NUMERICAL ANALYSIS PROJECT
MANUSCRIPT NA-92-09

AUGUST 1992

An Implementation of a Generalized Lanczos Procedure for Structural Dynamic Analysis on Distributed Memory Computers

by

David R. Mackay
and
Kincho H. Law

NUMERICAL ANALYSIS PROJECT
COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305

An Implementation of a Generalized Lanczos Procedure for Structural Dynamic Analysis on Distributed Memory Computers[1]

David R. Mackay and Kincho H. Law
Department of Civil Engineering
Stanford University
Stanford, CA 94305-4020

Abstract

This paper describes a parallel implementation of a generalized Lanczos procedure for structural dynamic analysis on a distributed memory parallel computer. One major cost of the generalized Lanczos procedure is the factorization of the (shifted) stiffness matrix and the forward and backward solution of triangular systems. In this paper, we discuss load assignment of a sparse matrix and propose a strategy for inverting the principal block submatrix factors to facilitate the forward and backward solution of triangular systems. We also discuss the different strategies in the implementation of mass matrix-vector multiplication on parallel computer and how they are used in the Lanczos procedure. The Lanczos procedure implemented includes partial and external selective reorthogonalizations and spectral shifts. Experimental results are presented to illustrate the effectiveness of the parallel generalized Lanczos procedure. The issues of balancing the computations among the basic steps of the Lanczos procedure on distributed memory computers are discussed.

[1]This work is sponsored by the National Science Foundation grant number ECS-9003107, and the Army Research Office grant number DAAL-03-91-G-0038.

Contents

List of Figures

List of Tables

1 Introduction

Structural dynamic analysis often involves the solution of the generalized eigenvalue problem :

$$(K - \omega^2 M)\phi = 0 \tag{1}$$

or

$$(K - \lambda M)\phi = 0, \tag{2}$$

where K and M are, respectively, the stiffness matrix and the mass matrix of the structure. The coefficients λ, ω, and the vector ϕ are, respectively, the eigenvalue, the natural frequency and the natural modal eigenvector of the system. In structural dynamics, the smallest eigenvalues are of primary interest but they are often clustered and poorly separated. Efficient and robust solution methods that are effective in computing these eigenvalues accurately are of significant importance in structural engineering.

Recently, Lanczos method has been rapidly becoming the preferred method for the eigenvalue problems [2,4,7,16,18,23]. The recent emergence of parallel computers has brought much interest in the practical implementation of the Lanczos algorithm on these high performance computers. An implemention of a generalized Lanczos procedure on shared memory parallel computers has been reported by Jones and Patrick[8]. This paper describes an implementation of a generalized Lanczos algorithm on distributed memory parallel computers. The algorithm implemented has been influenced by the studies of Lanczos method by Golub, Underwood and Wilkinson[2], the convergence rate of Lanczos procedure by Kaniel, Paige, and Saad [9,17,20], the spectral transformation by Ericsson and Ruhe [1], and the partial and selective reorthogonalization techniques by Parlett , Scott, and Simon [18,22,23]. Our implementation follows closely the approach by Grimes et. al. [5].

One major cost of the generalized Lanczos procedure is the factorization of the (shifted) stiffness matrix and the forward and backward solution of triangular systems. In this paper, we discuss load assignment of a sparse matrix on distributed memory computers and propose a strategy for inverting the principal block submatrix factors to facilitate the forward and backward solution of triangular systems. We also discuss the different strategies in the implementation of mass matrix-vector multiplication on parallel computers and how they are used in the Lanczos procedure. The Lanczos procedure implemented includes partial and external selective reorthogonalizations. Spectral shifts are introduced when memory space is not sufficient for storing the Lanczos vectors. The tradeoffs between spectral shifts and Lanczos iterations are discussed.

This paper is organized as follows: First, in Section 2, we review the basic steps of the Lanczos method for generalized eigenvalue problems. In Section 3, we describe in detail the parallel implementation of the Lanczos algorithm on an Intel's i860 Hypercube computer. In Section 4, we present a few experimental results to illustrate the effectiveness of the parallel Lanczos method. Section 5 summarizes the results of this study.

2 Lanczos Method

2.1 The Standard Lanczos Algorithm

Given a symmetric n by n matrix, C, the Lanczos method tridiagonalizes the matrix by forming a set of orthonormal Lanczos vectors $q_1, \ldots, q_n$. The orthogonal matrix $Q = [q_1\ q_2\ \ldots\ q_n]$ has the following properties:

$$Q^T Q = I \tag{3}$$

$$Q^T C Q = T \tag{4}$$

where T is a tridiagonal matrix having the form:

$$T = \begin{vmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ & & & \beta_n & \alpha_n \end{vmatrix}. \tag{5}$$

Equation 4 may be rewritten as

$$CQ = QT$$

$$C[q_1\ \ldots\ q_j\ \ldots\ q_n] = [q_1\ \ldots\ q_j\ \ldots\ q_n]T \tag{6}$$

By equating each column of the above equation, we obtain a three term relationship among the Lanczos vectors:

$$Cq_j = \beta_j q_{j-1} + \alpha_j q_j + \beta_{j+1} q_{j+1} \tag{7}$$

where $\alpha_j = q_j^T C q_j$. Let's define the residual vector r_j as

$$r_j = \beta_{j+1} q_{j+1} = Cq_j - \beta_j q_{j-1} - \alpha_j q_j. \tag{8}$$

We have

$$q_{j+1} = \frac{1}{\beta_{j+1}} r_j \qquad \text{and} \qquad \beta_{j+1} = \|r_j\|_2 \tag{9}$$

The above equations form the basis of the Lanczos algorithm.

At step j of the Lanczos process we can write

$$CQ_j = Q_j T_j + r_j e_j^T \tag{10}$$

where e_j is a canonical vector with zero entries except a unit value in position j of the vector, $Q_j = [q_1\ \ldots\ q_j]$ and

$$T_j = \begin{vmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{j-1} & \alpha_{j-1} & \beta_j \\ & & & \beta_j & \alpha_j \end{vmatrix}. \tag{11}$$

Should the residual vector r_j become zero, then an invariant subspace is found.

2.2 The Lanczos Algorithm and Generalized Eigenproblem

Given the generalized eigenvalue problem, $K\# = \lambda M\phi$, there are various ways to transform the generalized eigenproblem into the standard form. Generally, a transformation based on the inverse of the stiffness matrix K improves convergence to the smallest eigenvalues of the original eigenproblem. In this study, the transformation is based on:

$$K^{-1}M\phi = \frac{1}{\lambda}\phi = \theta\phi. \tag{12}$$

where $\theta = \frac{1}{\lambda}$.

To symmetrize the eigenproblem shown in Equation 12, we multiply the equation by M = $L_M L_M^T$ and obtain

$$\begin{aligned} MK^{-1}M\phi &= \theta M\phi \\ L_M L_M^T K^{-1} L_M L_M^T \phi &= \theta L_M L_M^T \phi \\ L_M^T K^{-1} L_M y &= \theta y \\ cy &= \theta y \end{aligned} \tag{13}$$

where y = $L_M^T\phi$ and $C = L_M^T K^{-1} L_M$. This transformation restores symmetry to the problem, but it appears to require the factorization of the mass matrix M (which may be semi-positive definite). If we now consider the standard eigenvalue problem in Equation 13 and subtitute C = $L_M^T K^{-1} L_M$ into Equation 4, we have

$$Q^T L_M^T K^{-1} L_M Q = T .$$

Let $\hat{Q} = L_M^{-T} Q$ or Q = $L_M^T \hat{Q}$, we have

$$\hat{Q}^T M K^{-1} M \hat{Q} = T \qquad \text{and} \qquad \hat{Q}^T M \hat{Q} = I \tag{14}$$

By working with $\hat{Q}$ directly we can avoid factorizing M into $L_M L_M^T$ but we must now ensure that $\hat{Q}$ is M-orthonormal. Although we must form mass matrix-vector multiplications to retain orthogonality of Q, the matrix vector products are cheaper than an additional factorization, particularly since the mass matrix M may not be positive definite. Furthermore, as we shall show below, computing the eigenvectors ϕ using $\hat{Q}$ does not require the factor L_M.

For the standard eigenvalue problem, Cy = θ y , the eigenvector y can be written as a linear combination of the Lanczos vectors Q, that is

$$\mathrm{Y} = Qs \tag{15}$$

where s is an eigenvector of the tridiagonal matrix T. Thus, for the original generalized eigenvalue problem, an eigenvector can be expressed as:

$$\phi = L_M^{-T} y = L_M^{-T} Q s = \hat{Q} s, \tag{16}$$

That is, the eigenvectors remain unchanged for the transformed problem and the original problem.

Besides keeping $\hat{Q}$ M-orthonormal, we must transform the recurrence formula given by Equation 7 in terms of $\hat{Q}$ rather than Q. Substituting $q_j = L_M^T \hat{q}_j$ into Equation 8 gives

$$L_M^T \hat{q}_{j+1} = \frac{1}{\beta_{j+1}}(C L_M^T \hat{q}_j - \beta_j L_M^T \hat{q}_{j-1} - \alpha_j L_M^T \hat{q}_j) \tag{17}$$

$$\tag{18}$$

Since C = $L_M^T K^{-1} L_M$, we have:

$$L_M^T \hat{q}_{j+1} = \frac{1}{\beta_{j+1}}(L_M^T K^{-1} L_M L_M^T \hat{q}_j - \beta_j L_M^T \hat{q}_{j-1} - \alpha_j L_M^T \hat{q}_j) \tag{19}$$

$$= \frac{1}{\beta_{j+1}}(L_M^T K^{-1} M \hat{q}_j - \beta_j L_M^T \hat{q}_{j-1} - \alpha_j L_M^T \hat{q}_j) \tag{20}$$

Now multiplying the above equation by L_M^{-T} yields

$$\hat{q}_{j+1} = \frac{1}{\beta_{j+1}}(K^{-1} M \hat{q}_j - \beta_j \hat{q}_{j-1} - \alpha_j \hat{q}_j) \tag{21}$$

where $\alpha_j = \hat{q}_j^T M K^{-1} M \hat{q}_j$. It should be noted that β_{j+1} is now defined by the M-norm of the residual vector r, i.e. $\beta_{j+1} = \|r\|_M = \sqrt{r^T M r}$. For simplicity, we shall use **Q** and q instead of $\hat{Q}$ and $\hat{q}$ in the remainder of this paper.

To summarize, the Lanczos relations for the generalized eigenproblem can be stated as follows:

$$\begin{aligned} K^{-1} M \phi &= \frac{1}{\lambda}\phi = \theta\phi \\ Q^T M K^{-1} M Q &= T \\ Q^T M Q &= I \end{aligned} \tag{22}$$

The three term recurrence formula for the Lanczos vectors is:

$$r_j = \beta_{j+1} q_{j+1} = K^{-1} M q_j - \alpha_j q_j - \beta_j q_{j-1} \tag{23}$$

where

$$\alpha_j = q_j^T M K^{-1} M q_j \qquad \text{and} \qquad \beta_{j+1} = \|r\|_M = \sqrt{r^T M r} \tag{24}$$

The α's and β's form the diagonal and offdiagonal entries of the symmetric tridiagonal matrix T in Equation 5 or 11.

2.3 Spectral Transformation

To take full advantage of the Lanczos method it is beneficial to make the desired eigenvalues the extremal eigenvalues in the transformed problem. A spectral transformation using shifts has been

proposed by Ericsson and Ruhe [1]:

$$\begin{aligned} (K - \lambda M)\phi &= 0 \\ ((K - \sigma M) - (\lambda - \sigma)M)\phi &= 0 \\ (K_\sigma - \frac{1}{\vartheta}M)\phi &= 0 \\ \text{or} \qquad (K_\sigma^{-1}M - \vartheta I)\phi &= 0. \end{aligned} \tag{25}$$

where $K_\sigma = K - \sigma M$ and $\vartheta = \frac{1}{(\lambda - \sigma)}$. The shifting quantity σ is chosen to be close to the eigenvalues sought after, and ϑ is the eigenvalue of the shifted eigenproblem. The shift helps separate the eigenvalues further and increases their magnitude. Once the eigenvalues 6 of the transformed system are obtained, the eigenvalues λ of the original system are given as:

$$\lambda = \frac{1}{\vartheta} + \sigma. \tag{26}$$

For the generalized eigenvalue problem with shifts, we simply substitute K_σ for K in the relations shown in Equations 22, 23 and 24. In practice, the inverse K_σ^{-1} is not computed explicitly, rather the matrix K_σ is factorized into LDL^T. Therefore, the Lanczos procedure for the generalized eigenproblem is dominated mainly by matrix-vector multiplication and forward and backward solution of triangular systems.

2.4 Test for Convergence of the Generalized Eigenproblem

We shall now consider the convergence properties of the Lanczos algorithm for the spectrally transformed system. Let's denote $\hat{\vartheta}$ as the approximations to the eigenvalues ϑ of T, and $\hat{\lambda}$ as the approximations to the eigenvalues λ of the original problem. The difference between λ_i and $\hat{\lambda}_i$ can be expressed as

$$\begin{aligned} |\lambda_i - \hat{\lambda}_i| &= |\lambda_i - \sigma - \frac{1}{\hat{\vartheta}_i}| \\ &= |\frac{1}{\hat{\vartheta}_i}(\lambda_i - \sigma)(\frac{1}{\lambda_i - \sigma} - \hat{\vartheta}_i)| \\ &= |\frac{1}{\hat{\vartheta}_i}(\lambda_i - \sigma)(\vartheta_i - \hat{\vartheta}_i)|. \end{aligned} \tag{27}$$

Since the approximate eigenvalue $\hat{\vartheta}_i$ comes from projecting the eigenproblem onto a subspace, we have $|\hat{\vartheta}_i| \leq |\vartheta_i|$ or $|\hat{\vartheta}_i| \leq |\frac{1}{\lambda_i - \sigma}|$. Equation 27 becomes

$$|\lambda_i - \hat{\lambda}_i| \leq \frac{1}{\hat{\vartheta}_i^2}|\vartheta_i - \hat{\vartheta}_i| \tag{28}$$

Equation 28 can be used to estimate the bounds of the original eigenproblem by the bounds of the transformed problem.

