	1.3	1.1	1.0	1.0

Table 7. Para-efficiencies for Laplace's Equation

Several conclusions follow from Table 7.

(1) α=.01 is virtually as good as the p-array for all the algorithms.

(2) For α=0.1, an array like the FEM supports very efficiently all algorithms that do not use the bus. In addition, the R/B SSOR PCG(Bus) algorithm is very efficient for $p \leq 64$.

(3) For α=1, the bus should only be considered when $p \leq 16$.

(4) For α=10, only p=4 yields a good para-efficiency.

The values of $PEFF(p)$ are given in Table 8 for the plane stress problem.

		PCG(SSOR) (Bus)		PCG (SSOR) (S/M)	
α	p	m=0	m=2	m=0	m=2
	4	1.2	1.3	1.2	1.3
10	16	1.8	2.2	1.6	2.2
	64	4.8	4.1	2.2	3.2
	256	44.6	20.5	3.4	5.0
1	4	1.0	1.0	1.0	1.0
	16	1.1	1.1	1.1	1.1
	64	1.4	1.3	1.1	1.2
	256	5.4	3.0	1.2	1.4
	4	1.0	1.0	1.0	1.0
.1	16	1.0	1.0	1.0	1.0
	64	1.0	1.0	1.0	1.0
	256	1.4	1.2	1.0	1.0
	4	1.0	1.0	1.0	1.0
.01	16	1.0	1.0	1.0	1.0
	64	1.0	1.0	1.0	1.0
	256	1.0	1.0	1.0	1.0

Table 8. Para-efficiencies for Plane Stress Problem

Several conclusions follow from Table 8.

(1) α=.01 is virtually as good as the p-array for all the algorithms.

(2) α=0.1 is virtually as good as the p-array for all algorithms that do not use the bus.

(3) For α=1 and α=10, the para-efficiency is more dependent on p;hence, the sum/max circuitry is becoming more important for larger p.

(4) m=0 is more efficient for α=1 and α=10 even though m=2 is seen from the next section to yield a smaller execution time.

7.3.3. Execution Time Results

If we consider factors such as simplicity and maintainability to be equal for all our algorithms, the best parallel algorithm to solve a given

problem will naturally be the one that requires the least time to execute. The execution times for the parallel algorithms are given in Table 9 for Laplace's equation.

			PCG(SSOR) (Bus)			PCG(SSOR) (S/M)		
α	p	R/B SOR	m=0	m=1	m=2	m=0	m=1	m=2
	1	30.1	47.3	34.6	35.3	47.3	34.6	35.3
	4	11.1	15.8	14.6	14.3	15.6	14.4	14.2
10	64	2.1	9.6	7.0	5.1	2.6	3.2	3.0
	128	1.2	15.7	9.5	6.0	1.5	1.8	1.7
	384	.55	43.7	23.9	13.9	.82	.94	.85
	4	7.9	12.2	9.2	9.4	12.2	9.2	9.4
	16	2.2	3.5	2.8	2.8	3.3	2.7	2.7
1	64	.65	1.6	1.2	1.0	.94	.82	.80
	128	.34	1.9	1.2	.86	.51	.44	.43
	384	.14	4.5	2.5	1.5	.22	.19	.17
	4	7.6	11.9	8.7	8.9	11.9	8.7	8.9
	16	1.9	3.0	2.2	2.3	3.0	2.2	2.3
.1	64	.51	.85	.61	.60	.78	.58	.58
	128	.26	.54	.37	.34	.41	.30	.30
	384	.10	.58	.34	.24	.16	.11	.11
	4	7.5	11.9	8.7	8.8	11.9	8.7	8.8
	16	1.9	3.0	2.2	2.2	3.0	2.2	2.2
.01	64	.49	.77	.56	.56	.77	.56	.56
	128	.26	.40	.29	.29	.40	.29	.28
	384	.10	.19	.12	.11	.15	.10	.10

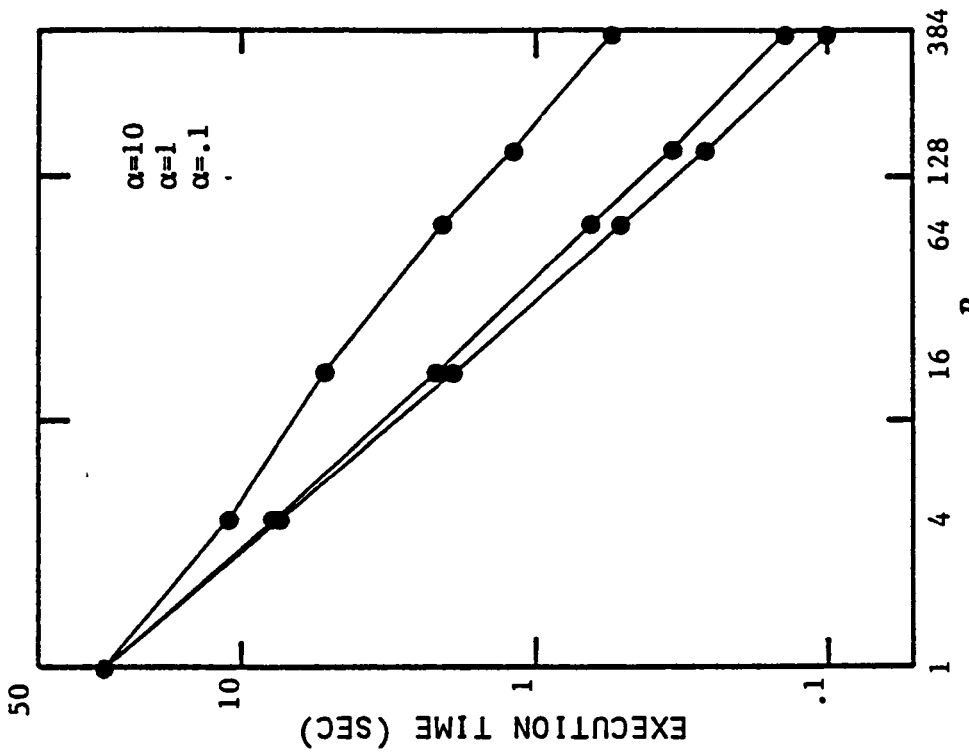
Table 9. Execution Times (sec) for Laplace's Equation
(1 mult/add = 0.0001 sec.)

Graphs 8A and 8B show the execution time (seconds) versus p, the number of processors, for the R/B SOR and the m=0,1,2 step PCG(SSOR)(S/M) algorithms for α=10 and α=1 respectively (for α=1, m=2 is not shown). R/B SOR is seen to be the fastest method for Laplace's equation for both α levels. It is interesting to note from Graph 8A that there exists a value of p between 4 and 16 for which taking more than

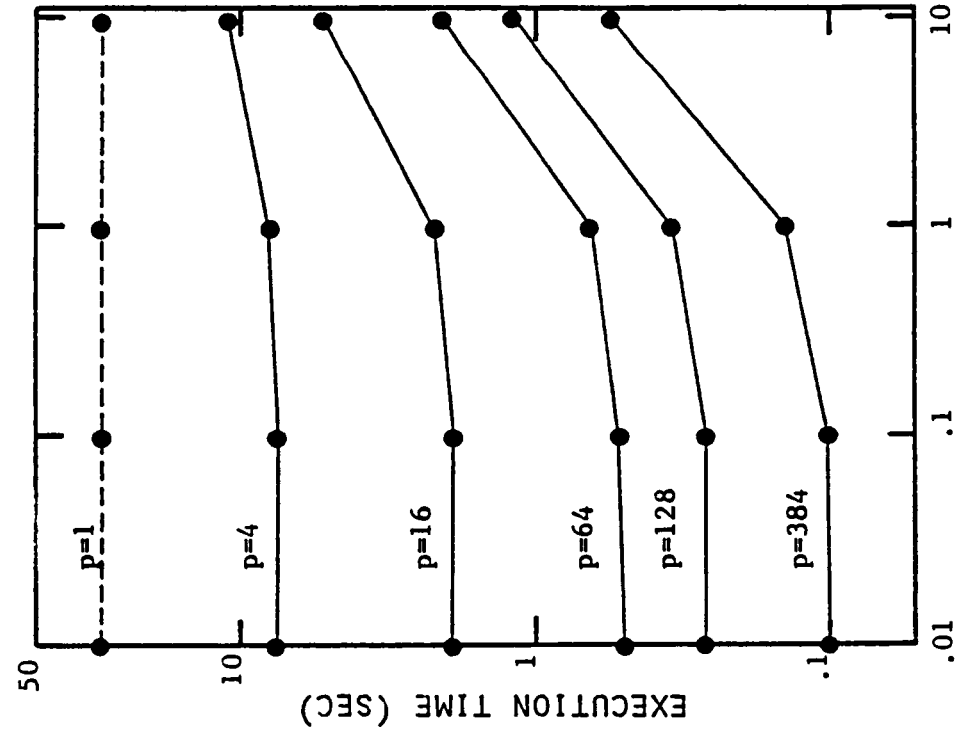
0 steps of m-step PCG(SSOR) is not cost effective. The reason for this is that the time to do the extra communication required for the preconditioner is more than the time gained by the fewer number of iterations. Note that when the cost of communication is reduced as in Graph 8B, the ranking of algorithms for all p is the same as the ranking for the case $p=1$. Furthermore, this ranking will continue as α decreases.

