SIAM REVIEW
Vol. 34, No. 1, pp. 82–109, March 1992

004

THE MULTIFRONTAL METHOD FOR SPARSE MATRIX SOLUTION: THEORY AND PRACTICE*

JOSEPH W. H. LIU†

Abstract. This paper presents an overview of the multifrontal method for the solution of large sparse symmetric positive definite linear systems. The method is formulated in terms of frontal matrices, update matrices, and an assembly tree. Formal definitions of these notions are given based on the sparse matrix structure. Various advances to the basic method are surveyed. They include the role of matrix reorderings, the use of supernodes, and other implementation techniques. The use of the method in different computational environments is also described.

Key words. sparse matrix, multifrontal method, Cholesky factor, numeric factorization, elimination tree, assembly tree, supernodes

AMS(MOS) subject classifications. 65F50, 65F25

1. Introduction.

The numerical solution of large sparse linear systems lies at the heart of many large-scale scientific and engineering computations. One of the significant advances in direct methods for sparse matrix solution was the development of the *multifrontal method* by Duff and Reid [20] in 1983. The method reorganizes the overall factorization of a sparse matrix into a sequence of partial factorizations of dense smaller matrices.

The method has proved to be extremely valuable. Its importance can be gauged by its wide acceptance in many scientific and engineering applications. The method has been used in many large-scale finite element applications [8], [9], [43]. Recently, its usage has been reported in the solution of the Navier–Stokes equation for computational fluid dynamics [3], in the solution of separable optimization problems [10], and in semiconductor device simulation [40]. The method has also attracted great interest in the linear programming community as an effective solution method for sparse linear least squares problems arising from Karmarkar's method.

The term "multifrontal" is used by Duff and Reid [20] since they developed the method as a generalization of the *frontal* method of Irons [28]. It is, however, interesting to note that the essence of the multifrontal method can be found in the *generalized element* method developed by Speelpenning in 1973 in an unpublished manuscript. The same manuscript was later issued, unmodified, in 1978 as a University of Illinois technical report [46]. Indeed, the element merge model of Eisenstat, Schultz, and Sherman [23] uses this notion of generalized elements and has the same basic features as the multifrontal method.

The main purpose of this paper is to give a uniform presentation of the multifrontal method and to survey the recent advances in its implementation. Most formulations of the multifrontal method in the literature rely heavily on the context of the finite element method [9], [18, pp. 218–225], [20]. The exposition here is purely from the matrix point of view. The frontal matrix, update matrix, and assembly tree are all formally defined in terms of the sparse matrix structure. It is our aim to put the terminologies, notation, and algorithm formulation into a standard framework for future references and development.

*Received by the editors April 16, 1990; accepted for publication (in revised form) May 1, 1991. This research was supported in part by the Natural Sciences and Engineering Research Council of Canada grant A5509.

†Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3.

An outline of the paper follows. In §2, some basic background on sparse Cholesky factorization is given. In particular, the notion of outer-product update is introduced in preparation for the multifrontal method. In §3, we introduce the elimination tree structure [39]. Based on this tree structure, we define formally the sequence of frontal and update matrices. Examples are then used to illustrate these notions.

Section 4 considers the way these frontal and update matrices can be formed. A matrix *extend-add* operator is defined, which is a useful tool in describing the formation of frontal matrices. A new notation "$\dotplus$" is introduced for this purpose. A simple formulation of the sparse Cholesky factorization is then given using the frontal/update matrices. Although we define frontal and update matrices in terms of the elimination tree structure, a more general form of an assembly tree can be used. A new general definition of an assembly tree is given, which can govern the way frontal matrices are to be formed.

Section 5 deals with the issues related to the storage and manipulation of these frontal and update matrices. In order to handle these matrices in an orderly fashion, we consider the use of tree postorderings to find equivalent matrix reorderings. With such reorderings, the fill-reducing quality of the original ordering is preserved, while the frontal and update matrices can now be managed with the use of a stack of full triangular matrices. An algorithm to generate the best postordering in minimizing this stack storage is given. The technique of elimination tree restructuring to further reduce this stack working storage requirement is also discussed.