Let (ϑ_i, s_i) be an eigenpair of T_j. The trial Ritz vector $\hat{\phi}_i$ can be computed as:

$$\hat{\phi}_i = Q_j s_i \tag{29}$$

After step j of the Lanczos algorithm, the accuracy of the approximate eigenvalue can be estimated by the residual as:

$$|\vartheta_i - \hat{\vartheta}_i| \leq \|K_\sigma^{-1} M \hat{\phi}_i - \vartheta_i \hat{\phi}_i\| \tag{30}$$

Substituting C = $K_\sigma^{-1} M$ into Equation 10 and multiplying it by s_i, the ith eigenvector of T_j, yields

$$K_\sigma^{-1} M Q_j s_i - Q_j T_j s_i = r_j e_j^T s_i \tag{31}$$

or

$$K_\sigma^{-1} M Q_j s_i - \vartheta_i Q_j s_i = r_j e_j^T s_i \tag{32}$$

Taking the norms on both sides of the above equation and substituting $\hat{\phi}_i = Q_j s_i$, we have

$$\|K_\sigma^{-1} M \hat{\phi}_i - \vartheta_i \hat{\phi}_i\| = \|r_j e_j^T s_i\| = \|r_j\| \, |e_j^T s_i| = \|r_j\| \, |s_{j,i}| = \beta_{j+1} |s_{j,i}| \tag{33}$$

where $s_{j,i}$ is the last (jth) element of the ith eigenvector of T_j. Combining Equations 28, 30 and 33, we have

$$|\lambda_i - \hat{\lambda}_i| \leq \frac{1}{\hat{\vartheta}_i^2} |\vartheta_i - \hat{\vartheta}_i| = \frac{1}{\hat{\vartheta}_i^2} \beta_{j+1} |s_{j,i}| \tag{34}$$

The eigenvalues near the shift σ will have a large value of ϑ_i which means that the value β_j does not need to become real small in order to obtain a good approximation of the eigenvalue A;.

There is another bound for testing the convergence for well separated eigenvalues proposed by Parlett [19]:

$$|\lambda_i - \hat{\lambda}_i| = \frac{1}{\hat{\vartheta}_i^2} \frac{(\beta_{j+1} s_{j,i})^2}{\gamma_i}. \tag{35}$$

where $\gamma_i = \min_{i \neq k} |\hat{\vartheta}_i - \hat{\vartheta}_k|, k = 1, \ldots,$ j. The definition of γ_i needs to be modified to account for double eigenvalues and for clusters of eigenvalues. In our implementation, the following procedure has been employed:

$k = i + 1$;
REPEAT
  $\gamma_R = |\hat{\vartheta}_i - \hat{\vartheta}_k|$;
  $k = k + 1$;
UNTIL $\gamma_R >$ *tol* $* \hat{\vartheta}_i$, OR $k > j$.
$k = i - 1$;
REPEAT
  $\gamma_L = |\hat{\vartheta}_i - \hat{\vartheta}_k|$;
  $k = k - 1$;
UNTIL $\gamma_L >$ *tol* $* \hat{\vartheta}_i$, OR $k < 1$.
$\gamma_i = \min\{\gamma_R, \gamma_L\}$.

In our implementation on the Intel's i860 hypercube computer, the parameter *tol* is taken to be 1.6 x 10^{-13} which is an estimate based on machine precision. This parameter seems to work well in the structural engineering examples that we have tested.

The test against the smaller of the two bounds in Equations 34 and 35 becomes

$$|\lambda_i - \hat{\lambda}_i| \leq \min\{\frac{\beta_{j+1}|s_{j,i}|}{\hat{\vartheta}_i^2}, \frac{(\beta_{j+1}s_{j,i})^2}{\hat{\vartheta}_i^2\gamma_i}\} \tag{36}$$

Equation 36 is the basis for the test of convergence in our implementation of the Lanczos algorithm for the generalized eigenvalue problem.

2.5 Reorthogonalization of the Generalized Eigenproblem

So far the discussion on the Lanczos algorithm has been based entirely on the assumption of exact arithmetic. An implementation of the Lanczos method, however, must rely on finite arithmetic and the associated round-off errors and cancellations. It is well known that the Lanczos vectors will not retain their orthogonality as the number of Lanczos vectors increases.

Full reorthogonalization requires that each Lanczos vector q_{j+1} orthogonalizes against all previous Lanczos vectors $\{ q_j \ldots q_1\}$ as it is formed such that $q_i^T M q_j = \delta_{i,j} + \epsilon n$ where $\delta_{i,j} = 1$ for i = j and 0 otherwise [2]. The process is expensive in terms of both memory space and CPU time, especially as the number of iterations and the number of Lanczos vectors increase. Alternatives to full reorthogonalization are partial reorthogonalization and selective reorthogonalization.

Both selective and partial reorthogonalizations seek to maintain semi-orthogonality rather than full orthogonality. In this study, we define semi-orthogonal@ as maintaining the orthogonality of the Lanczos vectors to the extent that $q_i^T M q_j = \delta_{i,j} + \sqrt{\epsilon n}$ [22]. In the following two subsections., we discuss in further detail the methods for partial and selective reorthogonalizations.

2.5.1 Partial Reorthogonalization

Partial reorthogonalization requires that each new Lanczos vector orthogonalizes with all previous Lanczos vectors when the bounds measuring the semi-orthogonality exceed an acceptable level. The bounds used to determine the loss of orthogonality can be derived from considering Equation 23 in floating point arithmetic:

$$\beta_{j+1} q_{j+1} = K_\sigma^{-1} M q_j - \alpha_j q_j - \beta_j q_{j-1} + f_j \tag{37}$$

where f_j represents any roundoff error occurring during the formation of q_{j+1} . Multiplying the above equation by $q_k^T M$ we obtain,

$$\beta_{j+1} q_k^T M q_{j+1} = q_k^T M K_\sigma^{-1} M q_j - \alpha_j q_k^T M q_j - \beta_j q_k^T M q_{j-1} + q_k^T M f_j. \tag{38}$$

Define $\eta_{i,k} = q_k^T M q_i = q_i^T M q_k = \eta_{k,i}$ and Equation 38 becomes,

$$\eta_{j+1,k}\beta_{j+1} = q_k^T M K_\sigma^{-1} M q_j - \alpha_j \eta_{j,k} - \beta_j \eta_{j-1,k} + q_k^T M f_j . \tag{39}$$

Similarly for the kth step of Lanczos process, we have

$$\beta_{k+1} q_{k+1} = K_\sigma^{-1} M q_k - \alpha_k q_k - \beta_k q_{k-1} + fk \tag{40}$$

Multiplying this equation by $q_j^T M$, we have

$$\eta_{k+1,j}\beta_{k+1} = q_j^T M K_\sigma^{-1} M q_k - \alpha_k \eta_{k,j} - \beta_k \eta_{k-1,j} + q_j^T M f_k. \quad (41)$$

Since $MK_\sigma^{-1}M$ is a symmetric matrix and $q_j^T M K_\sigma^{-1} M q_k = q_k^T M K_\sigma^{-1} M q_j$, from Equations 39 and 41, we obtain

$$\eta_{j+1,k} = \frac{1}{\beta_{j+1}}(\eta_{j,k+1}\beta_{k+1} + (\alpha_k - \alpha_j)\eta_{j,k} + \beta_k \eta_{j,k-1} - \beta_j \eta_{j-1,k} + g_{k,j}) \quad (42)$$

where $g_{k,j} = q_k^T M f_j - q_j^T M f_k$. It should be noted that, from the M-orthonormality of the Lanczos vectors, $\eta_{j+1,j+1} = 1$.

Equation 42, however, does not provide a value for $\eta_{j+1,j}$, which is considered to represent the local roundoff or cancellation error between the two vectors q_j and q_{j+1}. It is assumed this local round off will be bounded by $\epsilon\sqrt{n}$. So, we initialize $\eta_{j+1,j}$ as follows:

$$\eta_{j+1,j} = \max(\frac{\epsilon\sqrt{n}}{\beta_{j+1}}, \epsilon\sqrt{n}). \quad (43)$$

Since this value is assumed to account for every occurence of f_j or $g_{k,j}$, we may ignore $g_{k,j}$ in the subsequent calculations of η. When $g_{k,j}$ (k < j) is dropped from Equation 42, we have

$$\eta_{j+1,k} = \frac{1}{\beta_{j+1}}(\eta_{j,k+1}\beta_{k+1} + (\alpha_k - \alpha_j)\eta_{j,k} + \beta_k \eta_{j,k-1} - \beta_j \eta_{j-1,k}) \quad (44)$$

In the case of $k = j - 1$, and using the fact that $\eta_{i,i} = 1$, we can now write

$$\eta_{j+1,j-1} = \frac{1}{\beta_{j+1}}((\alpha_{j-1} - \alpha_j)\eta_{j,j-1} + \beta_{j-1}\eta_{j,j-2}) \quad (45)$$

Both conditions are used to track the loss of semi-orthogonality [4]. When $\eta_{max} = \max\{\eta_{j,k} | k < j\} \geq \sqrt{\epsilon n}$, both the Lanczos vectors q_j and q_{j+1} must be reorthogonalized against all previous Lanczos vectors:

$$q_i = q_i - (q_i^T M q_k) q_k \quad (46)$$

where i = j, j + 1 and k = 1,. . . i. Because of the three term recurrence relationship of the Lanczos vectors, if only q_{j+1} were orthogonalized but q_j was not reorthogonalized, the next Lanczos vector q_{j+2} would pick up poor orthogonality from q_j. This component would quickly propagate in the subsequent calculations. Thus both q_j and q_{j+1} need to be reorthogonalized.

2.5.2 Selective Reorthogonalization

Selective reorthogonalization attempts to reduce the number of vectors with respect to which the new Lanczos vector must be orthogonalized. Whenever an eigenvalue converges during the Lanczos process, the method is interrupted to form the corresponding approximate eigenvector or Ritz vector as a linear combination of the existing Lanczos vectors. This forms an approximate Ritz pair $(\theta_i, \hat{\phi}_i)$ to the eigenpair (λ_i, ϕ_i). Now, instead of measuring the loss of orthogonality with respect to previous

Lanczos vectors, the loss of orthogonality is measured in terms of the Ritz vectors. When the loss of orthogonality becomes too large, the Lanczos vector q_{j+1} is reorthogonalized not to the previous Lanczos but only to the Ritz vectors for which the bound on orthogonality has grown too large.

In this implementation, we follow closely the approach by Grimes et. al. [5]. In this approach, for each new shift, selective reorthogonalization is used to maintain semi-orthogonality of the new Lanczos vectors to the Ritz vectors that converged from a previous set of Lanczos vectors with a different σ. When a new shift σ is selected, the initial starting Lanczos vector is chosen that is orthogonal to all previously converged Ritz vectors. This orthogonality would be maintained in the Lanczos iterations if the Ritz vectors were exact eigenvectors and exact arithmetic were used. However, since we always deal with floating point arithmetic we need to trace the loss of orthogonality to the converged Ritz vectors. When the loss of orthogonality occurs, the Lanczos vector is reorthogonalized with the previously converged Ritz vectors. The converged Ritz vectors to which we want to maintain orthogonality are independent from this iteration of Lanczos. Therefore this method is appropriately called external selective reorthogonalization [5]. This external selective reorthogonalization is important in eliminating the possibility of computing the same Ritz (eigen) vector twice.

For the-external selective reorthogonalization of the Lanczos vectors with respect to the Ritz (eigen) vectors of different shift values, let's begin by multiplying Equation 37 by $\hat{\phi}_i^T M$ and form:

$$\beta_{j+1}\hat{\phi}_i^T M q_{j+1} = \hat{\phi}_i^T M K_\sigma^{-1} M q_j - \alpha_j \hat{\phi}_i^T M q_j - \beta_j \hat{\phi}_i^T M q_{j-1} + \hat{\phi}_i^T M f_j. \tag{47}$$

For the spectrally transformed problem, define the residual vector u_i of the Ritz vector $\hat{\phi}_i$ as

$$u_i = K_\sigma^{-1} M \hat{\phi}_i - \frac{1}{\vartheta_i - \sigma}\hat{\phi}_i. \tag{48}$$

Now from Equation 48 we can form

$$M K_\sigma^{-1} M \hat{\phi}_i = \frac{1}{\vartheta_i - \sigma} M \hat{\phi}_i + M u_i \tag{49}$$

If we define $\tau_{i,j} = \hat{\phi}_i^T M q_j$ and substitute Equation 49 into Equation 47, we obtain

$$\beta_{j+1}\tau_{i,j+1} = \frac{1}{\vartheta_i - \sigma}\tau_{i,j} - \alpha_j \tau_{i,j} - \beta_j \tau_{i,j-1} + \hat{\phi}_i^T M f_j + u_i^T M q_j \tag{50}$$

Taking the absolute value or bounds on the above equation, we have:

$$\begin{aligned} |\beta_{j+1}\tau_{i,j+1}| &= |\frac{1}{\vartheta_i - \sigma}\tau_{i,j} - \alpha_j \tau_{i,j} - \beta_j \tau_{i,j-1} + \hat{\phi}_i^T M f_j + u_i^T M q_j| \\ &\leq |\frac{1}{\vartheta_i - \sigma}\tau_{i,j} - \alpha_j \tau_{i,j} - \beta_j \tau_{i,j-1} + \hat{\phi}_i^T M f_j| + |u_i^T M q_j|. \end{aligned} \tag{51}$$

Let's examine the bounds on the term $u_i^T M q_j$.

$$\begin{aligned} \|u_i^T M q_j\| &= \|u_i^T M^{1/2} M^{1/2} q_j\| \\ &\leq \|u_i^T M^{1/2}\| \|M^{1/2} q_j\| \\ &\leq \|u_i\|_M \|q_j\|_M \\ &\leq \|u_i\|_M \end{aligned} \tag{52}$$

Let $\tau_{i,1} = \epsilon\sqrt{n}$ and let this value account for the term $\hat{\phi}_i^T M f_j$ in Equation 50. If we substitute $\|u_i\|_M$ for $u_i^T M q_j$, $\tau_{i,j+1}$ becomes an upper bound on the loss of orthogonality and is given by

$$\tau_{i,j+1} \leq \frac{1}{\beta_{j+1}}[|(\frac{1}{\vartheta_i - \sigma} - \alpha_j)\tau_{i,j} - \beta_j \tau_{i,j-1}| + \|u_i\|_M]. \tag{53}$$

Equation 53 is used to test for loss of orthogonality of the Lanczos vectors with respect to external Ritz vectors. It should be noted that $\|u_i\|_M$ of a converged Ritz vector is computed only once per each spectral shift. For a more formal discussion on the use of Equation 53 and an alternative formulation of $\|u\|_M$, see Reference [5].

For selective reorthogonalization, when the parameter $\tau_{i,j+1} \geq \sqrt{\epsilon n}$, the Lanczos vector q_{j+1} is reorthogonalized with the converged Ritz vector $\hat{\phi}_i$ as

$$q_{j+1} = q_{j+1} - (q_{j+1}^T M \hat{\phi}_i)\hat{\phi}_i \tag{54}$$

Furthermore, the next Lanczos vector q_{j+2} needs also be reorthogonalized to the same Ritz vector $\hat{\phi}_i$ because of the three term recurrence relationship of Lanczos vectors. In our implementation, a combination of both partial and external selective reorthogonalization is used.