Graph 9A shows the execution time (seconds) versus α for the R/B SOR algorithm for the cases $p=1$(dotted), $p=4$, $p=16$, $p=64$, $p=128$ and $p=384$. Graph 9B shows execution time versus p for the cases $\alpha=.1$, $\alpha=1$ and $\alpha=10$ for the R/B SOR algorithm. Note that $p=384$ gives the least exeuction time for all α levels. Both graphs show a slight increase in execution time from increasing α from .1 to 1 for $p \geq 64$ and a larger increase from increasing α from 1 to 10 for all values of p. Graph 9A also can be used to help answer the question of tradeoff between faster communication and more processors. For example, for a machine with $\alpha=10$ and $p=64$, a greater reduction in execution time can be realized by going to a $\alpha=1, p=64$ machine (faster communication) rather than adding more processors to get an $\alpha=10, p=128$ or possibly $p=384$ machine.

An examination of Graph 9B shows for $\alpha=10$, the graph changes convexity between $p=4$ and $p=64$. In other words, the execution time decreases more from $p=16$ to $p=64$ than from $p=4$ to $p=16$. The reason for this is that when $p=16$ there are completely interior processors and the number of local receives double (7.11) and the number of sends virtually double (7.10). When $\alpha=10$ this factor is amplified but for $\alpha=1$ or $\alpha=.1$, the communication cost is much less and this factor is not noticable.



GRAPH 9B. R/B SOR



GRAPH 9A. R/B SOR

Graph 9B also illustrates that the execution time reduces by much more when decreasing α from 10 to 1 than the reduction seen when α is reduced from 1 to 0.1. This suggests that there will be an optimal value of α below which the gains in execution time reduction will not justify the cost of making the machine's communication faster. In fact, α=.01 gives almost identical execution time results as α=.1 for R/B SOR and was not included in Graph 9B.

The execution times for the parallel algorithms for the plane stress problem are given in Tables 10A and 10B for the global bus and sum/max circuit respectively.

PCG(SSOR)
(Bus)

α	p	m=0	m=1	m=2	m=3	m=4
	1	1115.0	705.0	597.0	862.0	798.0
	4	328.0	230.0	195.0	286.0	267.0
10	16	126.0	99.0	83.0	123.0	116.0
	64	85.0	52.0	39.0	55.0	50.0
	256	201.0	85.0	49.0	58.0	47.0
	4	284.0	182.0	154.0	222.0	206.0
1	16	75.0	50.0	42.0	61.0	56.0
	64	24.3	15.2	12.3	17.6	16.2
	256	24.1	11.0	7.0	8.9	7.5
	4	279.0	177.0	150.0	216.0	200.0
.1	16	70.0	45.0	38.0	55.0	51.0
	64	18.1	11.5	9.7	13.9	12.9
	256	6.4	3.6	2.8	3.9	3.3
	4	279.0	176.0	149.0	215.0	200.0
.01	16	70.0	44.0	37.0	54.0	50.0
	64	17.6	11.1	9.4	13.5	12.5
	256	4.7	2.8	2.4	3.4	3.2

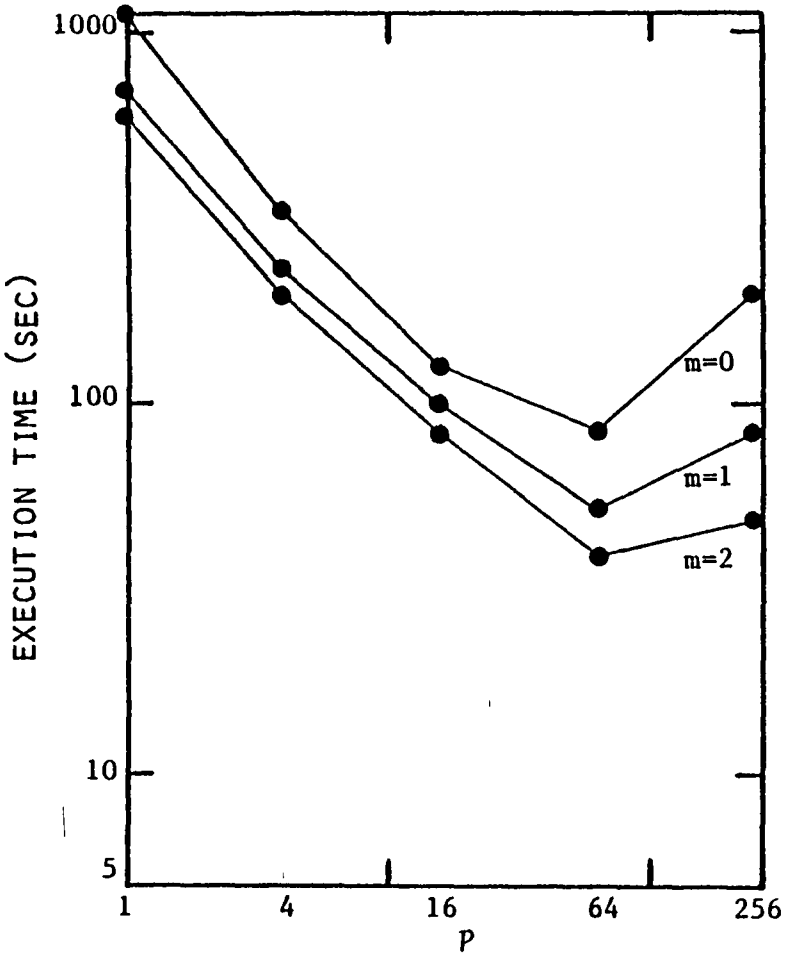
Table 10A. Execution Time (seconds)
Plane Stress Problem (Bus)
(1 mult/add = 0.0001 sec)

PCG(SSOR)
(Sum/Max)

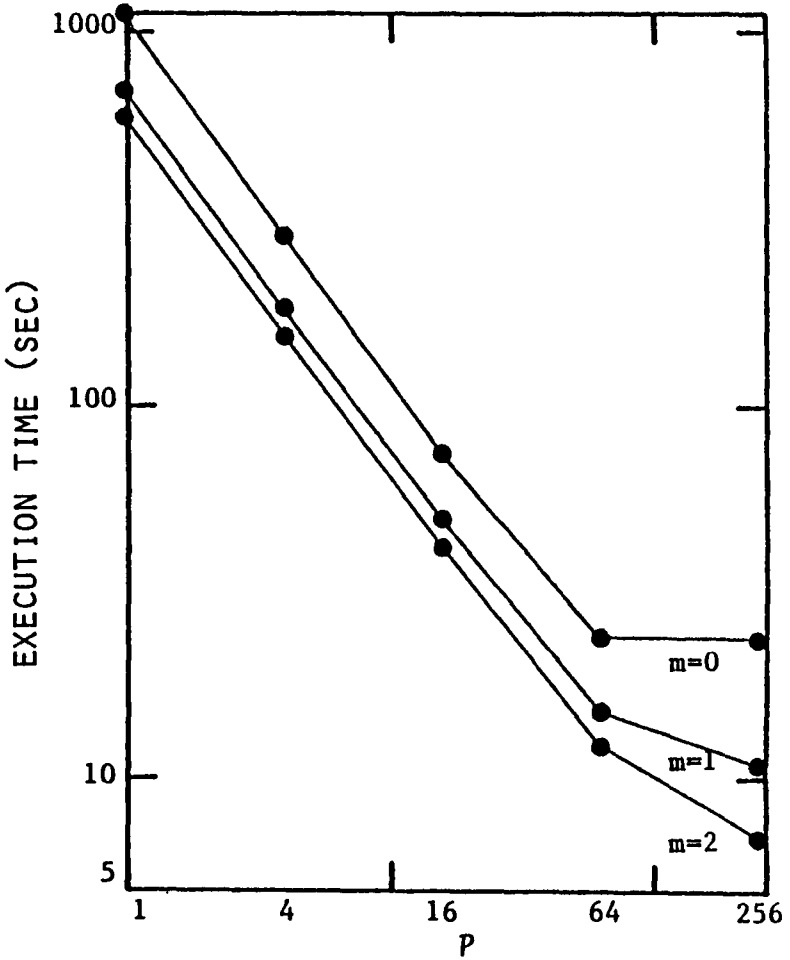
α	p	m=0	m=1	m=2	m=3	m=4
	1	1115.0	705.0	597.0	862.0	798.0
	4	326.0	229.0	195.0	285.0	267.0
10	16	115.0	95.0	81.0	121.0	114.0
	64	39.0	35.0	30.0	45.0	43.0
	256	15.0	14.0	12.0	18.0	17.0
	4	284.0	182.0	154.0	222.0	206.0
1	16	74.0	49.0	42.0	61.0	56.0
	64	19.8	13.5	11.4	16.6	15.5
	256	5.6	4.0	3.3	4.9	4.5
	4	279.0	177.0	150.0	216.0	200.0
.1	16	70.0	45.0	38.0	55.0	51.0
	64	17.8	11.3	9.6	13.8	12.8
	256	4.6	2.9	2.5	3.5	3.3
	4	279.0	176.0	149.0	216.0	200.0
.01	16	70.0	44.0	37.0	54.0	50.0
	64	17.6	11.1	9.4	13.5	12.5
	256	4.5	2.8	2.4	3.4	3.2