In §6, we describe the supernodal version of the multifrontal method. The generalization is conceptually simple, and yet it has significant practical advantages. Additional useful implementation techniques of the multifrontal method are discussed in §7. They include full matrix techniques, the use of local/relative indices, relaxed supernodes, and stack storage management.

In §8, we consider the usage of the multifrontal method in various different computational environments: virtual-memory paging systems, limited core-memory systems, and vector and parallel supercomputing machines. No experimental results are given on the performance of the method. Rather, references are provided to papers in the literature that contain actual performance results of the multifrontal method in these variety of system environments. In our concluding remarks in §9, we briefly discuss the use of the method in matrix systems with different numerical properties.

2. Background on Cholesky factorization.

2.1. Factorization by rows, columns, and submatrices. Consider the Cholesky factorization of an n-by-n symmetric positive definite matrix A into LL^t. The factorization can be carried out in many different ways depending on the order in which matrix entries are accessed and/or updated. In [26], three major schemes are identified, which correspond to Cholesky factorization by rows, by columns, and by submatrices. They are appropriately called *row-Cholesky*, *column-Cholesky*, and *submatrix-Cholesky* schemes, respectively.

Factorization by row has also been referred to as the *bordering* scheme. Each step computes a factor row by solving a triangular system. On the other hand, factorization by column computes the factor matrix column by column. Each factor column is computed by applying all updates from previous columns to the matrix column followed by a scaling. Most sparse matrix packages use this form of factorization [22], [27].

In the submatrix-Cholesky scheme, as each factor column is formed, all its updates to the submatrix remaining to be factored are computed. Based on this classification, we can view the multifrontal method as one that performs sparse Cholesky factorization by

submatrices. The novel feature of the multifrontal method is that the update contributions from a factor column to the remaining submatrix are computed, but not applied directly to the matrix entries. They are aggregated with contributions from other factor columns before the actual updates are performed.

2.2. Outer-product updates. In our exposition of the multifrontal method, we use the notion of outer-product updates. We introduce it now for a full matrix. Consider the Cholesky factorization of a *full* n-by-n matrix A into $L\,L^t$, using the submatrix-Cholesky approach. Recall that one step of the outer-product formulation of Cholesky factorization can be expressed as

$$A = \begin{pmatrix} d & v^t \\ v & C \end{pmatrix} = \begin{pmatrix} \sqrt{d} & 0 \\ v/\sqrt{d} & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & C - vv^t/d \end{pmatrix} \begin{pmatrix} \sqrt{d} & v^t/\sqrt{d} \\ 0 & I \end{pmatrix},$$

where d is the first diagonal entry and v an $(n-1)$-vector. The submatrix $C - vv^t/d$ is the portion remaining to be factored, whose factorization can be carried out recursively using the same technique.

This one-step of factorization can be easily generalized to block form. Assume that $j-1$ steps of factorization have been carried out, where $j > 1$. It is convenient to view the partial factorization via a 2-by-2 block partitioning:

$$(1)\quad A = \begin{pmatrix} B & V^t \\ V & C \end{pmatrix} = \begin{pmatrix} L_B & 0 \\ VL_B^{-t} & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & C - VB^{-1}V^t \end{pmatrix} \begin{pmatrix} L_B^t & L_B^{-1}V^t \\ 0 & I \end{pmatrix},$$

where $B = L_B L_B^t$ is the Cholesky factorization of the $(j-1)$-by-$(j-1)$ leading principal submatrix B. The *Schur complement* $C - VB^{-1}V^t$ in (1) represents the part of the matrix remaining to be factored.