2.6 Ritz Vector Refinement

The Ritz vectors or approximate eigenvectors are generally not as accurate as the approximations to the eigenvalues. Therefore it is necessary to refine the Ritz vector beyond $\hat{\phi}_i = Q s_i$. This refinement can be performed using one step of inverse iteration:

$$\hat{\phi}_i^{(new)} = \frac{1}{\vartheta_i} K_\sigma^{-1} M \hat{\phi}_i. \tag{55}$$

Since

$$K_\sigma^{-1} M \hat{\phi}_i = \vartheta_i \hat{\phi}_i + \beta_{j+1} q_{j+1} e_j^T s_i \tag{56}$$

we have

$$\hat{\phi}_i^{(new)} = \hat{\phi}_i + (\frac{1}{\vartheta_i} \beta_{j+1} s_{j,i}) * q_{j+1}. \tag{57}$$

This refinement was suggested by Ericsson and kuhe [1]. The values β_{j+1} and q_{j+1} are readily available since they are calculated in the same iteration as α_j is calculated. So the refinement accounts to one axpy operation for each Ritz vector.

2.7 Summary of Generalized Lanczos Algorithm

The generalized Lanczos algorithm described in this section is summarized as shown in Figure 1. There are five basic operations involved in the Lanczos procedure:

- Matrix factorization: The (shifted) stiffness matrix, K or K_σ, is factorized into its LDL^T factors.

```
Procedure: Generalized Lanczos Procedure
BEGIN
   WHILE (# of converged eigenvalues < # of desired eigenvalues) DO
   BEGIN
      Choose a shift σ ;
      Form K_σ = K − σM ;
      Factor K_σ = LDL^T ;
      set β_0 = 0 ;
      set q_0 = 0 ;
      /* Compute residual of Ritz vectors after spectral shift */
      FOR i = 1 to number of converged Ritz vectors, DO
         p = M φ̂_i ;
         p = K_σ^{-1} p ;
         p = p − (1/(λ_i − σ)) * φ̂_i ;
         ||u_i|| = sqrt(p^T M p) ;
      END.
      Choose r ;
      /* orthogonalize r to all converged Ritz vectors */
      FOR i = 1 to number of converged Ritz vectors, DO
         r_{i,j} = -1. ;
      END.
      Selective_Reorthogonalization ;
      p = Mr ;
      β_1 = sqrt(r^T p) ;
      q_1 = r/β_1 ;
      P = p/β_1 ;
      η_{1,1} = 1 ;
      /* Lanczos Loop */
      FOR j = 1 UNTIL insufficient space for Lanczos vectors q_j, DO
      BEGIN
         r = (K_σ)^{-1} p;
         r = r − β_{j-1} * q_{j-1} ;
         α_j = r^T p ;
         r = r − α_j * q_j ;
         p = Mr ;
         β_{j+1} = sqrt(r^T p) ;
         Reorthogonalize;
         q_{j+1} = r/β_{j+1} ;
         P = p/β_{j+1} ;
         Solve eigensystem T_j for (ϑ_i, s_{j,i}), i = 1,...,j;
         Test for convergence of eigenvalues ;
      END.
   END.
   Ritz_Refinement;
END.
```

(a) Lanczos Procedure

Figure 1: Generalized Lanczos Procedure

```
Procedure: Reorthogonalization
/* ε_s = ε√n where ε denotes machine precision */
BEGIN
   Selective_Reorthogonalization ;
   Partial_Reorthogonalization ;
END


Procedure: Selective_Reorthogonalization
BEGIN
   /* Selective Reorthogonalization - perform as needed
   FOR i = 1 TO number of converged eigenpairs (λ, φ̂), DO
   BEGIN
      IF τ_{i,j} ≥ 0 THEN τ_{i,j+1} = (|(λ_i - α_j)τ_{i,j} - β_j τ_{i,j-1}| + ||u_i||)/β_{j+1};
      IF τ_{i,j+1} ≥ √εn OR τ_{i,j} < 0 THEN
         q_{j+1} = q_{j+1} - (q_{j+1}^T M φ̂_i) φ̂_i ;
         IF τ_{i,j} < 0 THEN
            τ_{i,j} = 0 ;
            τ_{i,j+1} = ε_s ;
         ELSE
            /* set negative as a flag for next Lanczos */
            τ_{i,j+1} = -1 ;
         ENDIF.
      ENDIF.
   END.
END


Procedure: Partial_Reorthogonalization
BEGIN
   /* Partial Reorthogonalization - perform as needed */
   η_{j+1,j+1} = 1 ;
   η_{j+1,j} = max{ε_s, ε_s/β_{j+1}} ;
   η_{j+1,j-1} = ((α_{j-1} - α_j)η_{j,j-1} + β_{j-1}η_{j,j-2})/β_{j+1} ;
   FOR k = 1 TO j - 2, DO
   BEGIN
      η_{j+1,k} = (β_{k+1}η_{j,k+1} + β_k η_{j,k-1} - β_j η_{j-1,k} + (α_k - α_j)η_{j,k})/β_{j+1} ;
   END.
   η_max = max_{k≤j}(η_{j+1,k})
   IF η_max ≥ √εn THEN
      FOR k = 1 TO j - 1, DO
      BEGIN
         q_j = q_j - (q_j^T M q_k) · q_k ;
         q_{j+1} = q_{j+1} - (q_{j+1}^T M q_k) · q_k ;
      END.
      ρ = q_{j+1}^T M q_{j+1} ;
      REPEAT
         q_{j+1} = q_{j+1} - (q_{j+1}^T M q_j) · q_j ;
         ρ_old = ρ ;
         ρ = q_{j+1}^T M q_{j+1} ;
      UNTIL (ρ/ρ_old ≥ √2/2) ;
      β_{j+1} = √ρ ;
   ENDIF.
END.
```

(b) Reorthogonalization

Figure **1** (continued)

```
Procedure: Ritz_Refinement
BEGIN
   /*Form eigenvectors of converged eigenvalues ● /
   /* Use TQL2 to obtain eigenpairs of tridiagonal matrix*/
   Solve eigensystem T_j for (λ_i, s_i), i = 1,...,j;
   l = 0 ;
   FOR i = 1 TO j, DO
   BEGIN
      IF (eigenvalue i converged) THEN
         l = l + 1;/* counter for converged eigenvalues */
         φ̂_l = 0 ;
         FOR k = 1 TO j, DO
         BEGIN
            φ̂_l = φ̂_l + s_{k,i} * q_k
         END.
         φ̂_l = φ̂_l + (β_{j+1} s_{j,i}/ϑ_i) * q_{j+1};/*refinement ● /
      ENDIF.
   END.
   /* orthogonal&e eigenvectors for each converged eigenvalue*/
   FOR i = 1 TO l, DO
   BEGIN
      p = M φ̂_i ;
      ρ = φ̂_i^T p ;
      /* normalize Ritz vector and scale mass matrix vector product ● /
      || = φ̂_i/√ρ |
      p = p/√ρ ;
      /*orthogonalize following Ritz vectors */
      FOR k = i + 1 to l, DO
      BEGIN
         ρ = p^T φ̂_k
         φ̂_k = φ̂_k − ρ * φ̂_i ;
      ENDFOR.
   ENDFOR.
END.
```

(c) Ritz-Vector Refinement

Figure 1 (continued)

- Solution of triangular systems: The forward and backward solves of triangular systems of equations are performed for calculating the residual vector, $r = K_\sigma^{-1} p$, at each Lanczos iteration.
- Matrix vector products: Mass matrix-vector multiplication is performed at each Lanczos iteration as well as when the Lanczos vectors are reorthogonalized.
- Vector-vector products: These operations are of two types: dot products, $u^T v$, and axpy operations, u = u + ρv. These two vector operations are the basic BLAS routines which are often available as library routines.
- Eigensolution of tridiagonal systems: The eigenvalues of the tridiagonal matrix T_j are solved at each Lanczos iteration. However, the size of the tridiagonal matrix is often small in structural dynamics problems. Standard sequential routines such as **TQL2** of **EISPACK** are available for computing the eigenvalues [24].

In the next section, we describe in detail the implementation of these operations on an Intel's i860 hypercube, a distributed memory parallel computer.

3 Parallel Implementation

In this section, we examine the parallel implementation of the generalized Lanczos procedure. First, we discuss sparse matrix solution methods in Section 3.1, where we introduce a solution scheme that is particularly suitable for the problems with multiple right-hand sides by partially inverting the matrix factor. In Section 3.2, we discuss in detail the mass matrix-vector multiplications that are involved in the Lanczos scheme. The parallel generalized Lanczos procedure is given in Section 3.3.

3.1 Parallel Matrix Factorization

The parallel matrix factorization is based on a row-oriented storage scheme that takes full advantages of the sparsity of the (shifted) stiffness matrix, K_σ. The development of the parallel solution procedures is discussed in details in Reference [11]. In this section, we discuss the use of the parallel solution procedures for the generalized Lanczos algorithm. We first discuss a load assignment strategy for sparse matrices on a multiprocessing system. We then describe a parallel implementation of the LDL^T factorization procedure. An approach to partially invert a matrix factor is also discussed.

3.1.1 Parallel Assignment of Sparse Stiffness Matrix

The notion of elimination tree plays a significant role in sparse matrix study [14]. Let's define a list array ***PARENT***:

$$PARENT(j) = \min\{i \mid L_{i,j} \neq 0\} \tag{58}$$

The array ***PARENT*** represents the row subscript of the first nonzero entry in each column of the lower triangular matrix factor ***L***. The definition of the list array ***PARENT*** results in a monotonically

ordered (elimination) tree T of which each node has its numbering higher than its descendants [12,14,21]. With the definition of the array ***PARENT,*** the nonzero entries induced by a nonzero entry $K_{i,j}$ or $L_{i,j}$ can be determined based on the following statement:

Lemma 1: ***If*** $K_{i,j}$ (or $L_{i,j}$) $\neq 0$ ***then*** for each $k = PARENT(\ldots(PARENT(j)))$, $L_{i,k} \neq 0$ where $k < i$.

That is, the list array ***PARENT*** contains sufficient information for determining the nonzero pattern of any row in L [10,12,21].

If the elimination tree T is post-ordered topologically, the nodes in any subtree are numbered consecutively [12]. Furthermore, the resulting sparse matrix factor is partitioned into block submatrices where the columns/rows of each block correspond to the node set of a branch in T [13,15]. Figure 2 shows the matrix structure and its post-ordered elimination tree representation. This partitioning divides a sparse matrix into two basic data sets: principal block submatrices and the row segments outside the diagonal blocks [15].

The coefficients of a sparse matrix factor are distributively stored among the processors according to the column blocks. Figure 3 shows an example of the data assignment of a sparse matrix on multiple processors. The strategy is to assign the rows corresponding to the nodes along each branch (column block) of the elimination tree to a processor or a group of processors. Beginning at the root of the elimination tree, the nodes belonging to this branch of the tree are assigned among the available processors in a rotating block round robin fashion, or a block wrap mapping [3]. As we traverse down the elimination tree, at each fork of the elimination tree, the group of processors is divided to match the number and the size of the subtrees below the current branch. A separate group of processors is assigned to each branch at the fork and the process is repeated for each subtree. For a balanced elimination tree, the group of processors assigned to the branch is always a subcube or subring. Otherwise, the procedure is to follow as closely as possible the mapping of subcubes or subrings to subtrees. The process of assigning subcubes or groups of processors to each branch of the elimination tree continues until each subcube consists of only one processor, then all remaining nodes in the subtree are assigned to the single processor.

As noted earlier, a sparse matrix is partitioned into two basic sets: the principal diagonal block submatrices and the row segments outside the principal block submatrices. For the principal block submatrix, which has the profile structure, the processor assignment proceeds on a row group by row group basis. In our implementation, we assign a row group corresponding to a node in the finite element model, grouping individual degrees of freedom per that node as a unit.

The row segments are assigned to the processors that share the column block. When the node set of a branch in the elimination tree is shared among a number of processors, the rows are assigned to the processors sharing the node set (column block) in an alternating round robin or wrap fashion. That is, for a subtree-to-subcube mapping, two successive rows are assigned to the neighboring processors in the subring. This can be determined easily using a simple formula as follows:

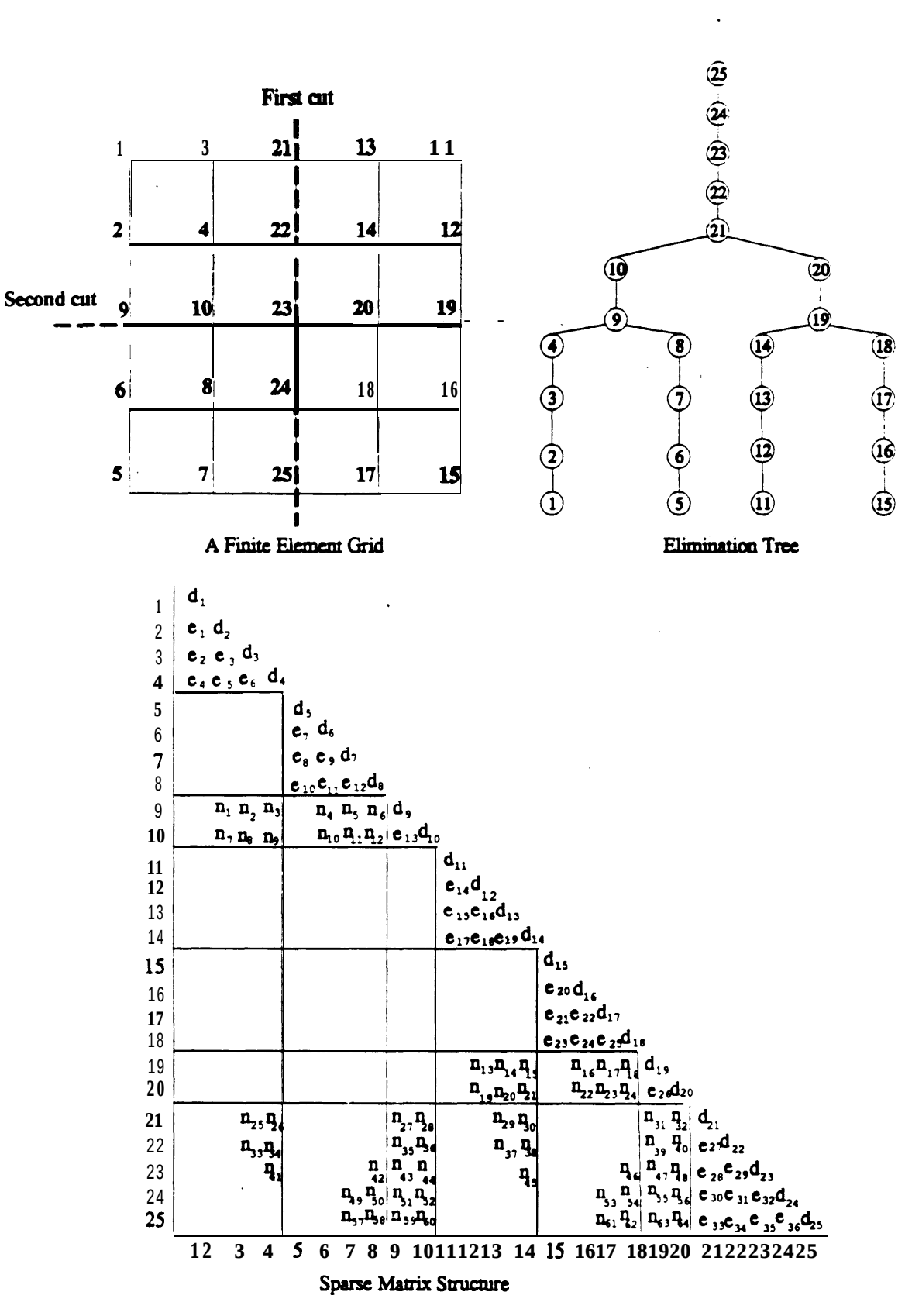


Figure 2: Sparse Matrix Structure and Post-Ordered Elimination Tree

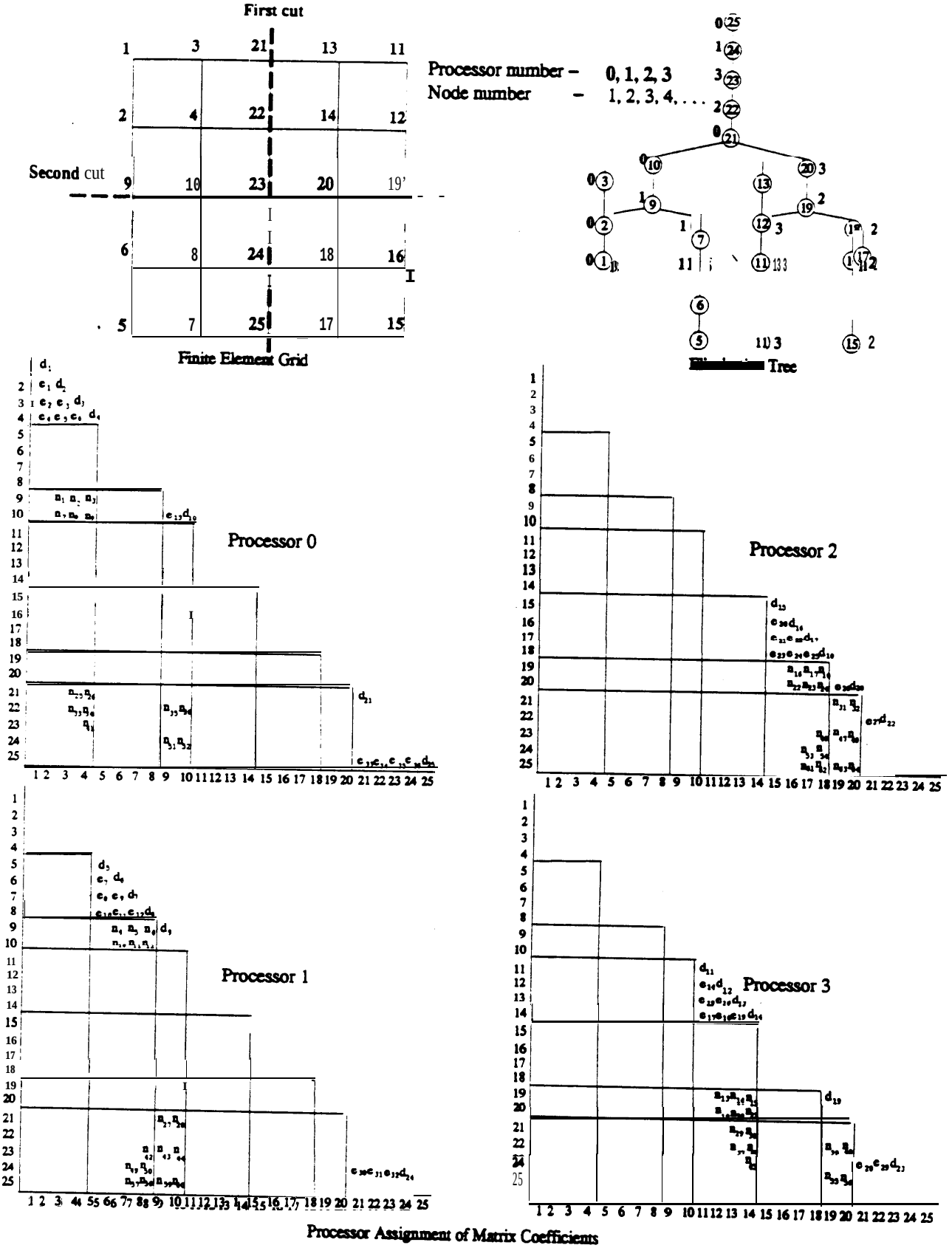


Figure 3: Matrix Partitioning for Parallel Computations

```
Procedure: processor( row_number, #_of_shared_processors, processor_list)
BEGIN
  index = mod(row_number, #_of_shared_processors);
  processor = processor_list[index];
END.
```

where *processor_list* is a list of processors sharing the column block, *index* points to the position in the list where the processor number can be found, and *processor* is the processor to which the row segment is assigned. Using this simple procedure, if the entire node set of a branch in the elimination tree is assigned to a single processor, the coefficients of the entire column block, including the row segments, are assigned to the same processor.

3.1.2 Parallel Matrix Factorization

The sparse matrix factorization is basically a block column scheme. The block factorization scheme consists of (1) a profile factorization for the principal diagonal block submatrices; and (2) a profile forward solve for the row segments per each column block. The matrix factorization is divided into two distinct phases. During the first phase, the column blocks assigned entirely to a single processor are factorized. During the second phase, the column blocks shared by more than one processor are factorized.

In the first phase, each processor independently factorizes the column blocks that are not shared by other processors. There are two distinct stages in this first phase of LDL^T decomposition.

D.I.l Factoring the column blocks entirely in the same processor:

D.I.l.l update the coefficients in the column block by the computed factors in the previous column blocks;

D.I.1.2 decompose the principal block submatrix;

D.I.1.3 factor the row segments by a series of forward solves with the principal block submatrix factor.

D.I.2 Forming dot products among the row segments. These dot products are then fanned-out to update the remaining matrix coefficients in the same processor or saved in the buffer to be fanned-in to another processor during the second phase of factorization.

The strategy is to carry out as much computations as possible in the processor. When a processor sends the dot products to another processor, all dot products saved in the buffer for that processor are sent as a package. This procedure is graphically illustrated as shown in Figure 4.

In the second phase of numerical factorization, the column blocks shared by more than one processor are factorized. The parallel factorization of a column block proceeds as follows:

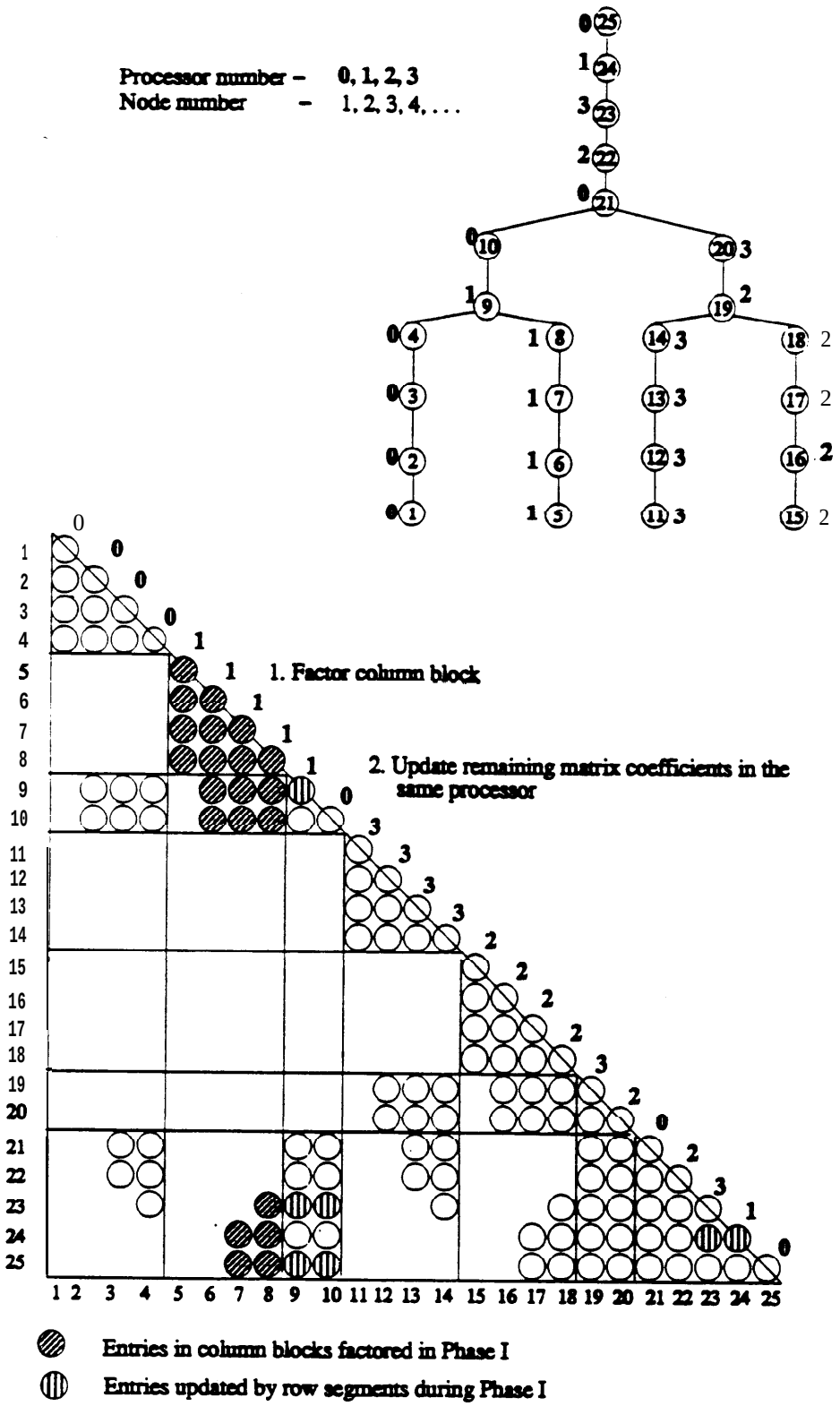


Figure 4: Phase I of Parallel Factorization Scheme

D.II.1 Each processor fans-in the dot products saved previously in the buffers on the other processors sharing the column block. The dot products received are used to update the principal block submatrix and the row segments.

D.II.2 Perform a parallel profile factorization and update the row segments. The profile factorization proceeds in a row by row basis. For each row in the principal block submatrix,

D.II.2.1 compute the row factor of L and D in the column block; and

D.II.2.2 broadcast the row factor and update the remaining coefficients in the column block.

D.II.3 Form dot products among row segments in the column block. This step consists of two basic operations:

D.II.3.1 Form dot products among the row segments stored in the processor.

D.II.3.2 Form dot products between the row segments stored in different processors. This operation is carried out by circulating the row segments of the column block among the processors sharing the column block. When a processor receives another processor's row segments, it forms the dot products between its own row segments and the row segments received from the neighboring processor. The row segments received are then passed on to the next processor.

- The dot products are fanned-out to update the remaining matrix coefficients in the same processor or saved in the buffer to be fanned-in to another processor (see Step D.II.1.)

This procedure is illustrated in Figure 5.

The forward solve is divided into two phases as shown in Figure 6. In the first phase, each processor calculates the portion of the solution vector corresponding to the column blocks which reside entirely within a single processor. Each processor also updates the shared portions of the solution vector based on the row segments in these column blocks residing in the processor. In tte second phase, the parallel forward solve for the shared portions of the vector is performed. This parallel procedure is carried out in a column block by column block basis. There are three basic operations for the parallel forward solve for the portion of solution vector shared by multiple processors:

F.II.1 Send and receive updates for the solution vector corresponding to the current block.

F.II.2 Calculate the solution for the current block using the principal block submatrix. Since the principal block submatrix is distributively stored, after each solution value is computed, it is broadcast to the processors sharing the column block to update the remaining coefficients in the solution vector.

F.II.3 Use the solution computed to update the remaining coefficients using the row segments in the column block.