Table 10B. Execution Time (seconds)
Plane Stress Problem (Sum/Max)
(1 mult/add = 0.0001 sec)

Graphs 10A and 10B show the execution time (seconds) versus p for the 0,1,2-step PCG(SSOR)(Bus) algorithms for α=10 and α=1 respectively. For both graphs, m=2 solves the problem in the least time for all values of p. Note from Graph 10A that the execution time increases if 256 processors are used. This is because communication is expensive for α=10 and the cost of the bus for this many processors is prohibitive. Graph 10B shows that if communication is less expensive (α=1), that p=256 processors still give a further reduction in execution time for m=2 and m=1 but not for m=0. This clearly shows for a large number of processors that preconditioning is necessary to reduce the number of iterations in order to decrease the global communications even though



GRAPH 10A. α=10 CG(BUS)



GRAPH 10B. α=1 CG(BUS)

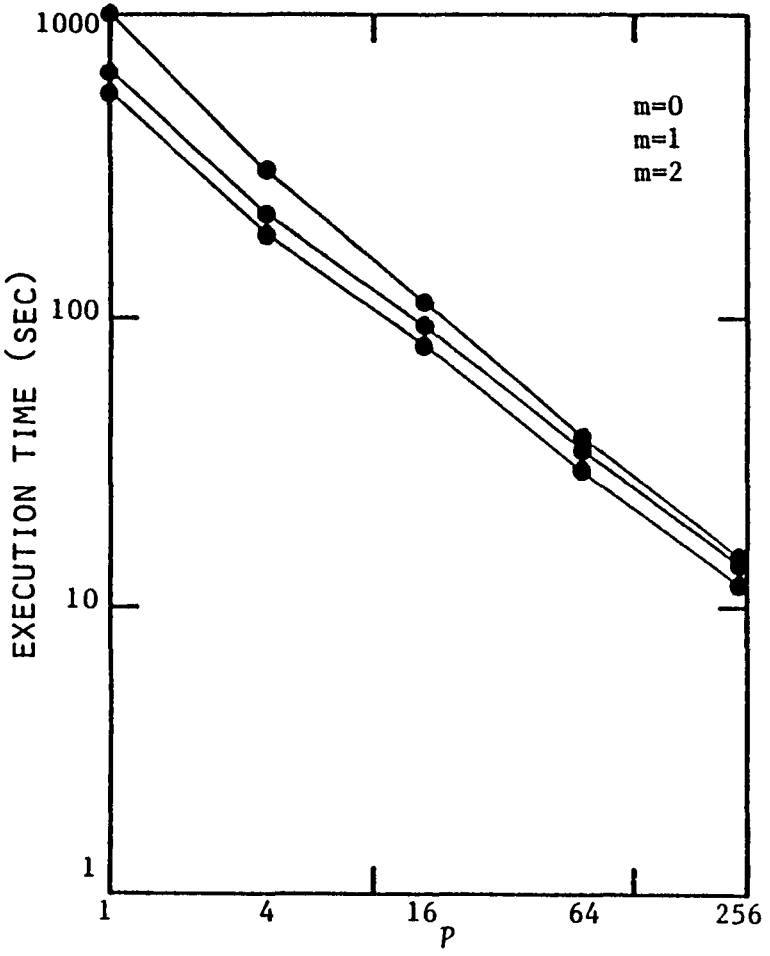
more local communications result.

The execution time for the same algorithms and the sum/max circuit for the inner products is plotted in Graphs 11A and 11B for α=10 and α=1 respectively. By comparing these graphs with Graphs 10A and 10B, the need for special hardware to do the inner products for the conjugate gradient methods is apparent whenever $p \geq 64$.

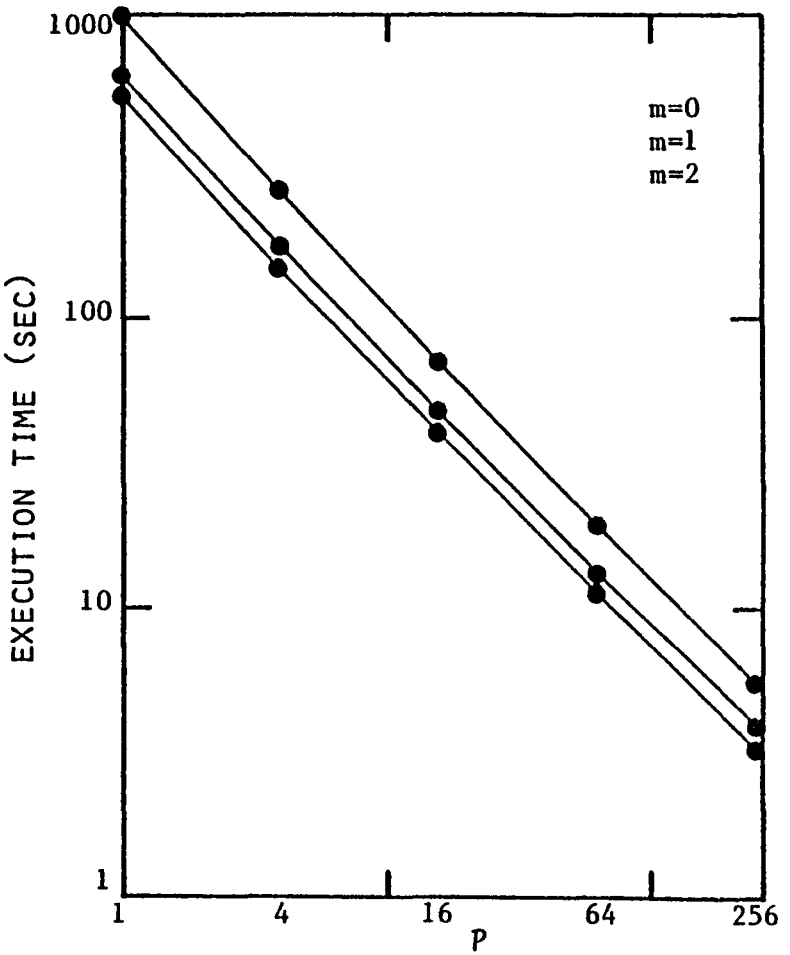
The time gained in the reduction in the number of iterations with $m > 0$ is greater than the extra time required for the preconditioner communications; hence, m=1,2 are faster than m=0 for both graphs. Note from Graph 11B that more of an improvement is seen with α=1 because of reduced communication. The convexity changes around p=16 in Graph 11B reflect the increase in communication cost of completely interior processors as was discussed for Graph 9B. Again, when α=1, this increase is not observed in Graph 11B.

Graph 12A shows the execution time (seconds) for R/B/G 2-step SSOR PCG(S/M) as a function of α with special cases shown for p=1(dotted), p=4, p=16, p=64, and p=256. For all α levels, p=256 yields the least execution time. However, note that it may be more cost effective to use 64 processors and make communication faster (α=1) rather than use α=10 and and increase p to 256.