Note that the $(n-j+1)$-by-$(n-j+1)$ submatrix $-V\,B^{-1}\,V^t$ represents the update contributions from the first $j-1$ rows and columns to the submatrix C to form $C - VB^{-1}V^t$. This update contribution submatrix can be expressed in terms of the first $j-1$ columns of the Cholesky factor as follows:

$$-V\,B^{-1}\,V^t = -(VL_B^{-t})\,(L_B^{-1}V^t) = -\sum_{k=1}^{j-1} \begin{pmatrix} \ell_{j,k} \\ \vdots \\ \ell_{n,k} \end{pmatrix} \begin{pmatrix} \ell_{j,k} & \cdots & \ell_{n,k} \end{pmatrix}.$$

That is, it is simply the sum of *outer-product updates* from appropriate portions of the $j-1$ columns of the Cholesky factor. The notion of outer-product update forms the basis for the definitions of frontal and update matrices for sparse matrix factorization. Indeed, the multifrontal method can be viewed as providing an effective management of the outer-product updates in the submatrix $-VB^{-1}V^t$ when the matrix is sparse.

3. Frontal and update matrices.

3.1. The elimination tree structure. The key concepts in the multifrontal method are frontal and update matrices. We shall formally define them in terms of the sum of a selected subset of outer-product updates. The selected subset is governed by the structure of a tree, which we now introduce.

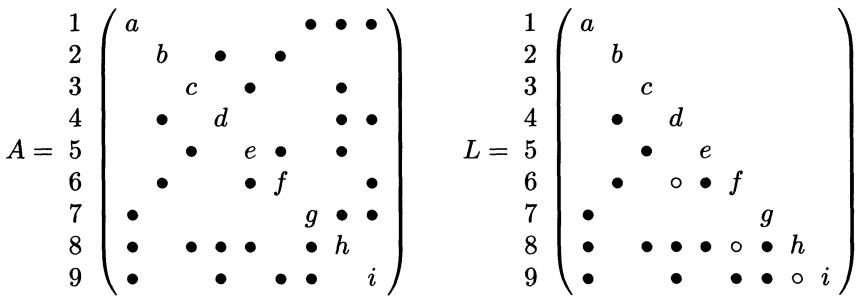

FIG. 3.1. *A sparse matrix example and its Cholesky factor.*

Let A be an n-by-n sparse symmetric positive definite matrix with Cholesky factor L. The *elimination tree* [29], [39], [45] of the matrix A is defined to be the structure with n nodes $\{1, \cdots, n\}$ such that node p is the parent of j if and only if

$$p = \min \{i > j \mid \ell_{ij} \neq 0\}.$$

If A is irreducible, it is easy to verify that this is a tree. But for a reducible matrix, the data structure is a forest. We shall assume throughout this paper that the given matrix A is irreducible, so that the structure is indeed a tree. It should also be noted that we use j to refer to both a column of the matrix and the corresponding node in the elimination tree.

We shall use the notation $T(A)$ to represent the elimination tree of A; and when A is clear from context, T will be used. This tree structure plays an important role in many aspects of sparse matrix computation. A survey of its use has been given by the author in [39]. We shall quote the following relevant properties of the elimination tree. Here, by the structure of a factor column, we mean the set of row subscripts of the nonzeros in the column.

THEOREM 3.1. *If node k is a descendant of j in the elimination tree, then the structure of the vector $(\ell_{jk}, \cdots, \ell_{nk})^t$ is contained in the structure of $(\ell_{jj}, \cdots, \ell_{nj})^t$.*

THEOREM 3.2. [45] *If $\ell_{jk} \neq 0$ and $k < j$, then the node k is a descendant of j in the elimination tree.*

We shall use the notation $T[j]$ to represent the set of descendants of the node j in the elimination tree T including j. In other words, it contains j and the set of nodes in the subtree rooted at the node j. In Fig. 3.1, we have a small sparse matrix example. Each "•" represents an original nonzero in the matrix A, and "○" a fill in the factor matrix L. The row numbers for the matrices are provided, and diagonal entries are labeled by an alphabet for reference. We shall use this example throughout the paper to illustrate various notions.

In Fig. 3.2, we display the elimination tree corresponding to the matrix example of Fig. 3.1. For the subtree rooted at the node 6, we have $T[6] = \{2, 3, 4, 5, 6\}$.