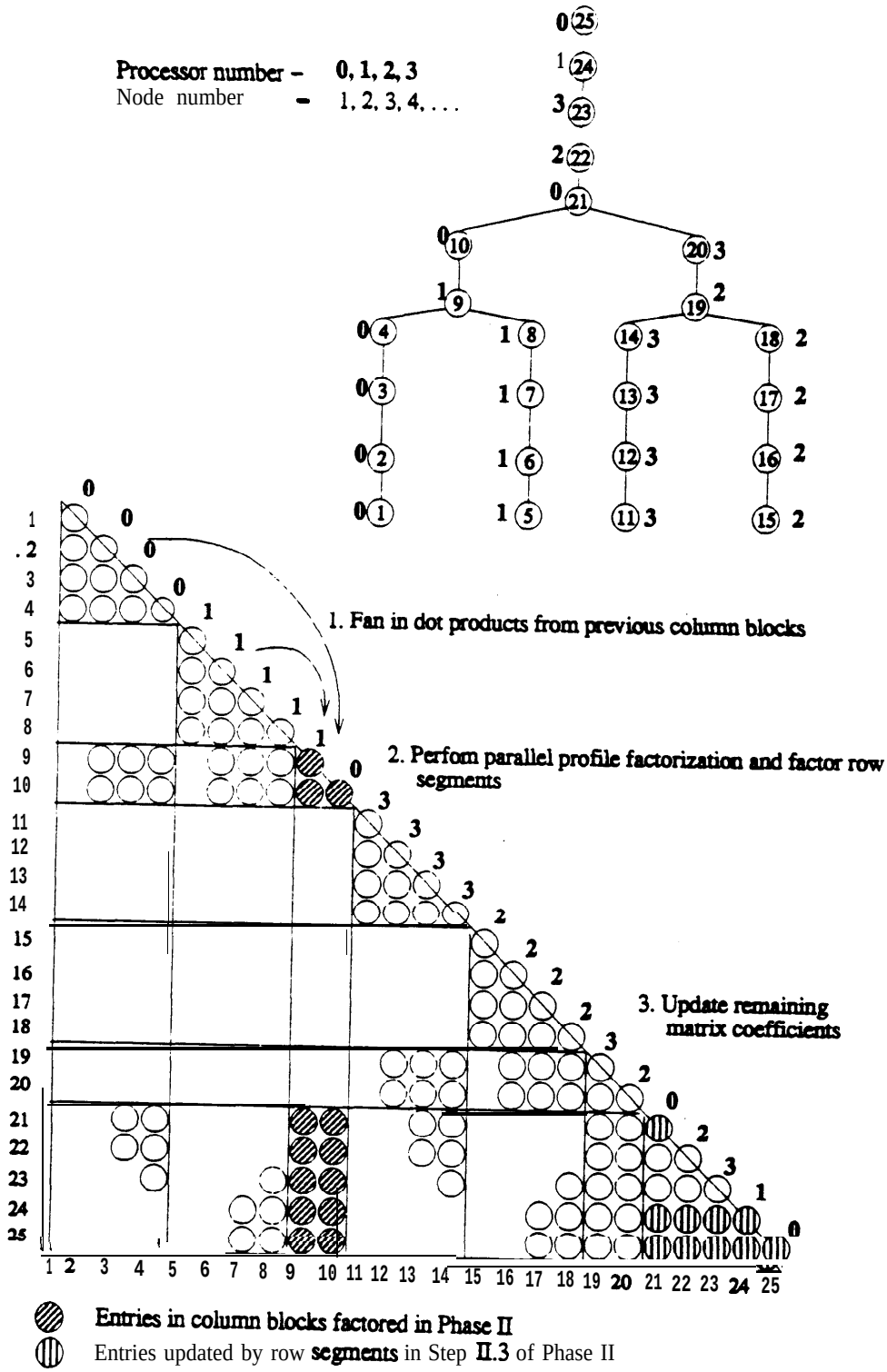
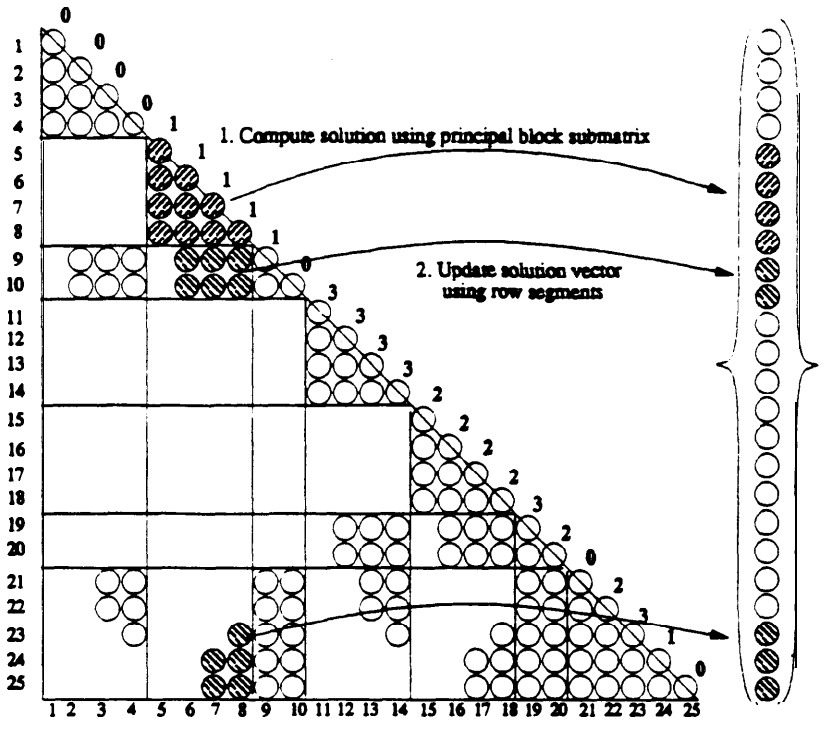
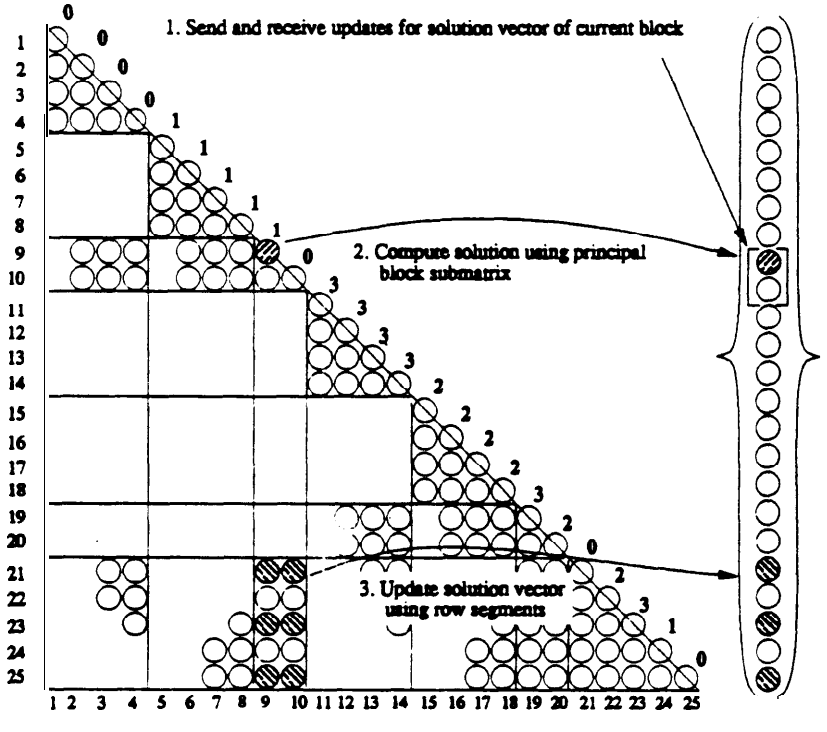


Figure 5: Phase II of Parallel Factorization Scheme



(a) Phase I of Forward Solution



(b) Phase II of Forward Solution

Figure 6: Parallel Forward Solve

In the forward solve, each processor begins working independently (in Phase I) and finishes working concurrently with all other processors on the last (root) column block.

The backward substitution procedure is essentially a reverse of the forward solve. The backward solution procedure is described in Figure 7. Similar to the forward solve and the factorization, the procedure is divided into two phases. Phase one deals with the portion of the solution vector shared by multiple processors. The procedure is essentially a reverse of Phase II in the forward solve and consists of the following steps for each shared column block:

B.I.1 Update the portion of solution vector corresponding to the current block by the row segments;

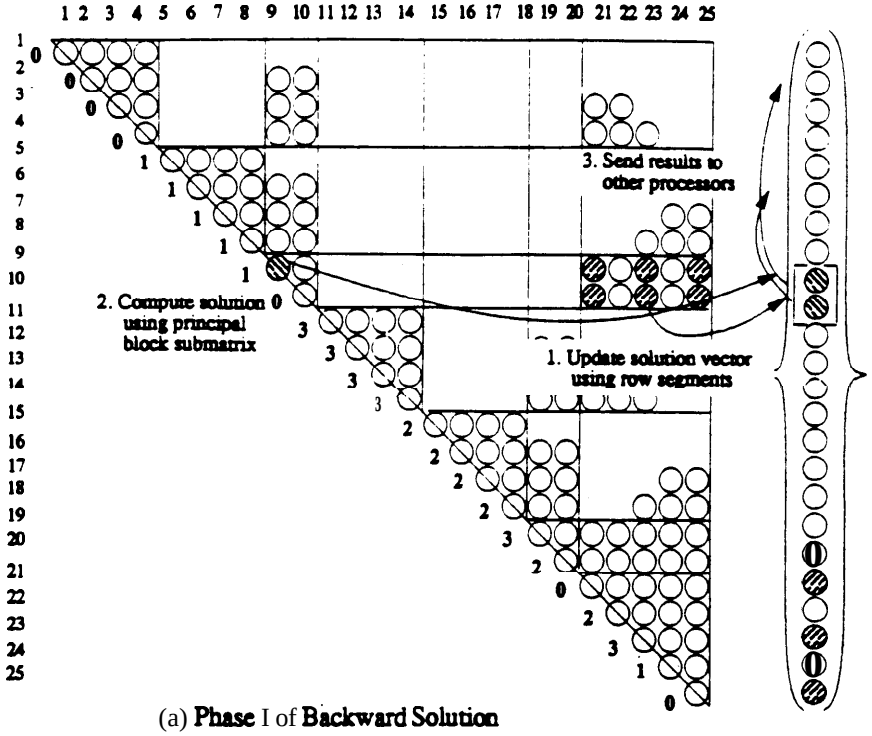
B.I.2 Calculate the solution for the current block using the principal block submatrix. After each solution value is computed, the solution vector is updated and the update is sent to the next processor to update the remaining coefficients of the solution vector;

B.I.3 Send the results to other processors.

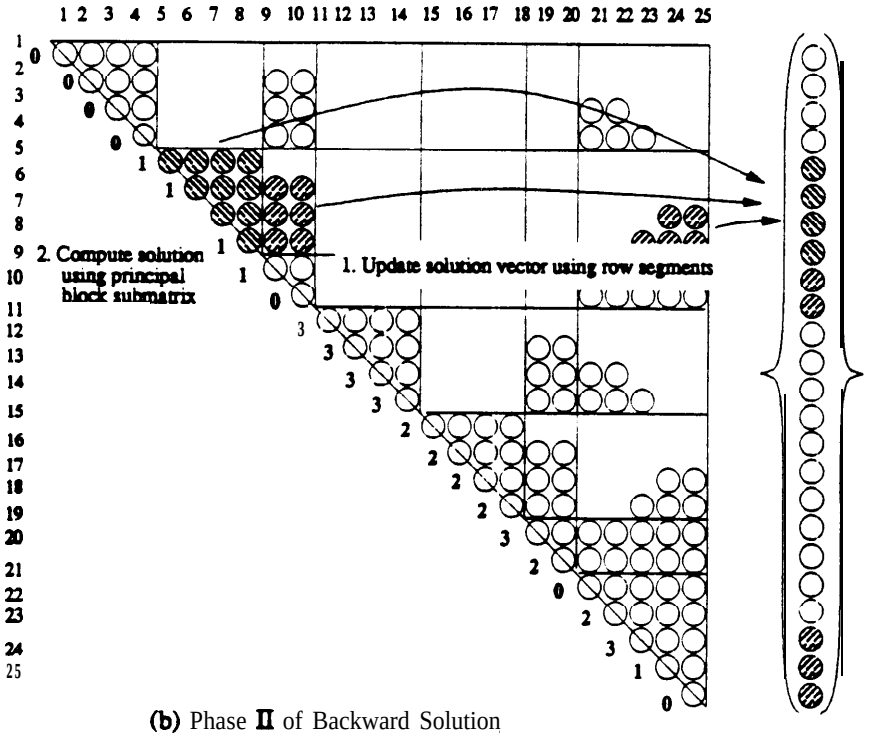
Our implementation of the backward solve for the principal profile submatrix follows closely the forward solve procedure described in Reference [3]. In the second phase, each processor calculates the portion of the solution vector correponding to the column blocks residing within a single processor. The processors perform the calculations independently without any processor communications and may complete the solution at different times.

3.1.3 Parallel Matrix Factorization with Partial Inverses

The Lanczos procedure for the generalized eigenvalue problems requires the solution of triangular systems at each Lanczos iteration step. While the parallel matrix factorization procedure described in the previous section performs well, the parallel forward and backward solves do not exhibit similar efficiency. It has been noted that there is little that can be done to improve the performance of the parallel triangular solvers [6]. However, when examining closely the procedures of the forward and backward solves, most of the parallelism come from assigning many column blocks to a single processor so that the processors can work independently. Reasonable parallelism also occurs when working with the distributed row segments. The main deficiency is due to the parallel solutions of the triangular systems for the dense principal submatrix factors (see Step F.II.2 of the forward solve and Step B.I.2 of the backward solve). The triangular solution procedures have significant number of communication overhead because of the data dependencies in the solution of the dense triangular systems. In this section, we describe an alternative method that can expedite the solution of triangular systems. The strategy is to invert the dense principal submatrix factors that are shared by multiple processors so that the triangular solution can be carried out by matrix-vector multiplication. The problem is to directly compute the inverse of a dense matrix factor.



(a) **Phase** I of **Backward Solution**



(b) Phase **II** of Backward Solution

Figure 7: Parallel Backward Solve

First, let's define L_i to represent the triangular matrix factor that contains the ith row of L :

$$L_i = \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & \ddots & & & & & \\ & & & 1 & & & & \\ L_{i,1} & \cdots & L_{i,j} & \cdots & L_{i,i-1} & 1 & & \\ & & & & & & 1 & \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{bmatrix} \tag{59}$$

The triangular matrix factor $\boldsymbol{L}$ can now be written as:

$$\boldsymbol{L} = L_1 \ldots L_i L_{i+1} \ldots L_n = L^{(i)} L_{i+1} \ldots L_n \tag{60}$$

where $L^{(i)} = L_1 \ldots L_i$ contains the first i rows of the triangular factor $\boldsymbol{L}$. The inverse of the triangular factor becomes:

$$L^{-1} = L_n^{-1} \ldots L_{i+1}^{-1} L_i^{-1} \ldots L_1^{-1} = L_n^{-1} \ldots L_{i+1}^{-1} L^{(i)-1} \tag{61}$$

where

$$L_i^{-1} = \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & \ddots & & & & & \\ & & & 1 & & & & \\ -L_{i,1} & \cdots & -L_{i,j} & \cdots & -L_{i,i-1} & 1 & & \\ & & & & & & 1 & \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{bmatrix} \tag{62}$$

In other words, L_i^{-1} can be obtained by negating the offdiagonal values in the ith row of $\boldsymbol{L}$. Note that the ith row, $L_{i,:}$, of $\boldsymbol{L}$ can be computed as:

$$L_{i,:}^T = L^{(i-1)-1}[K_{i,1}, \ldots, K_{i,i-1}, 1, 0, \ldots, 0]^T \tag{63}$$

Once the ith row of $\boldsymbol{L}$ is computed, the inverse of the matrix factor $L^{(i)}$ is given by

$$L^{(i)-1} = L^{(i-1)-1} L_i^{-1} \tag{64}$$

Based on the equations above, the procedure for inverting a lower triangular factor on a row by row basis can be summarized as follows:

D.II.2.1 Compute the row factor of $\boldsymbol{L}$ (and $\boldsymbol{D}$) by Equation 63.