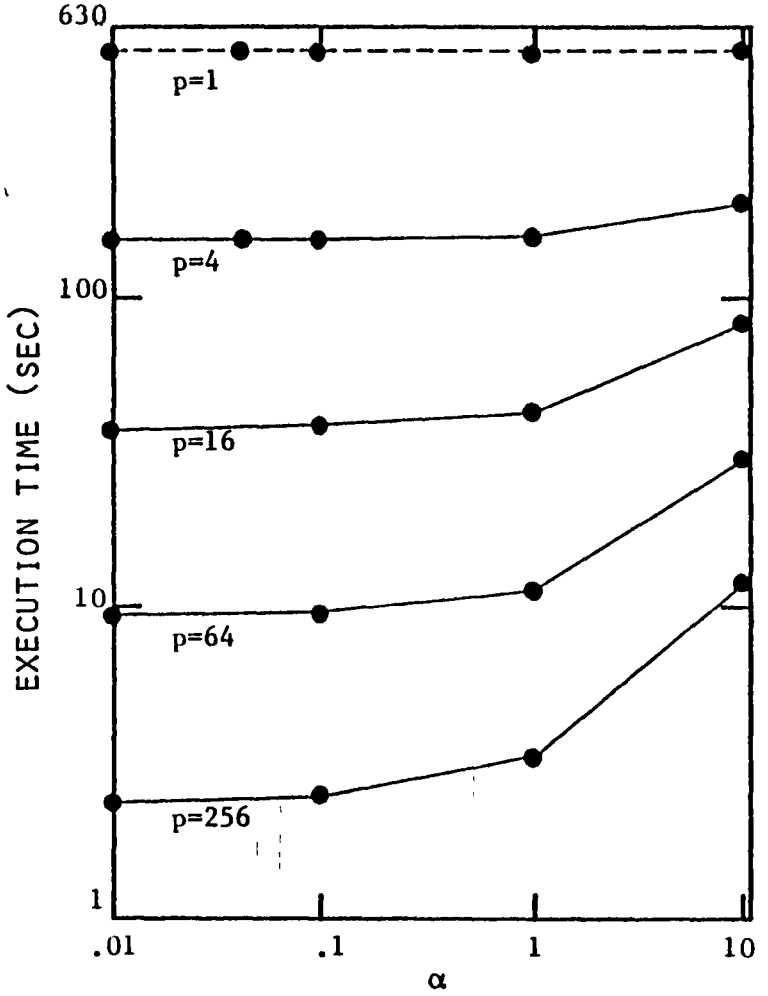
Finally, Graph 12B shows execution time (seconds) versus p for the R/B/G SSOR PCG(Sum/Max) algorithm for the special cases α=10, α=1, and α=0.1. This graph closely resembles Graph 9B for R/B SSOR PCG(Sum/Max) for Laplace's equation In particular, for $p \leq 64$ there is very little difference between the execution time for α=.1 and α=1, but there is a significant execution time difference between α=.1 and α=10.



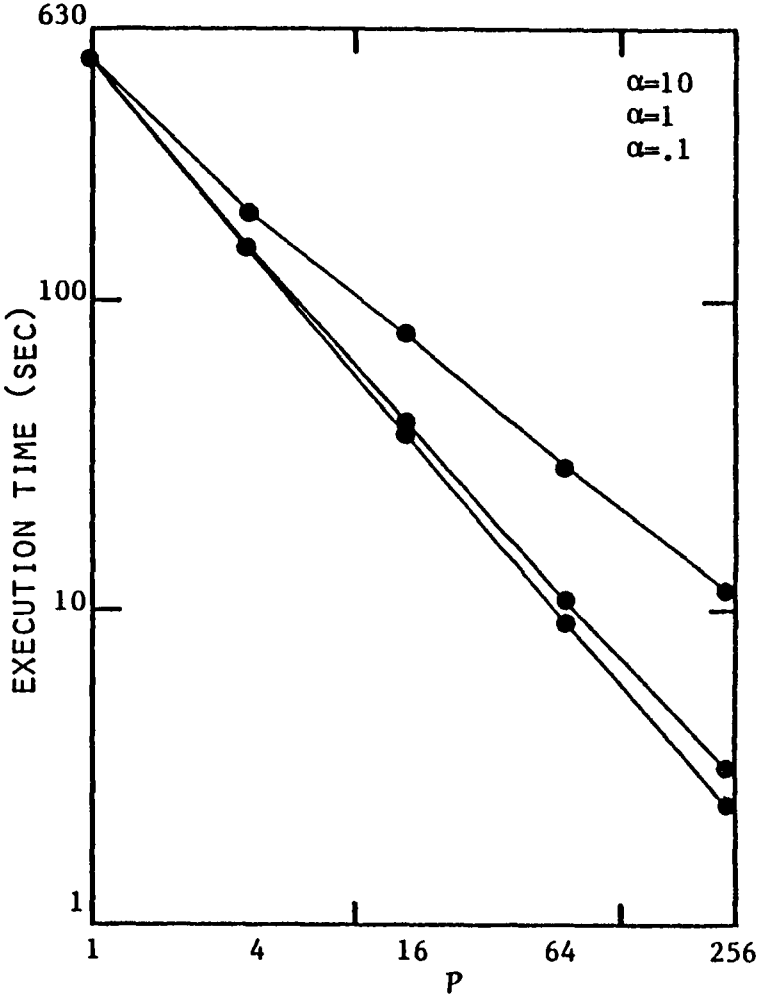
GRAPH 11A. α=10 (SUM/MAX)



GRAPH 11B. α=1 (SUM/MAX)



GRAPH 12A. R/B/G SSOR 2-PCG(SUM/MAX)



GRAPH 12B. R/B/G SSOR 2-PCG(SUM/MAX)

This suggests that there an optimal value of α, below which it is not cost effective to produce faster hardware.

7.3.4. Reliability Considerations

Speedup (p) is an indication of how much faster a problem can be solved on p processors than on a single processor. However, this measurement does not account for the fact that as more processors are added, the machine may be less reliable because of an increased probability of component failures.

We now define the reliability, $R(p,t)$, of the array computer to be the probability that all p processors will run at least t units of time without failure. To derive an expression for $R(p,t)$ the following assumptions must be made:

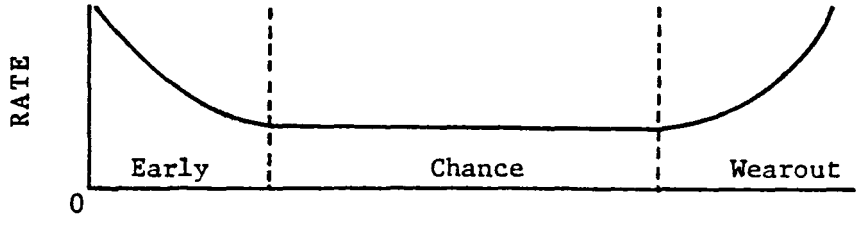
(1) The machine fails if any one of its processors fail.

(2) The failure of a processor is independent of the failures of the other processors.

(3) The failure rate, β, for each processor is constant.

(4) The reliability is a decreasing function with time.

We are interested only in the failures due to chance and not failures due to component "burn-in" or old age, as shown below (see Miller[1977]).



With these assumptions, the reliability of one processor is given by

$$R(1,t_1) = e^{-\beta t_1} \tag{7.4.1}$$

and the mean time until failure for one processor is

$$\mu_1 = \frac{1}{\beta} \tag{7.4.2}$$

Now, assumption (2) implies that the multiplicative law of probability holds for the p-processor case and that

$$R(p,t_p) = e^{-\beta p t_p} \tag{7.4.3}$$

where t_p denotes the time to solve the problem on a p-processor machine.