3.2. Definitions of frontal/update matrices. We are now ready to define frontal and update matrices. As before, let A be the given n-by-n matrix and L be its Cholesky factor. Consider the jth column of L. Let $i_0, i_1, \cdots, i_r$ be the row subscripts of nonzeros in L_{*j} with $i_0 = j$; that is, column j has r off-diagonal nonzeros.

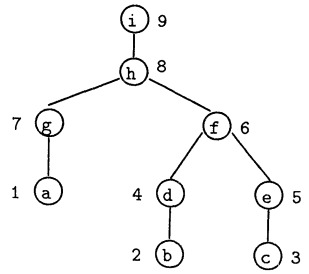

FIG. 3.2. *The elimination tree for the matrix example in Fig.* 3.1.

The *subtree update matrix* at column j for the sparse matrix A is defined to be the matrix

$$\overline{U}_j = - \sum_{k \in T[j]-\{j\}} \begin{pmatrix} \ell_{j,k} \\ \ell_{i_1,k} \\ \vdots \\ \ell_{i_r,k} \end{pmatrix} \begin{pmatrix} \ell_{j,k} & \ell_{i_1,k} & \cdots & \ell_{i_r,k} \end{pmatrix}. \tag{2}$$

That is, it contains outer-product contributions from those preceding columns that are proper descendants of j in the elimination tree. It follows from Theorem 3.1 that if k is a proper descendant of j in the tree, the subscript set of nonzeros in $(\ell_{j,k}, \ell_{j+1,k}, \cdots, \ell_{n,k})^t$ is contained in $\{j, i_1, \cdots, i_r\}$. The nonzero contributions from the outer-product updates of all descendant columns of j are included in $\overline{U}_j$.

The jth *frontal matrix* F_j for A is defined to be

$$F_j = \begin{pmatrix} a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & 0 & \\ a_{i_r,j} & & & \end{pmatrix} + \overline{U}_j. \tag{3}$$

The order $r+1$ of the frontal matrix F_j and the subtree update matrix $\overline{U}_j$ is the number of nonzeros in the jth column of the factor L. The diagonal entry is included in the number of nonzeros.

It is instructive to note that if $k < j$ and $\ell_{jk} \neq 0$, by Theorem 3.2, $k \in T[j] - \{j\}$ so that the outer-product update from column k is included in $\overline{U}_j$ in (2). We can therefore express $\overline{U}_j$ in two components:

$$\overline{U}_j = - \sum_{\substack{k<j \\ \ell_{jk} \neq 0}} \begin{pmatrix} \ell_{j,k} \\ \ell_{i_1,k} \\ \vdots \\ \ell_{i_r,k} \end{pmatrix} \begin{pmatrix} \ell_{j,k} & \ell_{i_1,k} & \cdots & \ell_{i_r,k} \end{pmatrix}$$

$$- \sum_{\substack{k \in T[j] - \{j\} \\ \ell_{jk} = 0}} \begin{pmatrix} 0 \\ \ell_{i_1,k} \\ \vdots \\ \ell_{i_r,k} \end{pmatrix} \begin{pmatrix} 0 & \ell_{i_1,k} & \cdots & \ell_{i_r,k} \end{pmatrix}.$$

It is easy to see that only the first component contributes to the first column of the subtree update matrix $\overline{U}_j$. Indeed, the first column of $\overline{U}_j$ is given by

$$- \sum_{\substack{k < j \\ \ell_{jk} \neq 0}} \ell_{j,k} \begin{pmatrix} \ell_{j,k} \\ \ell_{i_1,k} \\ \vdots \\ \ell_{i_r,k} \end{pmatrix},$$

which contains all the nonzero update entries to column j. This quantity has been referred to as the *complete update column* to column j in [5].