D.II.2.2 Negate the entries of the row factor to form L_i^{-1} as in Equation 62.

D.II.2.3 Compute $L^{(i)-1}$ by multiplying L_i^{-1} with $L^{(i-1)-1}$ as in Equation 64.

D.II.2.4 Broadcast the inverted row factor and update the remaining coefficients in the column block.

The multiplication shown in Equation 64 only affects the entries on row i of $L^{(i)-1}$. Therefore, no additional processor communications are needed when $L^{(i)-1}$ is formed in the processor responsible for row i. We can apply this simple procedure to directly compute the inverses of the dense principal block submatrix factors. That is, the procedure for matrix factorization with partial inverses is essentially the same as the direct parallel LDL^T factorization except in the factorization of column block (Step D.II.2). The number of processor communications are the same for both the direct LDL^T factorization and the factorization with partial factor inverses.

As noted earlier, one approach to speed up the solution of triangular systems is to transform the triangular solution into matrix-vector multiplication by inverting portions of the matrix factors. With the inverses of the principal submatrix factors that are distributively stored in multiple processors, the main difference is to change Step F.II.2 of the forward solution procedure and Step B.I.2 of the backward solution procedure described earlier and to replace the procedures with matrix-vector multiplication between the principal submatrix inverses and the solution vector. In parallel matrix-vector multiplication, each processor calculates its contribution to the product; the partial products are then summed across all processors sharing the block to complete the matrix-vector multiplication. That is, we can reduce the communication to a single global summation among the processors shared by the column block.

3.2 Mass Matrix-Vector Multiplication

While the element stiffness matrices are assembled into the global stiffness matrix for factorization, the Lanczos procedure does not require the assembly of element mass matrices since the matrix is used mainly in the matrix-vector multiplication. In this section, we examine the mass matrix-vector multiplication with respect to the Lanczos procedure and show that the multiplication can be performed with either assembled global mass matrix or the unassembled element mass matrices.

3.2.1 Matrix-Vector Multiplication with Global Mass Matrix

The coefficients of the global mass matrix are assigned to multiple processors similar to the stiffness matrix except that only the nonzero entries (without fills) are stored with an explicit indexing scheme. It is well known that a matrix-vector product can be formed by first performing the multiplication with the coefficients in each processor and then sum the partial products across all processors. In this section, we examine how the mass matrix vector product, $p = Mq$, is computed and being used in the Lanczos procedure.

Let's denote the global mass matrix as $M^{(1)}, M^{(2)}, \ldots, M^{(i)}, \ldots M^{(np)}$, where $M^{(i)}$ represents the matrix partition in processor i and np is the number of processors. On a distributed memory computer, each processor computes partial matrix-vector multiplication $p^{(i)} = M^{(i)}q$, where $p^{(i)}$

represents the resulting matrix vector product in processor i. The results are then summed from each processor $p = \sum_{np} p^{(i)}$. Since only that portion of the vectors p and q involved in the multiplication need to be stored in a processor, the processor assignment for the vectors can be depicted as shown in Figure 8.

Now let's consider how the mass matrix vector product p is used in the Lanczos procedure. In the first case, the vector p is used to form a vector-vector dot product. When a dot product is to be formed, we have:

$$q^T p = q^T \sum_{i=1}^{np} p^{(i)} = \sum_{i=1}^{np} q^T p^{(i)}. \tag{65}$$

Therefore we can form $q^T p^{(i)}$ in each processor and then sum the scalar results over all processors; this global sum operation is often provided as a library routine, such as the procedure **globalsum** on the Intel's hypercube. Forming the global sum of scalar values is considerably cheaper than a global sum of vectors.

The second case is when the mass matrix-vector product p is to be used as a right-hand side vector for a system of linear equations, $K_\sigma r = p$. In this case it would appear that a global sum would need to be formed. However, if we examine closely Step **F.II.1** of the parallel forward solve described in Section 3.1, we notice that it involves sending and receiving updates for the shared block vector. Therefore, we can directly send and sum the partial vector $p^{(i)}$ as the forward solution procedure is performed. That is, the global sum of vector p takes place as part of the forward solution phase.

3.2.2 Matrix-Vector Multiplication with Element Matrices

The matrix-vector multiplication, Mq, can also be formed directly with the element mass matrices. The mass matrix-vector product can be written as:

$$\begin{aligned} p &= M q \\ &= A^T M^e A q \\ &= A^T M^e q^e \end{aligned} \tag{66}$$

In the above equation, M^e is a block diagonal matrix consisting of element mass matrices $M^{(i)}$, $i = 1, \ldots, nel$ where nel is the number of elements, and A is a boolean (kinematic) matrix denoting the (displacement/compatibility) relationship between the global (displacement) vector q and the element (displacement) vector q^e. Thus we can write:

$$p = A^T p^e \tag{67}$$

where $p^e = M^e q^e$. The (static) relationship shown in Equation 67 simply denotes summing the contributions of element (force) vector p^e into the global (force) vector p. Each processor can perform the element matrix-vector multiplication $(p^{(i)} = M^{(i)} q^{(i)})$. The element mass matrix-vector products are then accumulated or summed over the multiple processors.

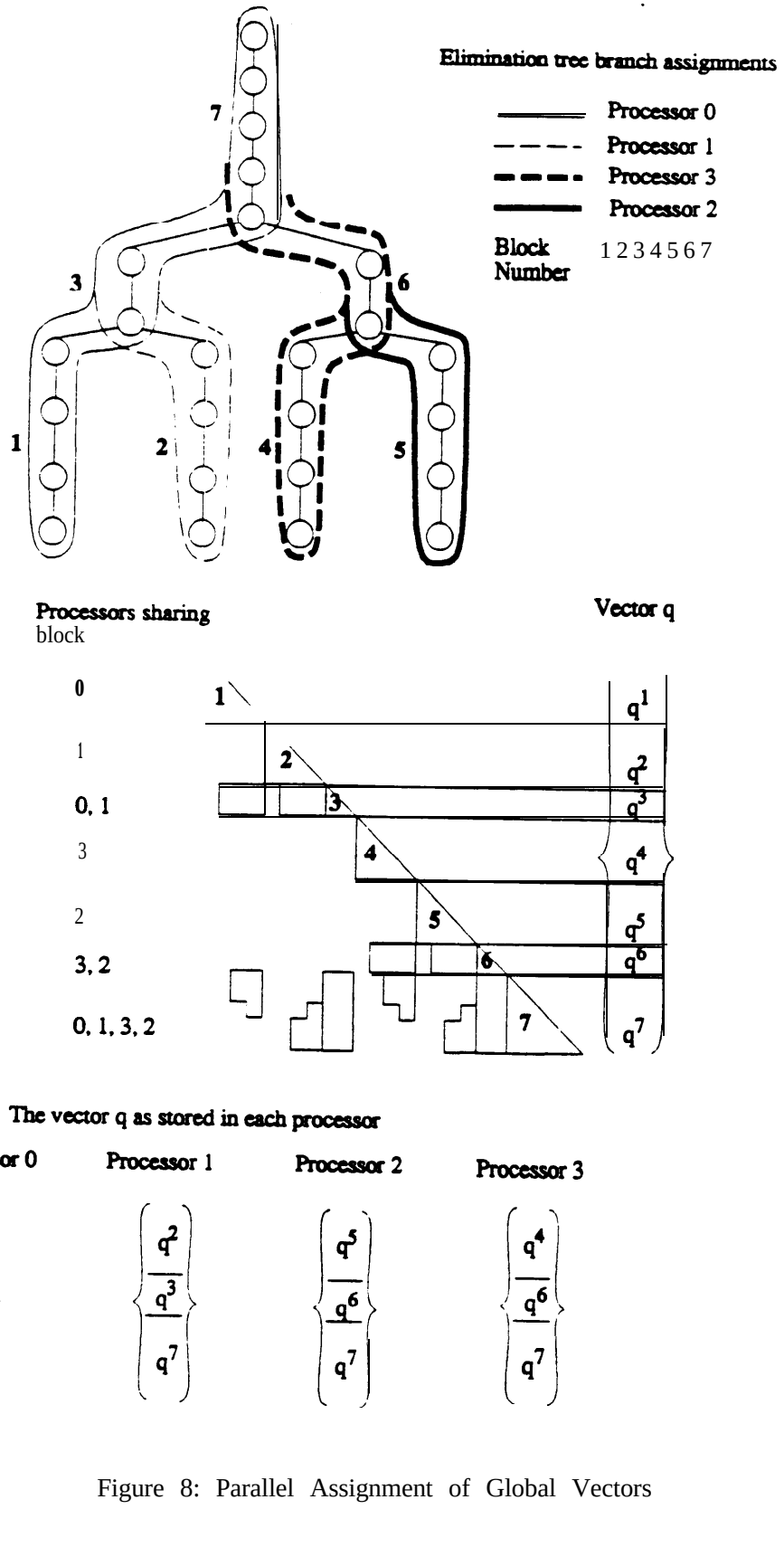


Figure 8: Parallel Assignment of Global Vectors

Let's now examine the use of the mass matrix-vector product p in the Lanczos procedure. From Equation 67, the vector product $q^T p$ can be rewritten as:

$$\begin{aligned} q^T p &= q^T A^T p^e \\ &= q^{eT} p^e \\ &= (q^{(1)T} \ldots q^{(nel)T})(p^{(1)} \ldots p^{(nel)})^T \\ &= \sum q^{(i)T} p^{(i)} \end{aligned} \tag{68}$$

where $q^e = Aq$ consists of element vectors $q^{(i)}$, $i = 1,\ldots, nel$. That is, the vector product can be obtained by summing over the multiple processors the scalar values resulting from the dot product between $q^{(i)}$ and $p^{(i)}$.

Let's consider that the mass matrix-vector product p is used as a right-hand side vector of the system of equations, $K_\sigma r = p$. As noted in Equation 67, the mass matrix-vector product p $(= A^T p^e)$ is a sum of the element vectors $p^{(i)}$. So when the vector p is used as a right-hand side vector in the parallel forward solve, the vector sum can be formed as part of the forward solve similar to the case for the assembled mass matrix. That is, the same forward solution procedure works for either assembled or unasembled mass matrices.

3.3 Parallel Generalized Lanczos Procedure

We will now introduce the parallel implementation of the generalized Lanczos algorithm. The procedure is summarized in Figure 9. The parallel factorization and solution procedures and the mass matrix-vector multiplication procedures described in the previous sections are employed in the implementation. In this section, we examine the difference between the parallel procedure and the sequential generalized Lanczos procedure described in Section 2.7. As shown in Figure 9, the Lanczos procedure is composed mainly of matrix-vector multiplications and vector operations. Based on the development discussed in Section 3.2, if no reorthogonalization is needed, the only communications required occur in the global sum operation in calculating α and β in lines 27, 38 and 42 of the procedure shown in Figure 9(a). When the global sum is performed, all processors must be synchronized.

A form of synchronization is also needed in the solution phase, see lines 10 and 34 of the procedure shown in Figure 9(a). The processors may begin the forward solution procedure asynchronously since each processor computes the factors of the column blocks residing on the processor. At the completion of the forward solution, all processors are synchronized before the backward solution procedure begins. In the backward solution, the processors begin working together on the same last (root) column block and complete the calculations asynchronously by working independently on the column blocks residing entirely in individual processors. As discussed in Section 3.1.2, the forward and backward solution of triangular systems require a number of messages passing among the shared matrix column blocks. Similarly, the factorization step also involves significant amount of message passing and synchronization.

```
Procedure: Parallel Generalized Lanczos Procedure
BEGIN
1   WHILE (# of converged eigenvalues < # of desired eigenvalues) DO
2   BEGIN
3       Choose a shift σ ;
4       Form K_σ = K − σM ;
5       Factor K_σ = LDL^T ;
6       Set β_0 = 0 ;
        /* Compute residual of Ritz vector after spectral rhift */
7       FOR i = 1 to number of converged Ritz vectors, DO
8           p = Mφ_i ;
9           Scatter(p, y) ; /* distribute p to vector y of kngth n */
10          Solve y = K_σ^{-1} y /* forward and backward solves */
11          Gather(r, y) ; /* assign y to local vector r of kngth = depth of tree */
12          r = r − 1/(λ_i − σ) φ_i ;
13          p = Mr ;
14          ||u_i|| = p^T * r ;
15      END.
16      globalsum(u)/*u.[||u_1||.. ||u_i||]^T
17      FOR i = 1 to number of converged Ritz vectors, DO
18          ||u_i|| = √||u_i|| ;
18      END.
20      Choose r ;
        /*orthogonalize r to all converged Ritz vectors ; */
21      FOR i = 1 to number of converged Ritz vectors, DO
22          r_{i,j} = -1. ;
23      END.
24      Parallel_Selective_Reorthogonalization ;
25      p = Mr ;
26      β = r^T p ;
27      β = globalsum(β) ;
20      β_1 = √β ;
29      q_1 = r/β_1 ;
30      p_1 = p/β_1 ;
        /* Lanczos Loop */
31      FOR j = 1 UNTIL insufficient space for Lanczos vectors q_j DO
32      BEGIN
33          Scatter(p, y) ; /* distribute p to vector y of kngth n */
34          Solve y = K_σ^{-1} y /* forward and backward solves */
35          Gather(r, y) ; /* assign y to local vector r of length = depth of tree */
38          r = r − β_{j-1} * q_{j-1} ;
37          α = p^T r_i
38          α_j = globalsum(α) ;
38          r = r − α_j * q_j ;
40          p = Mr ;
41          β = r^T p ;
42          β = globalsum(β);
43          β_{j+1} = √β;
44          Parallel_Reorthogonalization;
45          q_{j+1} = r/β_{j+1} i
46          p = p/β_{j+1} i
47          Solve eigensystem T_j for (ϑ_i, s_{j,i}), i = 1,. .., j;
48          Test for convergence of eigenvalues;
49      END.
50      Parallel_Ritz_Refinement;
51  END.
END.
```