The mean time until failure for the p-processor machine is given by

$$\mu_p = \frac{u_1}{p} \tag{7.4.4}$$

If $Eff(p)$ represents the efficiency with p processors,

$$pt_p = \frac{pt_1}{speedup(p)} = \frac{t_1}{Eff(p)} \tag{7.4.5}$$

and by using (7.4.5), equation (7.4.3) can be written as

$$R(p,t_p) = e^{\frac{-\beta t_1}{eff(p)}} \tag{7 4.6}$$

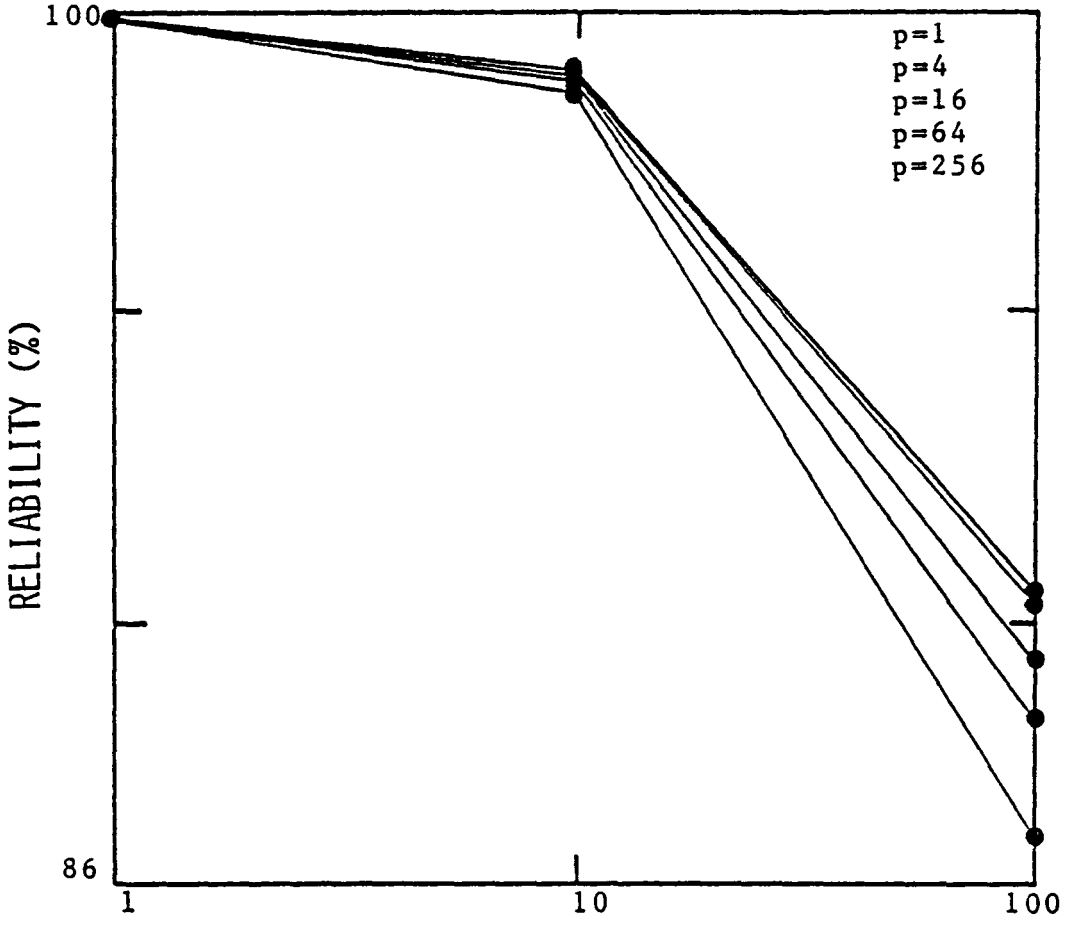
Note from (7.4.6) that the reliability is an increasing function of the efficiency. Hence, if we expect the efficiency to decrease with the number of processors, so will the reliability. However, if the machine parameter α and the algorithm were such that the efficiency remained nearly constant as the number of processors increased, the reliability would also be nearly constant for all p.

To illustrate these concepts, Table 11 shows how the reliability changes as the length of the solution time for the problem on a single processor increases. The speedups for the R/B/G SSOR PCG(Sum/Max) algorithm for the plane stress problem were used to calculate the efficiencies in (7.4.6). The failure rate of a single processor, β, was taken to be 0.001 (0.001 failures every 1 unit of time), where a unit of time can be specified to be any amount of time that precisely defines the failure rate.

Uniprocessor Job Length (time units)	p=1	p=4	p=16	p=64	p=256
1	99.90	99.87	99.78	99.68	99.49
10	99.00	98.72	97.86	96.85	94.98
100	90.48	87.89	80.56	72.61	59.74
		α=10, β=0.001			
1	99.90	99.90	99.89	99.88	99.86
10	99.00	98.98	98.89	98.79	98.58
100	90.48	90.25	89.41	88.50	86.69
		α=1 , β=0.001			
1	99.90	99.90	99.90	99.90	99.90
10	99.00	99.00	98.99	98.98	98.95
100	90.48	90.48	90.37	90.25	89.99
		α=.1, β=0.001			

Table 11. Reliabilities

Graph 13 shows the results from Table 11 for α=1. The graph indicates that the efficiency drops as the number of processors increases as was expected.



GRAPH 13. RELIABILITY

This factor should be kept in mind when analyzing the execution time graphs of section 7.3.3. Even though these graphs may show a decrease in execution time as more processors are added, the decrease may not be enough to balance the decrease in reliability. The level of reliability that is required is certainly a design issue for any machine.

7.3.5. Conventional Machines/Solvers

A natural question to ask is how well the execution time of an algorithm on an array computer such as the Finite Element Machine compares to the execution time of a conventional algorithm on a conventional machine. In other words, it may be of interest to determine the number of processors, p, and the communication to arithmetic ratio, α, that are needed for the algorithm on the array to be competitive with a benchmark algorithm implemented on a benchmark machine.

To answer this question we make several assumptions. First, we assume that a direct method such as banded Cholesky decomposition followed by forward and backward subsitiution will be the benchmark method. Secondly, assume that all the bands of the stiffness matrix K will fit into core storage of the benchmark computer We note that this is the best possible situtation for the benchmark algorithm/machine since for large problems time must be spent in bringing the matrix to and from core and backing store. Lastly, we assume three times, β, for performing one multiplication/addition pair on the benchmark computer; namely, 10^{-5},10^{-6}, and 10^{-7} μs.

The number of multiplication/addition pairs for the banded Chloesky algorithm is easily calculated, George and Liu[1981], to be

$$\frac{1}{2}(b)(b+3)N - (\frac{b^3}{3}+b^2+\frac{2}{3}b)$$

where b is the semi-bandwidth. The number of multiplication/addition pairs for both the forward and backward subititution is

$$(b+1)N - (\frac{b^2}{2}+\frac{b}{2})$$

where for the plane stress problem with an interior grid of 16 by 48 the bandwidth and number of equations are given by

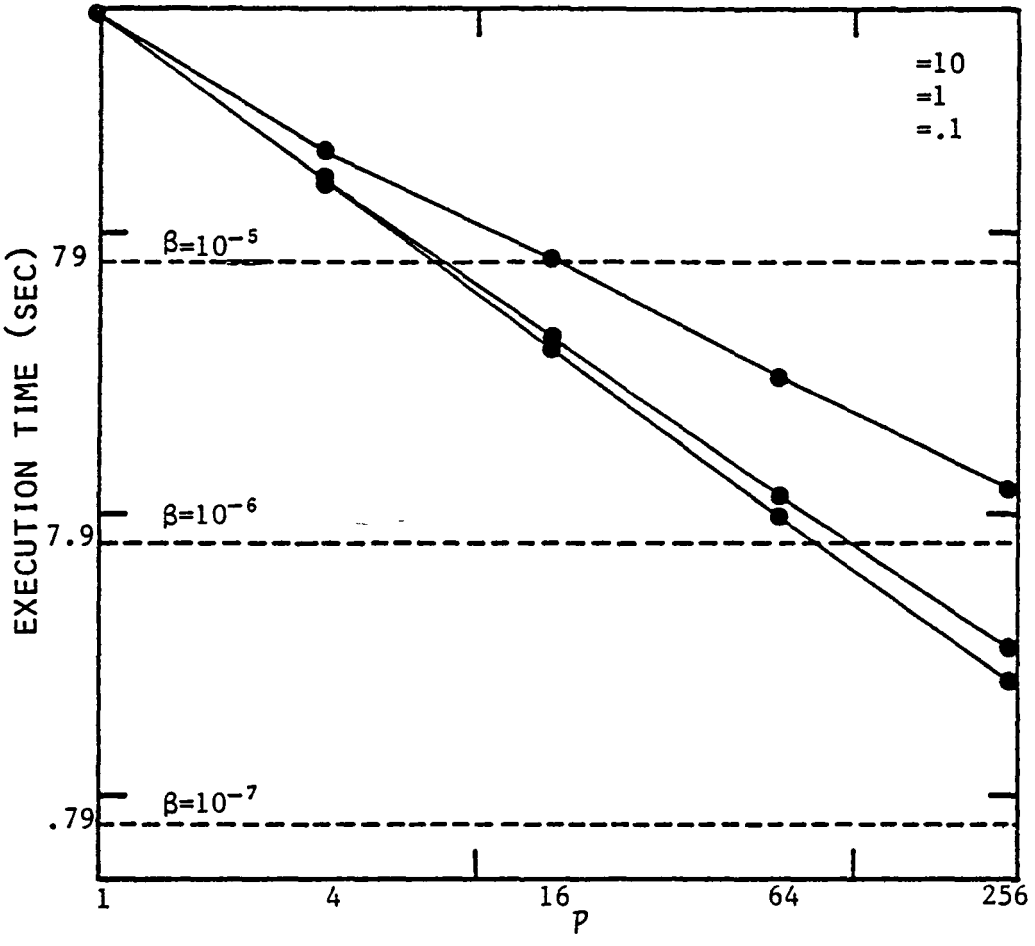
$$b = 96$$
$$N = 1536$$

Hence, the number of multiplication/addition pairs, a, is easily found to be

$$a = 7891936$$

For the three arithmetic speeds, $\beta=10^{-5}$, 10^{-6}, and 10^{-7}, the time in seconds for this algorithm is a linear function of β and equals 78.92, 7.89, and .79 for the three β values respectively.