From the definition of F_j in (3), the first row/column of the frontal matrix F_j is formed from A_{*j} and the complete update column to column j. Hence, when F_j is computed, the first row/column of F_j has been completely updated. Therefore, one step of elimination on F_j gives the nonzero entries of the factor column L_{*j}. More specifically, we have

$$F_j = \begin{pmatrix} \ell_{j,j} & 0 \\ \ell_{i_1,j} & \\ \vdots & I \\ \ell_{i_r,j} & \end{pmatrix} \begin{pmatrix} 1 & 0 \\ & \\ & \\ 0 & U_j \end{pmatrix} \begin{pmatrix} \ell_{j,j} & \ell_{i_1,j} & \cdots & \ell_{i_r,j} \\ & & & \\ & & & \\ 0 & & I & \end{pmatrix}.$$

Since $j, i_1, \cdots, i_r$ are the subscripts of nonzeros in L_{*j}, the vector $(\ell_{j,j}, \ell_{i_1,j}, \cdots, \ell_{i_r,j})^t$ is full. This is only possible if the first row/column of the frontal matrix F_j is also full. Moreover, the submatrix remaining to be factored after this one step of elimination on the frontal matrix must also be full. We have used U_j to represent this full submatrix. We shall refer to U_j as the *update matrix* from column j. The next result follows directly from the definitions of F_j and U_j.

THEOREM 3.3.

$$U_j = - \sum_{k \in T[j]} \begin{pmatrix} \ell_{i_1,k} \\ \vdots \\ \ell_{i_r,k} \end{pmatrix} \begin{pmatrix} \ell_{i_1,k} & \cdots & \ell_{i_r,k} \end{pmatrix}. \tag{4}$$

Proof. It follows from the elimination of the first column of F_j that

$$F_j = \begin{pmatrix} \ell_{j,j} \\ \ell_{i_1,j} \\ \vdots \\ \ell_{i_r,j} \end{pmatrix} \begin{pmatrix} \ell_{j,j} & \ell_{i_1,j} & \cdots & \ell_{i_r,j} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ & \\ 0 & U_j \end{pmatrix}.$$

But the submatrix of F_j without the first row/column is the same as the submatrix of $\overline{U}_j$ without the first row/column. Therefore, we have

$$-\sum_{k \in T[j]-\{j\}} \begin{pmatrix} \ell_{i_1,k} \\ \vdots \\ \ell_{i_r,k} \end{pmatrix} \begin{pmatrix} \ell_{i_1,k} & \cdots & \ell_{i_r,k} \end{pmatrix} = \begin{pmatrix} \ell_{i_1,j} \\ \vdots \\ \ell_{i_r,j} \end{pmatrix} \begin{pmatrix} \ell_{i_1,j} & \cdots & \ell_{i_r,j} \end{pmatrix} + U_j.$$

The result then follows. □

It is important to differentiate between $\overline{U}_j$ and U_j. The subtree update matrix $\overline{U}_j$ is used to form the jth frontal matrix F_j; whereas the update matrix U_j is generated from an elimination step with F_j. $\overline{U}_j$ has one more row/column than U_j. The structures of both $\overline{U}_j$ and U_j are based on the column structure of L_{*j}. U_j contains all outer-product contributions from columns in the subtree $T[j]$; while $\overline{U}_j$ contains those from columns in $T[j] - \{j\}$.

3.3. Examples of frontal/update matrices. We shall use the matrix example of Fig. 3.1 to illustrate the notion of frontal and update matrices. It is easy to see that for columns 1, 2, and 3, $\overline{U}_1 = 0$, $\overline{U}_2 = 0$, and $\overline{U}_3 = 0$. Since $T[4] - \{4\} = \{2\}$, the subtree update and frontal matrices for column 4 are given by:

$$\overline{U}_4 = -\begin{pmatrix} \ell_{42} \\ \ell_{62} \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} \ell_{42} & \ell_{62} & 0 & 0 \end{pmatrix},$$

$$F_4 = \begin{pmatrix} a_{44} & 0 & a_{48} & a_{49} \\ 0 & 0 & 0 & 0 \\ a_{84} & 0 & 0 & 0 \\ a_{94} & 0 & 0 & 0 \end{pmatrix} + \overline{U}_4 = \begin{pmatrix} a_{44} - \ell_{42}^2 & -\ell_{42}\ell_{62} & a_{48} & a_{49} \\ -\ell_{42}\ell_{62} & -\ell_{62}^2 & 0 & 0 \\ a_{84} & 0 & 0 & 0 \\ a_{94} & 0 & 0 & 0 \end{pmatrix}.$$