(a) Parallel Lanczos Procedure

Figure 9: Parallel Generalized Lanczos Algorithm

```
Procedure: Parallel_Reorthogonalization
/* ε_s = ε√n where ε denotes machine precision */
BEGIN
   Parallel_Selective_Reorthogonalization ;
   Parallel_Partial_Reorthogonalization ;
END
Procedure: Parallel_Selective_Reorthogonalization
BEGIN
   /*external selectivereorthogonalization-□ℳ□↗□□○ asneeded*/
1  FOR i = 1 ... TO number of converged eigenpairs (6, φ̂), DO
2  BEGIN
3     IF τ_{i,j} ≥ 0 THEN τ_{i,j+1} = (|(θ̂_i − α_j)τ_{i,j} − β_j τ_{i,j−1}| + ‖u_i‖)/β_{j+1};
4     IF τ_{i,j+1} ≥ √εn OR τ_{i,j} < 0 THEN
         /* parallel operation: reorthogonalise q_{j+1} ♦□ φ̂_i */
♦        ζ = q_{j+1}^T M φ̂_i ;
6        ζ = globalsum(ζ) ;
7        q_{j+1} = q_{j+1} − ζ * φ̂_i ;
8        IF (τ_{i,j} < 0) THEN /* sequential operations */
9           τ_{i,j} = 0 ;
10          τ_{i,j+1} = ε_s ;
11       ELSE
12          τ_{i,j+1} = −1 ;
13       ENDIF.
14    ENDIF.
15 END.
END
Procedure: Parallel_Partial_Reorthogonalization
BEGIN
   /* partial reorthogonalisation – perform as needed */
1  η_{j+1,j+1} = 1; /* sequential operations begin */
2  η_{j+1,j} = max{ε_s, ε_s/β_{j+1}};
3  η_{j+1,j−1} = ((α_{j−1} − α_j)η_{j,j−1} + β_{j−1}η_{j,j−2})/β_{j+1} ;
4  FOR k = 1 TO j − 2, DO
5  BEGIN
6     η_{j+1,k} = (β_{k+1}η_{j,k+1} + β_k η_{j,k−1} − β_j η_{j−1,k} + (α_k − α_j)η_{j,k})/β_{j+1} ;
7  END.
8  η_max = max_{k=1,j}(η_{j+1,k})/* sequential operations end */
9  IF η_max > √εn THEN /* parallel operations begin */
10    p_1 = M q_j ;
11    p_2 = M q_{j+1} ;
12    FOR i = 1 TO j − 1, DO
13    BEGIN
14       work[i] = p_1^T q_i ;
15       work[i + j − 1] = p_2^T q_i ;
16    END.
17    work = globalsum(work) ; /* vectosum */
18    FOR i = 1 TO j-1, DO
19    BEGIN
20       q_j = q_j − work[i] * q_i ;
21       q_{j+1} = q_{j+1} − work[i + j − 1] * q_i ;
22    END.
23    p_1 = M q_{j+1} ;
24    ρ = p_1^T q_{j+1} ;
25    ρ = globalsum(ρ) ;
26    REPEAT
27       s = p_1^T q_j ;
28       s = globalsum(s) ;
29       q_{j+1} = q_{j+1} − s * q_j ;
30       p_1 = M q_{j+1} ;
31       ρ_old = ρ ;
32       ρ = p_1^T q_{j+1} ;
33       ρ = globalsum(ρ) ;
34    UNTIL (ρ/ρ_old ≥ √2/2) ;
35    β_{j+1} = √ρ ;
36 ENDIF. /* parallel operations end */
END.
```

(b) Parallel Reorthogonalization

Figure 9 (continued)

```
Procedure: Parallel_Ritz_Refinement
BEGIN
   /* Form eigenvectors of converged eigenvalues */
   /* Use TQL2 to obtain eigenpairs of tridiagonal matrix*/
1  Solve eigensystem T_j for (λ_i, s_i), i = 1,...,j ;
2  l = 0 ;
3  FOR i = 1 TO j, DO
4  BEGIN
5     IF '(eigenvalue i converged) THEN
6        l = l + 1; /* counter for converged eigenvalues */
7        φ̂_l = 0 ;
8        FOR k = 1 TO j, DO /* parallel operation */
9        BEGIN
10          φ̂_l = φ̂_l + s_{k,i} * q_k ;
11       END.
12       φ̂_l = φ̂_l + (β_{j+1} s_{j,i}/ϑ_i) * q_{j+1} ; /* parallel refinement */
13    ENDIF.
14 END.
   /* parallel orthogonalization of eigenvectors for each converged eigenvalue*/
15 FOR i = 1 TO l, DO
16 BEGIN /* parallel operations */
17    p = M φ̂_i ;
18    ρ = φ̂_i^T p ;
19    ρ = globalsum(ρ);
      /*normalize Ritz vector and scale mass matrix vector product */
20    φ̂_i = φ̂_i/√ρ ;
21    p = p/√ρ ;
      /* orthogonalize following Ritz vectors */
22    FOR k = i + 1 TO l, DO
23    BEGIN
24       work[k − i] = φ̂_k^T p ;
26    END.
26    work = globalsum(work); /* vector sum */
27    FOR k = i + 1 TO l, DO
20    BEGIN
29       φ̂_k = φ̂_k − work[k − i] • φ̂_i ;
30    END.
31 END.
END.
```

(c) Parallel Procedure for Ritz-Vector Refinement

Figure 9 (continued)

While most operations in the Lanczos algorithm are well suited for parallization, certain operations are duplicated in each processor. For example the eigenvalues of the tridiagonal matrix T are solved for at each Lanczos iteration. Since the size of the tridiagonal matrix T is usually small and the solution of this small tridiagonal eigensystem problem is quite fast on a single processor, any attempt to distribute the calculations over a number of processors would in fact slow down the execution because of the communication overhead. As shown in line 47 of the procedure shown in **Figure** 9(a), each processor computes the eigenvalues and eigenvectors of the. tridiagonal matrix using a modification of the routine **TQL2** of the EISPACK software package so that it only solves for the last entries of the eigenvectors instead of the entire eigenvector [24]. The routine generates all the information needed to test for convergence of the eigenvalues, so this step is also duplicated in each processor.

For the parallel reorthogonalization procedure shown in Figure 9(b), the steps to test for loss of orthogonality are duplicated in each processor. If orthogonality has been deteriorated, then the work to orthogonalize q_j and q_{j+1} is distributed among multiple processors. The matrix-vector and vector-vector multiplications are performed in parallel as described in Section **3.2.** The results are accumulated across all the processors using the global sum operation. **As** shown in Figure 9(b), the procedure for reorthogonalization requires four additional **globalsum** operations to sum the vector dot products: one for a vector of length $2(j-1)$ and three for simple scalar values.

Figure 9(c) summarizes the parallel procedure for Ritz-vector refinement. After all desirable eigenvalues are obtained, the routine **TQL2** is used to compute the eigenvectors of the tridiagonal matrix; the operation is duplicated in each processor. The Ritz vectors are then refined in that each processor works on its portion of the vectors. Finally, for each converged eigenvalue, a parallel orthogonalization procedure is used to refine the eigenvectors.

4 Experimental Results and Discussions

The procedures described in the previous section have been implemented in a finite element program written in the C programming language and run on au Intel iPSC/i860 hypercube. Version **2.0** of the compiler and optimized level 1 BLAS routines were used. In this section, we present the results on two different finite element models that we have used to evaluate the parallel Lanczos procedure. The two models are a set of square finite element grids and a high speed civil transport model.

The square grid model is ordered using a coordinate nested dissection scheme which recursively partitions the grid into smaller subgrids and provides a very regular and well balanced work load distribution on a parallel computer. The civil transport model is an irregular model that does not yield to good load balance for the a number of re-ordering schemes that we have experimented with. Here, we show the results based on an incomplete nested dissection scheme. Figures 10 and 11 show, respectively, the square grid and the civil transport model; the number of equations and the number of nonzero entries in the stiffness matrices and the matrix factors are also shown in the figures.

In this implementation, the (shifted) stiffness matrix factor, mass matrix, converged eigenvectors and Lanczos vectors are all stored in memory. For the square grid model, the mass matrix is not assembled and the computations are carried out with the element mass matrices. The matrices for the civil transport model, however, are provided in assembled form.

We conduct two different types of experiments to evaluate the Lanczos procedure. The first experiment is intended to examine the various steps in the parallel Lanczos procedure. In the second experiment, we examine the situation when multiple spectral shifts are required due to insufficient space for storing the Lanczos vectors. To initialize the Lanczos procedure, we use the following initial heuristic shift [5]:

$$\sigma = \frac{1}{\sqrt{neq} * \sum \frac{m_{ii}}{k_{ii}}}$$

The results are discussed in the following.

For the first set of experiment, we run the Lanczos procedure for 40 iterations without spectral shifts other than the initial shift. Our objective is to examine the performance of the Lanczos algorithm implemented and to compare the effectiveness in the use of the factorization with partial factor inverses and the direct LDL^T factorization. The results for the square finite element grid models are tabulated as shown in Table 1. It is clear that the use of factorization with partial factor inverses is more efficient than the direct LDL^T factorization, particularly when the number of processors increases. Furthermore, the processors are utilized more effectively for larger problems. In Table 2, we profile the steps in the Lanczos procedure for an 120 by 120 square finite element model. It is interesting to note that the most costly step is the forward and backward solutions which further explain the importance in the use of partial factor inverses. Finally, we can also observe that, when only a few eigenvalues are solved, the sequential eigensolution of tridiagonal system is quite inexpensive comparing to the other operations in the Lanczos procedure.

Similar results are obtained as shown in Table 3 for the civil transport model. Again, there is a moderate gain in the solution time when using the factorization scheme with partial factor inverses. As shown in Table 3, which gives the profile on the various steps of the Lanczos procedure, the factorization cost is the most expensive operation when using eight processors but the forward and backward solution of triangular system of equations dominate the computation when 32 processors are utilized. As shown in the table, the benefit of using 32 processors is not high because of the problem size and that, as noted earlier, the computational loads on the processors are not well balanced for this irregular finite element model.

In the second experiment, we test the Lanczos procedure for problems that may require multiple shifts due to insufficient memory space for storing the Lanczos vectors. As noted earlier, external selective re-orthogonalization is used when a new shift is selected to ensure that the starting Lanczos vector is orthogonal to the previous set. The shifted stiffness matrices are factorized with partial factor inverses. We select the civil transport model as a test problem and solve for 60 and 105 eigenvalues using 8, 16 and 32 processors. As more eigenvalues are solved on small number of processors,

elements per side	number of elements	number of equations	number of nonzeros (in matrix factor L and D)
80	6,400	13,114	841,951
100	10,000	20,394	1,450,027
120	14,400	29,274	2,221,911
150	22,500	45,594	3,736,351

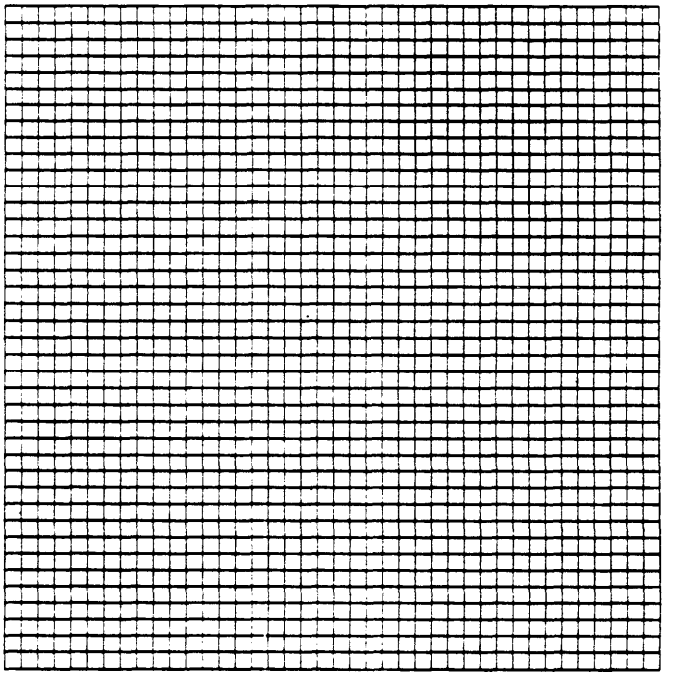


Figure 10: Square Plane Stress Finite Element Grid Models

number of equations	number of nonzeros (in lower triangle of K)	number of nonzeros (in matrix factor L and D)
16,146	515,651	*3.783.704*

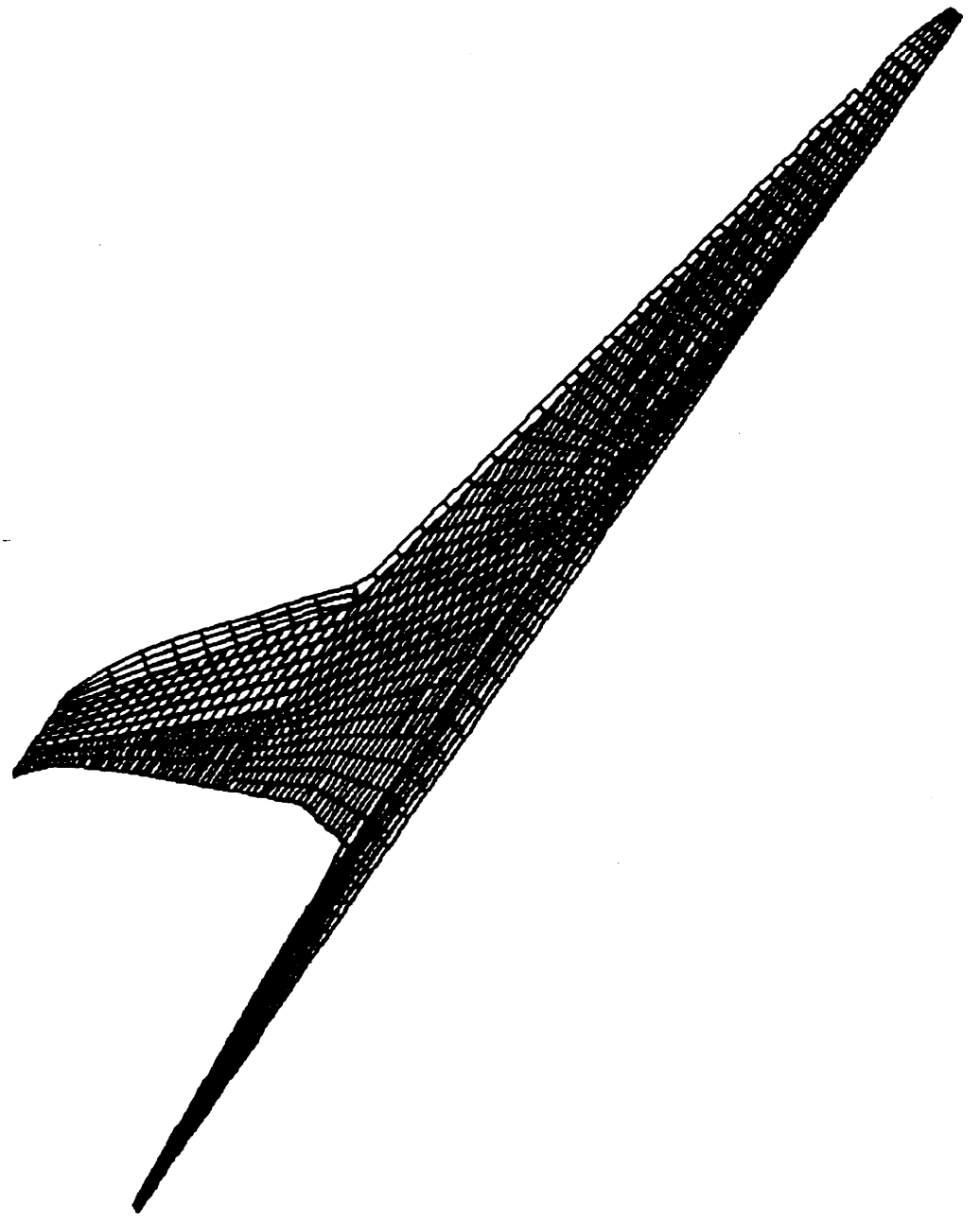


Figure 11: A High Speed Civil Transport Model

Table 1: Solution Time for Square FEM Grid Models (Time in seconds, *40* Lanczos iterations)