Graph 14 shows the execution time as a function of the number of processors for α=0.1, 1, and 10 for the R/B/G SSOR 2-step PCG(Sum/Max) algorithm where the dotted horizontal lines represent the three execution times of the conventional solver on the conventional machine. The numbers of processors required to yield a smaller execution time than the Cholesky algorithm for the three values of β are summarized in Table 12.



GRAPH 14. COMPARISON WITH A CONVENTIONAL COMPUTER

α	p
.1	> 256
1.0	> 256
10.0	> 256

$\beta=10^{-7}$

.1	> 64
1.0	> 64
10.0	> 256

$\beta=10^{-6}$

.1	> 4
1.0	> 4
10.0	> 16

$\beta=10^{-5}$

Table 12. Comparison to a Conventional Solver/Machine

We acknowledge that the comparison to a conventional solver is difficult to make because the story may change completely as the problem size grows. For instance, direct methods will require more storage and the number of operations may no longer be competitive with iterative methods, especially if good initial guesses are known. For an array computer like FEM, the storage is distributed across the processors and iterative methods do not require storage of any nonzero elements of the stiffness matrix K; hence, extra time will not be as likely to be needed to move the data to and from backing store to core as would be true with direct methods. The point to be made here is that this type of analysis is simple once the benchmark algorithm/machine are determined and realistic times for this algorithm/machine are obtained.

CHAPTER 8

Conclusions and Future Directions

8.1. Conclusions

Two algorithms were developed in Chapter 4 for assembling the system of linear equations by the finite element method on array computers. The first algorithm required no communication between processors but resulted in a duplication of effort among the processors. The second algorithm required no duplication of effort at the expense of communication between processors. Analytic formulas were obtained for the speedup, efficiency, and overhead of these algorithms on a p-processor array. The more efficient algorithm was shown for $p>4$ processors to be a function of the ratio of the time to send and receive one value to the time to calculate one coefficient of the stiffness matrix. For $p=4$ processors, the choice of algorithms also depends on the size of the grid of unknowns that is assigned to each processor. We also described in Chapter 4 how to calculate the stress vector in parallel without communication between processors or duplication of effort.

In Chapter 5 we developed a new stationary iterative method, called Multi-color SOR, for solving the large sparse linear systems arising from both finite element and finite difference discretizations. This method is a generalization of the classical Red/Black ordering and allows the successive overrelaxation (SOR) method to be implemented on both vector computers and parallel arrays as a multiple sweep Jacobi method which has ideal properties for these machines.

The stiffness matrix K that results from a Multi-color ordering of the problem grid, was shown in general not be be consistently ordered, p-cyclic, or generally consistently ordered; therefore, the development of a theory for this class of matrix that will lead to the determination of the optimal relaxation factor ω is yet to be found. Numerical results show that the SOR method with the Multi-color ordering and the natural ordering of the grid converges in approximately the same number of iterations; therefore, the coloring of the grid for our test problems was not detrimental to the convergence rate of the method.

An efficient implementation of a Multi-color SSOR method that is based on a forward followed by a backward Multi-color SOR step was also given in Chapter 5. Numerical results for this method for a plane stress problem show that the optimal ω is close if not equal to 1. It is well known that the optimal ω for the Red/Black ordering of a matrix with Property A is 1, but it has yet to be proved whether or not this is true for general Multi-colored matrices.

Lastly, in Chapter 5, the Multi-color SOR method was generalized to the Block Multi-color SOR method. If the grid points in each block are from k consecutive rows (or columns) of the problem grid so that the matrix will be π-consistently ordered, (see Young[1971]), a theory exists for determining the optimal relaxation factor. On the other hand, if the grid points are blocked by lxk blocks of convenient size for implementation on a array of processors, it is generally not the case that the matrix will be π-consistently ordered.

In Chapter 6, we developed and analyzed an m-step preconditioned conjugate gradient method that can be efficiently implemented on both

vector computers and parallel arrays. This method takes m steps of a linear stationary iterative method derived from a symmetric and nonsingular splitting of the stiffness matrix K in order to precondition the system. In Theorem 1, we extend a result of Dubois, Greenbaum, and Rodrigue[1979] by giving the necessary and sufficient conditions for the resulting preconditioning matrix, M, to be symmetric and positive definite. In Theorem 2, we relate the positive definiteness of M to the convergence of the linear stationary iterative method and thereby generalize the Jacobi Convergence Theorem.

In Theorem 3, the condition number of the preconditioned system, $\kappa(\hat{K}_m)$, was proven to be a decreasing function of m if all the eigenvalues of G, the iteration matrix for the linear stationary iterative method, are positive. However, if the smallest eigenvalue of G is negative, the condition number behaves differently for m odd and m even. In particular, Theorem 4 shows if $\lambda_n > |\lambda_1|$, the condition number is decreasing for m odd, but is decreasing for m even if and only if the following inequality holds

$$(1+|\lambda_1|^{m+1})(1-\lambda_n^m) < (1-\delta^m)(1-\lambda_n^{m+1})$$

where $\delta = \max_i |\lambda_i|$. This means that increasing m from an odd to a consecutive even number of steps is more beneficial, in some cases, than increasing m from an even to a consecutive odd number of steps. These results further explain observations of Dubois, Greenbaum, and Rodrigue[1979].

The most promising linear stationary iterative method that we used for the m-step preconditioner was the Multi-color SSOR method. Numerical results show that the ratio of the number of iterations with the 2-

step Multi-color SSOR PCG method to the 1-step Multi-color SSOR PCG method was 1.40(1.36) for the plane stress problem and Laplace's equation respectively (the theoretical maximum is 2.0). To improve these results, we developed an m-step extrapolated PCG method that can be effective whenever all the eigenvalues of G are nonnegative (as is true for SSOR). Numerical results with this method show the ratios 1.40(1.36) are reduced to 1.93(1.76) respectively with little additional work. The disadvantage of using this method is that little theory exists for determining the extrapolation factor γ.

Finally, in Chapter 6, we compared our m-step extrapolated PCG method to the Parametrized Preconditioned Conjugate Gradient Method (PPCG) of Johnson, Micchelli, and Paul[1982] and showed the two are equivalent whenever $m=2$. For $m>2$, the PPCG method appears more general since the freedom of choosing more than one parameter can possibly lead to a better preconditioner. By using our more general symmetric and nonsingular splitting of the matrix K, we showed how to generalize the PPCG method. More research is required to determine the effectiveness of this approach.

In Chapter 7, we developed a model for comparing the execution time of parallel algorithms on an array computer. This model included the time for arithmetic, local convergence testing, synchronization for decision making, sending and receiving values over a global bus, and performing a summation of p numbers via the global bus or alternatively by a special hardware circuit. The hardware times for doing one of each of the above operations was varied to determine the dependence of an algorithm's performance on these parameters. The model was vali-

dated on a 4-processor Finite Element Machine at NASA Langley Research Center; however, as more processors are added to this machine, a more detailed validation can be done.

The model was used to predict speedups as well as execution times for our algorithms for Laplace's equation and the plane stress problem on a p-processor machine where α respresents the ratio of communication to arithmetic time. The major results are itemized below for Laplace's equation.

(1) R/B SOR was the most efficient and fastest algorithm for all values of p. The speedup for 384 processors was 307 for α=.01, 295 with α=.1, 211 with α=1 and as low as 55 with α=10.

(2) The conjugate gradient method with the global bus for the inner products should not be used with a large number of processors unless α<.01.

(3) Some improvement over the CG(Bus) algorithm is obtained by taking two steps of the Red/Black SSOR preconditioner; however, more improvement is gotten by using the sum/max hardware circuit for the inner products with this 2-step method but not enough to be competitive with the Red/Black SOR algorithm.

The major results are itemized below for the plane stress problem.

(1) The Red/Black/Green SOR algorithm, even though very efficient on an array, takes too many iterations to converge to be competitive with the conjugate gradient methods.

(2) The Red/Black/Green 2-step extrapolated SSOR preconditioned conjugate gradient algorithm with the sum/max circuit for the

inner products was the fastest method for this problem and quite efficient as well. The speedup values for 256 processors are 251 for α=.01, 242 for α=.1, 179 for α=1, and only 50 for α=10.

(3) The execution time for the method in (2) above varied very little when α was increased from .1 to 1 but varied significantly for $p \geq 16$ when α was increased from 1 to 10. Therefore, if a parameter study were done to determine the value of α for design purposes, extra model runs should be done between the range α=1 and α=10.