Note that one step of elimination on F_4 will give the factor column L_{*4} and the full update matrix U_4:

$$U_4 = \begin{pmatrix} -\ell_{62}^2 - \ell_{64}^2 & -\ell_{64}\ell_{84} & -\ell_{64}\ell_{94} \\ -\ell_{64}\ell_{84} & -\ell_{84}^2 & -\ell_{84}\ell_{94} \\ -\ell_{64}\ell_{94} & -\ell_{84}\ell_{94} & -\ell_{94}^2 \end{pmatrix}.$$

It is instructive to observe that since $T[4] = \{2, 4\}$, by the result of Theorem 3.3, U_4 can be expressed in terms of the outer-product updates from columns L_{*2} and L_{*4}:

$$U_4 = -\begin{pmatrix} \ell_{62} \\ 0 \\ 0 \end{pmatrix}\begin{pmatrix} \ell_{62} & 0 & 0 \end{pmatrix} - \begin{pmatrix} \ell_{64} \\ \ell_{84} \\ \ell_{94} \end{pmatrix}\begin{pmatrix} \ell_{64} & \ell_{84} & \ell_{94} \end{pmatrix}.$$

The corresponding subtree update and frontal matrices for column 6 are more involved. There are four proper descendants of node 6 in the elimination tree: $T[6]-\{6\} = \{2, 3, 4, 5\}$. They all contribute to the subtree update matrix $\overline{U}_6$.

$$\overline{U}_6 = -\begin{pmatrix} \ell_{62} \\ 0 \\ 0 \end{pmatrix}\begin{pmatrix} \ell_{62} & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 \\ \ell_{83} \\ 0 \end{pmatrix}\begin{pmatrix} 0 & \ell_{83} & 0 \end{pmatrix}$$

$$- \begin{pmatrix} \ell_{64} \\ \ell_{84} \\ \ell_{94} \end{pmatrix}\begin{pmatrix} \ell_{64} & \ell_{84} & \ell_{94} \end{pmatrix} - \begin{pmatrix} \ell_{65} \\ \ell_{85} \\ 0 \end{pmatrix}\begin{pmatrix} \ell_{65} & \ell_{85} & 0 \end{pmatrix},$$

$$F_6 = \begin{pmatrix} a_{66} & 0 & a_{69} \\ 0 & 0 & 0 \\ a_{96} & 0 & 0 \end{pmatrix} + \overline{U}_6$$

$$= \begin{pmatrix} a_{66} - \ell_{62}^2 - \ell_{64}^2 - \ell_{65}^2 & -\ell_{64}\ell_{84} - \ell_{65}\ell_{85} & a_{69} - \ell_{64}\ell_{94} \\ -\ell_{64}\ell_{84} - \ell_{65}\ell_{85} & -\ell_{83}^2 - \ell_{84}^2 - \ell_{85}^2 & -\ell_{84}\ell_{94} \\ a_{69} - \ell_{64}\ell_{94} & -\ell_{84}\ell_{94} & -\ell_{94}^2 \end{pmatrix}.$$

It is interesting to note that $\overline{U}_6$ collects all outer-product contributions from columns associated with its proper descendants. But for column 3, $\ell_{63} = 0$ so that it does not contribute to the first column of the frontal matrix F_6. Yet its contributions to subsequent columns are still collected in the frontal matrix F_6.