Number of processors	with LDL^T Factorization	with partial matrix factor inverse
80 by 80 mesh (13,114 equations, 17 eigenvalues)		
2 PROCESSORS	***42.77***	***42.42***
4 PROCESSORS	23.55	22.93
8 PROCESSORS	14.71	13.76
16 PROCESSORS	10.87	9.40
32 PROCESSORS	10.06	8.01
100 by 100 mesh (20,394 equations, 16 eigenvalues)		
2 PROCESSORS	68.60	68.13
4 PROCESSORS	36.79	35.98
8 PROCESSORS	22.18	20.90
16 PROCESSORS	15.42	13.52
32 PROCESSORS	13.82	10.92
120 by 120 mesh (29,274 equations, 16 eigenvalues)		
4 PROCESSORS	53.94	53.00
8 PROCESSORS	31.63	29.90
16 PROCESSORS	20.93	18.55
32 PROCESSORS	17.83	14.21
150 by 150 mesh (45,594 equations, 17 eigenvalues)		
8 PROCESSORS	50.18	48.08
16 PROCESSORS	31.98	28.97
32 PROCESSORS	25.34	20.48

Table 2: A Profile of Solution Time for a 120 by 120 Grid Model (Time in seconds, 40 Lanczos iterations, 16 converged eigenvalues)

Number of processors	LDL^T factorization 4	8	16	32	Partial matrix factor inverse 4	'8	16	32
Spectral shift	0.99	0.55	0.31	0.21	0.99	0.55	0.31	0.21
Factor $K_\sigma = LDL^T$	12.76	7.77	5.08	4.05	13.04	7.87	5.05	4.00
Data initialization	0.20	0.10	0.05	0.03	0.20	0.10	0.05	0.03
Triangular Solution	19.35	12.08	9.22	9.39	18.14	10.43	6.87	5.83
Formation of α, β and τ	7.99	4.03	2.05	1.19	7.99	4.03	2.06	1.18
Reort hogonalization	5.54	2.81	1.45	0.83	5.54	2.81	1.45	0.83
Tridiagonal eigensolver	1.21	1.23	1.23	1.21	1.21	1.23	1.23	1.21
Formation of Ritz vectors	5.71	2.98	1.50	0.91	5.71	2.79	1.50	0.91
Miscellaneous	0.19	0.09	0.04	0.02	0.19	0.09	0.04	0.02
Total	53.94	31.63	20.93	17.83	53.00	29.90	18.55	14.21

Table 3: A Profile of Solution Time for the Civil Transport Model (Time in seconds, 40 Lanczos iterations, 9 converged eigenvalues)

Number of processors	LDL^T factorization 8	16	32	Partial inverse 8	16	32
Spectral shift	1.52	0.58	0.24	1.52	0.57	0.24
Factor $K_\sigma = LDL^T$	22.63	15.84	12.53	23.34	16.19	12.48
Data Initialization	0.09	0.07	0.03	0.09	0.07	0.03
Triangular Solutions	16.58	14.58	18.32	14.17	11.14	12.25
Formation of α, β and τ	3.46	2.73	1.41	3.45	2.74	1.41
Reort hogonalization	4.43	3.59	2.00	4.44	3.59	2.00
Tridiagonal eigensolver	1.37	1.37	1.37	1.37	1.37	1.37
Formation of Ritz vectors	1.33	1.10	0.720	1.33	1.11	0.72
Miscenallaneous	0.06	0.06	0.05	0.06	0.06	0.05
Total	51.46	39.91	36.66	49.77	36.82	30.54

spectral shifts are required because of insufficient memory spaces on the 8 and 16 processors. The results are tabulated in Table 4.

Let's examine closely the solution time required to solve the eigenvalues for the civil transport model. We separate the timing results for computation and for input and output of the stiffness matrix. From the computation point of view, the forward and backward solutions remain to be a very costly step. Since each shift requires that the shifted stiffness matrix to be refactorized, the factorization cost for spectral shifts can become expensive. However, when spectral shift is used, the tridiagonal matrix T becomes small and the sequential eigensolution of the tridiagonal matrix is very efficient. When spectral transformation is not used, the sequential eigensolution of the tridiagonal matrix can become expensive and the number of Lanczos iterations and reorthogonalizations may also increase. There appears that, besides the case when insufficient space is available for storing the Lanczos vectors, spectral shifts should also be used to optimize the number of Lanczos iterations, reorthogonalizations and eigensolution of tridiagonal systems.

As noted in Table 4, another cost in the spectral shift is the input and output of the stiffness matrix. In our implementation, the stiffness matrix is stored using secondary storage. When a spectral shift is performed, the stiffness matrix is re-stored to compute the shifted stiffness matrix $K_\sigma = K - \sigma M$. Presently, the input and output on the Intel's i860 hypercube are fairly time consuming. One way to improve the efficiency of the I/O operations is to interleave the factorization procedure and the input of the stiffness matrix. In summary, the use of more processors is beneficial to minimize the number of re-starts when memory storage is limited. The optimal use of input and output devices and spectral shifts is, however, system and architecture dependent.

5 Summary

In this paper, we have discussed an implementation of generalized Lanczos procedure for distributed memory parallel computers. While the Lanczos procedure is well suited for parallization, the forward and backward solutions required at each step of Lanczos is expensive, particularly when only a few eigenvalues is desired. We have developed a strategy to invert the dense principal submatrix factors that are shared among multiple processors. Although the number of operations required for the factorization increases slightly, the number of communications remains the same with or without the inversion of submatrix factors. With the partial factor inverses, the parallel solution of triangular systems can be made more efficient, and higher parallelism for the triangular solution process can be recognised. The benefit of this factorization with partial factor inverses is clearly demonstrated for the test problems used in this study. We believe that the scheme will work even better with block Lanczos algorithm because more computations are distributed among the processors in the triangular solution process. Furthermore, the block Lanczos scheme may justify solving the block tridiagonal eigensystem in parallel rather than duplicating the computations on each processor.

Our implementation includes partial and external selective reorthogonalizations in the Lanczos

Table 4: Solution Time for 60 and 105 Eigenvalues for the Civil Transport Model (Time in seconds)

Solution for 60 eigenvalues			
Number of processors	8 processors	16 processors	32 processors
Total number of Spectral shifts	3	1	1
Total number of lanczos iterations	167	127	127
Total Number of partial reorthogonalizations	49	63	63
Solution time (in seconds):			
Data Initialization:			
Setup τ, η, u and other parameters	31.87	*0.07*	*0.04*
Reorthogonalize initial vector	0.34		
Spectral shifts	4.62	0.53	0.25
Factoring $K_\sigma = LDL^T$	*69.74*	16.20	12.57
Triangular solutions	10.48	36.69	36.69
Formation of α and β and τ	14.38	8.67	4.62
Reort hogonalization	27.11	19.30	12.30
Tridiagonal eigensolutions	10.48	36.69	36.69
Formation of Ritz vectors	17.23	35.94	21.45
Miscellaneous	0.27	0.18	0.15
Total solution time	235.22	153.00	126.77
Input of stiffness matrix	59.44		
Output of stiffness matrix	69.30	80.17	81.33
Total time	363.96	233.17	208.10
Solution for 105 eigenvalues			
Total number of Spectral shifts	*54*	*2*	1
Total number of lanczos iterations	*744*	280	287
Total Number of partial reorthogonalizations	122	130	143
Solution time (in seconds):			
Data Initialization:			
Setup τ, η, u and other parameters	1,558.43	41.91	0.03
Reorthogonalize initial vector	0.93	0.40	
Spectral shifts	73.84	1.34	0.24
Factoring $K_\sigma = L\ DL^T$	1,317.56	30.83	12.47
Triangular solutions	261.21	77.26	87.88
Formation of α and β and τ	*74.77*	19.07	10.05
Reort hogonalization	86.91	66.41	52 .00
Tridiagonal eigensolutions	3.82	268.45	381.03
Formation of Ritz vectors	265.58	80.36	63.05
Miscellaneous	1.26	0.41	0.34
Total solution time	3,644.31	586.04	607.10
Input of stiffness matrix	1,225.30	33.26	
Output of stiffness matrix	66.62	76.99	81.17
Total time	4,936.24	696.28	688.26

process. A new spectral shift is selected when memory space is insufficient to store the Lanczos vectors. Each new shift or re-start requires restoring the stiffness matrix, re-factorization of the shifted stiffness matrix and re-orthogonalization of the new Lanczos vector with respect to the converged eigenvectors. The number of processors should be used to store as many Lanczos vectors as possible so that the number of re-starts can be minimized. Based on our experimental results, use of more processors is well justified for large problems and when relatively large number of eigenvalues is needed. Last but not least, for each parallel computer system, some criterion is needed to measure the tradeoffs between spectral shifts and computation costs.

Acknowledgment

The authors would like to thank Prof. Gene Golub of Stanford University and Dr. Horst Simon of NASA Ames Research Center for many helpful discussions about the generalized Lanczos procedure. The data for the civil transport model was provided by Dr. Olaf 0. Storaasli of NASA Langley Research Center. This work is sponsored by the National Science Foundation grant number ECS-9003107 and the Army Research Office grant number DAAL-03-91-G-0038.

References

[1] T. Ericsson and A. Ruhe. The spectral transformation lanczos method for the numerical solution. of large sparse generalized symmetric eigenvalue problems. ***Math. Comp.,*** 35:1251–1268, 1980.

[2] G. H. Golub, R. Underwood, and J. H. Wilkinson. ***The Lanczos algorithm for the symmetric*** $Ax = \lambda Bx$ ***problem.*** Technical Report STAN-CS-72-270, Stanford University Department of Computer Science, Stanford University CA 94305, 1972.

[3] G. H. Golub and C. F. Van Loan. ***Matrix Computations .*** The Johns Hopkins University Press, 1989.

[4] R. G. Grimes, J. G. Lewis and H. D. Simon. ***The implementation of a block shifted and inverted Lanczos algorithm for eigenvalue problems in structural engineering.*** Technical Report ETA-TR-39, Applied Mathematics Unit, Boeing Computer Services, 1986.

[5] R. G. Grimes, J. G. Lewis and H. D. Simon. ***A shifted block Lanczos algorithm for solving sparse symmetric genemlized eigenproblems.*** Technical Report RNR-91-012, NASA Ames research Center, 1991.

[6] M. T. Heath, E. Ng and B. Peyton. Parallel algorithms for sparse Linear systems. ***SIAM Review,*** 33:420-460, 1991.

[7] M. T. Jones and M. Patrick. ***Lanz: Software for solving the large sparse symmetric genemlized eigenproblem.*** Technical Report NAS1-18605, ICASE, NASA Langley Research Center, 1990.

[8] M. T. Jones and M. Patrick. ***The use of Lanczos's method to solve the large generalized symmetric definite eigenvalue problem in parallel.*** Technical Report 90-48, ICASE, NASA Langley Research Center, 1990.

[9] S. Kaniel. Estimates for some computational techniques in linear algebra. ***Math. Comp., 20:369-378, 1966.***

[10] K. H. Law and S. J. Fenves. A node-addition model for symbolic factorization. ***ACM Trans. Math. Software,*** 12:37–50, 1986.

[11] K. H. Law and D. R. Mackay. A parallel row-oriented sparse solution method for finite element structural analysis. (submitted for publication.) 1992.

[12] J. W. H. Liu. A compact row storage scheme for cholesky factors using elimination trees. ***ACM Tran. Math. Software,*** 12:127–148, 1986.

[13] J. W. H. Liu. A generalized envelope method for sparse factorization by rows. Technical Report CS-88-09, Department of Computer Science, York University, Canada, 1988.

[14] J. W. H. Liu. The role of elimination trees in sparse factorization. ***SIAM J. Matrix Anal. Appl.,*** 11:134–172, 1990.

[15] D. R. Mackay, K. H. Law and A. Raefsky. An implementation of a generalized sparse/profile finite element solution method. ***Computers and Structures,*** 41:723–737, 1991.

[16] B. Nour-Omid, B. N. Parlett and R. L. Taylor. Lanczos versus subspace iteration for solution of eigenvalue problems. ***International Journal for Numerical Methods in Engineering,*** 19:859–871, 1983.

[17] ***C. C.*** Paige. ***The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices.*** PhD thesis, London University, 1971.

[18] B. N. Parlett and D. Scott. The Lanczos algorithm with selective orthogonalization. ***Math. Comp.,*** 33:217–238, 1979.

[19] B. N. Parlett. ***The Symmetric Eigenvalue Problem.*** Prentice Hall, 1980.

[20] Y. Saad. On the rates of convergence of the Lanczos and the block-Lanczos methods. ***SIAM J. Numer. Anal.,*** 17:687–706, 1980.

[21] R. Schreiber. A new implementation of sparse Gaussian elimination. ***ACM Trans. Math. Software,*** 8:256–276, 1982.

[22] H. D. Simon. Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. ***Linear Algebra and Its Applications,*** 61:101–131, 1984.

[23] H. D. Simon. The Lanczos algorithm with partial reorthogonalization. *Math. Comp.*, 42:115-142, 1984.

[24] B.T. Smith, J.M. Boyle, B.S. Garbow, Y. Ikobe, V.C. Klema and C.B. Moler. ***Matrix Eigensystem Routines - EISPACK Guide.*** 2nd edition, Springer Verlag,1976.