In Chapter 7, we showed that the reliability of a p-processor array decreases as the value of α increases. For example with p=256, a job length of 100 time units, and a component failure rate of 1 every time unit, the reliability decreases from 90% for α=.1 to 87% for α=1 to 60% for α=10. Finally, in Chapter 7, we outlined the procedure for comparing our algorithms with Cholesky decomposition followed by forward and backward substition on a conventional computer.

8.2. Future Directions

The efficient implementation of the Multi-color SOR and the m-step SSOR preconditioned conjugate gradient methods on an array of processors depends on the coloring of the nodes of the discretization followed by a particular mapping, or assignment, of the problem nodes to the processors. We gave in Chapter 5 the solution to this assignment problem for the special case of a rectangular problem domain. However, for irregular regions, the coloring of the nodes corresponds to a graph

coloring problem and in general is an NP-complete problem (see McDiarmid[1979], and More'[1981] for examples). Furthermore, Bokhari[1979] showed that assigning p nodes to p processors in order to reduce the communication time is also an NP-complete problem for a general problem domain. However, Bokhari did not consider the assignment of multiple nodes per processor. We note that the assignment of nodes to the processors is not independent of the solution algorithm used to solve the system of linear equations and for our algorithms must be viewed in conjunction with the coloring problem.

A second area for further research is the comparison of block and point iterative methods for parallel processors. Because of the overhead due to communication that was seen for our point methods, block methods may be competitive on these machines since the processor will become more computationally bound. These methods may prove effective for structural engineering problems since they are closely related to the modular or substructuring approach that is commonly used by structural engineers.

A third important area for the extension of our ideas is in the development of new iterative algorithms such as asynchronous methods and multi-grid techniques. The asynchronous methods of Baudet[1978] were implemented on a multi-processor system with central shared memory. For the distributed memory multiple instruction multiple data Finite Element Machine, research is needed to determine if these methods are competitive with synchronous methods. For example, the Multi-color SOR method can be implemented in an asynchronous fashion, thereby eliminating the wait time due to the synchronous receive and the

overhead due to some global flag checking, but it has yet to be determined whether the time saved by this overhead outweighs the possible increase in time if more iterations are necessary for convergence. We note that the current conjugate gradient methods can not run asynchronously since synchronization is necessary to accumulate the partial sums for the inner products.

An efficient multi-grid method for an architecture like the Finite Element Machine is another area for future research. This method requires relaxation on different sized problem grids and therefore may require a different communication strategy than the eight nearest neighbor connections. In addition, a parallel iterative method for performing the smoothing relaxations must also be developed.

REFERENCES

Advanced Micro Devices, Inc. [1979]. "Floating Point Processor." *Advanced MOS/LSI.*

Axelsson, O. [1976]. "A Class of Iterative Methods for Finite Element Equations," *Computer Methods in Applied Mechanics and Engineering*, Vol. 9, pp. 123-137.

Baudet, G. [1978]. "Asynchronous Iterative Methods for Multiprocessors," *Journal Association of Computing Machinery*, 25, pp. 226-224.

Barlow, R., Evans, D. [1982]. "Parallel Algorithms for the Iterative Solution to Linear Systems," *The Computer Journal*, Vol. 25, No. 1, pp. 56-60.

Becker, E., Carey, G., Oden, J. [1981]. Finite Elements: An Introduction, Volume I, Prentice Hall, Englewood Cliffs, N.J.

Bokhari, G. [1979]. "On the Mapping Problem," *Proceedings 1979 International Conference on Parallel Processing*, pp. 239-248.

Buzbee, B.L. [1978]. "Implementing Techniques for Elliptic Problems on Vector Processors," *LA-UR 80-2343*, Los Alamos Scientific Laboratory, Los Alamos, NM.

Buzbee, B.L., Boley, D., Parter, S.V. [1979]. "Applications of Block Relaxation," *1979 Society of Petroleum Engineers of AIME Fifth* Symposium on Reservoir Simulation," Denver, Co.

Chandra, R. [1978]. "Conjugate Gradient Methods for Partial Differential Equations," *Ph.D. thesis, Research Report #129*, Department of Computer Science, Yale University.

Coleman, T., More, J. [1981]. "Estimation of Sparse Jacobian Matrices and Graph Coloring Problems." *ANL-81-39*, Applied Mathematics Division, Argonne National Laboratory, Argonne, Ill.

Concus, P., Golub, G., O'Leary, D. [1976]. "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations." *STAN-CS-76-533*, Computer Science Department, Stanford University.

Conrad, V., Wallach, Y. [1977]. "Iterative Solution of Linear Equations on a Parallel Processor System." *IEEE Transactions on Computers*, Vol. C-26, No. 9, pp. 838-847.

Conrad, V., Wallach, Y. [1979]. "Alternating Methods for Sets of Linear Equations." *Numerische Mathematik*, Vol. 32, pp. 105-108.

Cryer, C.W., et.al. [1981]. "The Solution of Linear Complementary Problems on an Array Processor." *MRC Technical Summary Report #2170*, Mathematics Research Center, University of Wisconsin, Madison, WI.

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

Fix, G., Larsen, K. [1971]. "On the Convergence of SOR Iterations for Finite Element Approximations to Elliptic Boundary Value Problems." *SIAM Journal Numerical Analysis*, Vol. 8, No. 3, pp. 536-547.

Flynn, M.J. [1976]. "Very High-Speed Computing Systems." *Proceedings IEEE*, Vol. 54, pp. 1901-1909.

Fuller, S H , Harbison, S.P. [1978] "The C.mmp Multiprocessor." *CMU-CS-78-146*, Department of Computer Science, Carnegie Mellon University.

Forsythe, G. and Wasow, W. [1960]. Finite Difference Methods for Partial Differential Equations, John Wiley, New York.

Gannon, D. [1981]. "On Mapping Non-uniform P.D.E. Structures and Algorithms onto Uniform Array Architectures," *Proceedings 1981 International Conference on Parallel Processing*, pp. 100-105.

Gannon, D. [1980]. "A Note on Pipelining a Mesh Connected Multiprocessor for Finite Element Problems by Nested Dissection," *Proceedings 1980 International Conference on Parallel Processing*, pp. 197-204.

Gantmacher, F. [1959]. The Theory of Matrices, Vol. I., Chelsea Publishing Company, New York, pp. 296-298.

George, A., Poole, W.G. Jr., Voigt, R. [1976]. "Analysis of Dissection Algorithms for Vector Computers." *ICASE Report No. 76-17.*

George, A., Liu, J [1981]. Computer Solution of Large Sparse Positive Definite Systems, Prentice Hall, Inc., Englewood Cliffs, N.J.

Grosch, C. [1978]. "Poisson Solvers on a Large Array Computer," *TR 78-4*, Department of Mathematical and Computing Sciences, Old Dominion University, Norfolk, Va.

Grosch, C. [1979]. "Performance Analysis of Poisson Solvers on Array Computers,"TR 79-3, Department of Mathematical and Computing Sciences Old Dominion University, Norfolk, Va.

Hackbusch, W. [1977]. "On the Multi-Grid Method Applied to Difference Equations," *Computing*, Vol 20, pp. 291-306.

Hayashi, K., Yokoyama, M. [1977]. "Direct Simulation of Engineering Problems with a Fast Array Computer," *Bulletin of the Japan Society of Mechancial Engineers*, Vol. 20, No. 149, pp. 1438-1445.

Hayes, L. [1974]. "Comparative Analysis of Iterative Techniques for Solving Laplace's Equation on the Unit Square on a Parallel Processor," *M.S. Thesis*, Department of Mathematics, University of Texas, Austin.

Heller, D., Stevenson, D., Traub, J. [1976]. "Accelerated Iterative Methods for the Solution of Tridiagonal Systems on Parallel Computers," *Journal Association Computing Machinery*, Vol. 23, No. 4, pp. 636-654.

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

Hockney, R., Jessope [1982]. Parallel Computers, Adam Hilger Ltd., Techno House, Redcliffe Way, Bristol BS1, Great Britian.

Hotovy, S., Dickson, L. [1979]. "Evaluation of a Vectorizable 2-D Transonic Finite Difference Algorithm", *American Institute of Aeronautics and Astronautics, Inc.*, pp. 1-7.

Johnson, O., Micchelli, C., Paul, G. [1982] "Polynominal Preconditioners for Conjugate Gradient Calculations," *IBM Research Report #40444#*, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y.

Jones, A., Schwarz, R. [1980]. "Experience Using Multiprocessor Systems-A Status Report," *Computing Surveys*, Vol. 12, No. 2, pp. 121-165.

Jordan, H. [1978]. "A Special Purpose Architecture for Finite Element Analysis," *Proceedings 1978 International Conference on Parallel Processing*, pp. 263-266.

Jordan, H., Sawyer, P. [1978]. "A Multi-microprocessor System for Finite Element Structural Analysis," *Trends in Computerized Structural Analysis and Synthesis*, (A.Noor and H. McComb, Jr., Eds.), Pergamon Press, pp. 21-29.

Jordan, H. [1979]. "The Finite Element Machine-Programmer's Reference Manual," *CSDG-79-2*, University of Colorado, Boulder.

Jordan, H., Scalabrin, M., Calvert, W. [1979]. "A Comparison of Three Types of Multiprocessor Algorithms," *Proceedings 1979 International Conference on Parallel Processing*, pp. 231-238.

Kung, H., Leiserson C. [1978]. "Systolic Arrays for VLSI," *Sparse Matrix Proceedings*, Duff, I. and Stewart, G., editors.

Lambiotte, J. [1975]. "The Solution of Linear Systems of Equations on a Vector Computer," *Ph.D. Dissertation*, University of Virginia, Charlottesville, Va.

Manteuffel, T. [1979]. "An Incomplete Factorization Technique for Positive Definite Linear Systems," *Mathematics of Computation*, Vol. 34, No. 150, pp. 473-479.

McDiarmid, C. [1979]. "Determining the Chromatic Number of a Graph," *SIAM Journal Computing*, Vol. 8, No. 1, pp. 1-14.

Miller, I., Freund, J. [1977]. Probability and Statistics for Engineers, Prentice Hall, Inc., Englewood Cliffs, N J., pp. 450-453.

Nolen, J. [1979]. "Application of Vector Processors to the Solution of Finite Difference Equations." *1979 Society of Petroleum Engineers of AIME Fifth Symposium on Reservoir Simulation*, Denver, Co.

Noor, A., Fulton, R. [1974]. "Impact of the CDC-STAR 100 Computer on Finite Element Systems." *Sixth National ASCE Conference on Electronic Computation*, Atlanta, Ga.

Noor, A., Voigt, S. [1975]. "Hypermatrix Scheme for Finite Element Systems on the CDC-STAR Computer." *Computers and Structures*, Vol. 5, 1975, pp. 287-296.

Norrie, D., DeVries, G. [1978]. An Introduction to Finite Element Analysis, Academic Press, New York.

Oden, J., Becker, E., Carey, G. [1981]. Finite Elements: An Introduction, Volume I, Prentice Hall, Englewood Cliffs, N.J.

Ortega, J. [1972]. Numerical Analysis: A Second Course, Academic Press, New York.

Ortega, J., Voigt, R. [1977]. "Solutions of Partial Differential Equations on Vector Computers." *Proc. 1977 Army Numerical Analysis Conference*, pp. 475-526.

Ortega, J., Voigt, R. [1983]. "Solutions of Partial Differential Equations on Parallel Computers." (To appear in SIAM Review.)

Parter, S., Steuerwalt, M. [1980]. "On K-line and KxK Block Iterative Schemes for a Problem Arising in Three-Dimensional Elliptic Difference Equations." *SIAM Journal Numerical Analysis*, Vol. 17, No. 6, pp. 823-839.

Podsiadlo, D., Jordan, H [1981]. "Operating Systems Support for the Finite Element Machine." *CSDG-800-1*, University of Colorado, Boulder.

Reid, J. [1971]. "On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations," *Proceedings Conference on Large Sparse Sets of Linear Equations*, Academic Press, New York.

Reid, J. [1972]. "The Use of Conjugate Gradients for Systems of Linear Equations Possessing 'Property A'," *SIAM Journal Numerical Analysis*, Vol. 9, No. 2, pp. 325-332.

Reid, J. [1980]. "Solution of Linear Finite Element Equations," Invited chapter for special AMD volume on State of the Art Surveys of Finite Element Methods., (Eds. A. Noor and W. Pilkey).

Rieger, C., Trigg, R., Bane, B. [1981]. "ZMOB: A New Computing Engine for AI," *TR-1028*, Department of Computer Science, Maryland Artificial Intelligence Group, University of Maryland, College Park, Md.

Saad, Y., Sameh A. [1981]. "Iterative Methods for the Solution of Elliptic Difference Equations on Multiprocessors," Lecture Notes in Computer Science, CONPAR 81, Nürnberg, (Ed. Wolfgang Händler), Springer-Verlag, New York.

Sameh A., Kuck, D. [1978]. "On Stable Parallel Linear System Solvers", *Journal Association Computing Machinery*, Vol 25, No. 1, pp. 81-91

Schreiber, R. [1983]. "Implementation of the Conjugate Gradient Method on a Vector Computer."(To appear in *SIAM Journal on Scientific and Statistical Computation.*)

Schultchen, E., Kostem, C. [1973]. "Solution of Linear Equations by Iterative Methods in Finite Element Analysis," *Fritz Engineering Laboratory Report No. 400.10*, Department Civil Engineering, Lehigh University, Bethlehem, Penn.

Schwartz, J. [1979]. "Ultracomputers." *New York University Technical Report.*

Smith, B. [1978]. "A Pipelined Shared Resource MIMD Computer." *Proceedings 1978 International Conference on Parallel Processing*, pp. 6-8.

Smith, C., Loendorf, D. [1982]. "Performance Analysis of Software For An MIMD Computer." *CS-1982-7*, Duke University, Durham, N.C.

Storaasli, O., Peebles, S., Crockett, T., Knott, J., Adams, L. [1982]. "The Finite Element Machine: An Experiment in Parallel Processing." *NASA Technical Memorandum 84514*, NASA Langley Reseach Center, Hampton, Va.

Strang, G., Fix, G. [1973]. An Analysis of the Finite Element Method, Prentice-Hall, Englewood Cliffs, N.J.

Swan, R. [1977]. "CM*-A Modular Multi-Microprocesor." *AFIPS Conference Proceedings*, Vol. 46, AFIPS Press, Montvale, N.J., pp. 637-644.

van der Vorst, H. [1981]. "A Vectorizable Variant of Some ICCG Methods." *preprint*, Academisch Computer Centrum Utrecht, Budapestlaam 6, Utrecht, the Netherlands.

Varga, R. [1962]. Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, N.J.

Wellford, L., Vahdani, B. [1981]. "A Block Iteration Scheme for the Solution of Systems of Equations Resulting from Linear and Nonlinear Finite Element Models." *Computer Methods in Applied Mechanics and Engineering*, Vol. 26, pp. 33-52.

Whetstone, W. [1969]. "Computer Analysis of Large Linear Frames." *Journal of the Structural Division, Proceedings of ASCE*, pp. 2401-2417.

Young, D. [1971]. Iterative Solution of Large Linear Systems, Academic Press, New York.

Zave, P., Rheinboldt, W. [1979]. "Design of an Adaptive, Parallel Finite Element System." *ACM Transactions on Mathematical Software*, Vol. 5, No. 1, pp. 1-17.

Zienkiewicz, O. [1971]. The Finite Method in Engineering Science, McGraw-Hill, London, Great Britian.

Report No	2 Government Accession No.	3 Recipient's Catalog No
NASA CR-166027		

Title and Subtitle

Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers

5 Report Date: November 1982

6 Performing Organization Code:

Author(s)

Loyce M. Adams

8 Performing Organization Report No:

10 Work Unit No:

Performing Organization Name and Address

Department of Applied Mathematics & Computer Science
University of Virginia
Charlottesville, VA 22904

11 Contract or Grant No: NAG1-46

13 Type of Report and Period Covered: Contractor Report

Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Washington, DC 20546

14 Sponsoring Agency Code: 505-37-13-01

Supplementary Notes

Langley Technical Monitor: Dr. Olaf O. Storaasli

Abstract

Large sparse linear systems of equations require hours to solve on conventional mainframe computers; however, with the advent of parallel architectures, such as vector computers or arrays of microprocessors, these problems may be solved in less time. In addition, with hardware becoming cheaper, parallel algorithms for solving problems on these architectures may prove to be cost-effective.

In this thesis, we develop algorithms for assembling in parallel the sparse system of linear equations that result from finite difference or finite element discretizations of elliptic partial differential equations, such as those that arise in structural engineering. Parallel linear stationary iterative algorithms and parallel preconditioned conjugate gradient algorithms are developed for solving these systems. In addition, a model for comparing parallel algorithms on array architectures is developed and results of this model for our algorithms are given.

Key Words (Suggested by Author(s))	18 Distribution Statement
parallel processing, finite element methods, iterative algorithms, sparse linear systems, parallel computers	Unclassified - Unlimited Subject Category 64

Security Classif (of this report)	20 Security Classif (of this page)	21 No of Pages	22 Price
Unclassified	Unclassified	234	A11

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

End of Document