One step of elimination on F_6 will give the factor column L_{*6} and the update matrix U_6. Again, by Theorem 3.3, the update matrix U_6 can be expressed in terms of outer-product updates from columns in $T[6] = \{2, 3, 4, 5, 6\}$:

$$U_6 = -\begin{pmatrix} \ell_{83} \\ 0 \end{pmatrix}\begin{pmatrix} \ell_{83} & 0 \end{pmatrix} - \begin{pmatrix} \ell_{84} \\ \ell_{94} \end{pmatrix}\begin{pmatrix} \ell_{84} & \ell_{94} \end{pmatrix} - \begin{pmatrix} \ell_{85} \\ 0 \end{pmatrix}\begin{pmatrix} \ell_{85} & 0 \end{pmatrix} - \begin{pmatrix} \ell_{86} \\ \ell_{96} \end{pmatrix}\begin{pmatrix} \ell_{86} & \ell_{96} \end{pmatrix}.$$

Note that the outer-product update from L_{*2} to U_6 does not appear in this expression since it is zero.

4. Formation of frontal/update matrices.

4.1. Frontal matrix assembly: the matrix extend-add operator. The frontal matrices $\{F_j\}$ and the update matrices $\{\overline{U}_j\}$ and $\{U_j\}$ are central to the formulation of the multifrontal method. Recall that F_j is formed using $\overline{U}_j$, while U_j is obtained from eliminating one row/column of F_j. In this section, we explore the relationship between the set of frontal matrices $\{F_j\}$ and the set of update matrices $\{U_j\}$.

We need one more tool in order to consider the formal relationship between the frontal and update matrices. Let R be an r-by-r matrix with $r \leq n$; and S be an s-by-s matrix with $s \leq n$. Each row/column of R and S corresponds to a row/column of the given n-by-n matrix A. Let $i_1 \leq \cdots \leq i_r$ be the subscripts of R in A, and $j_1 \leq \cdots \leq j_s$ be those of S.

Consider the union of the two subscript sets. Let $k_1 \leq \cdots \leq k_t$ be the resulting union. The matrix R can be *extended* to conform with the subscript set $\{k_1, \cdots, k_t\}$ by introducing a number of zero rows and columns. In a similar way, the matrix S can be extended. We define $R \dotplus S$ to be the t-by-t matrix T formed by adding the two extended matrices of R and S. We shall refer to this matrix operator "$\dotplus$" as the matrix *extend-add* operator. This generalized matrix addition was called matrix *superposition* by Speelpenning [46].

For example, let

$$R = \begin{pmatrix} p & q \\ u & v \end{pmatrix}, \qquad S = \begin{pmatrix} w & x \\ y & z \end{pmatrix}$$

where {5,8} and {5,9} are the subscript sets of R and S, respectively. Then, $R \dotplus S$ is the following 3-by-3 matrix with subscript set {5,8,9}:

$$R \dotplus S = \begin{pmatrix} p & q & 0 \\ u & v & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} w & 0 & x \\ 0 & 0 & 0 \\ y & 0 & z \end{pmatrix} = \begin{pmatrix} p+w & q & x \\ u & v & 0 \\ y & 0 & z \end{pmatrix}.$$

The relation between $\{F_j\}$ and $\{U_j\}$ can be expressed formally in terms of the elimination tree structure and the matrix extend-add operator. Consider column j. As before, let $j, i_1, \cdots, i_r$ be the row subscripts of nonzeros in column L_{*j}.

THEOREM 4.1. *Let nodes $c_1, \cdots, c_s$ be the children of node j in the elimination tree. Then*

$$F_j = \begin{pmatrix} a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & 0 & \\ a_{i_r,j} & & & \end{pmatrix} \dotplus U_{c_1} \dotplus \cdots \dotplus U_{c_s}. \tag{5}$$

Proof. F_j is defined in terms of $\overline{U}_j$ in (3); and from (2), $\overline{U}_j$ is the aggregate of all outer-product updates from columns in $T[j] - \{j\}$. Since $c_1, \cdots, c_s$ are the children of the node j in the elimination tree, $T[j] - \{j\}$ is simply the *disjoint* union of the nodes in the subtrees $T[c_1], \cdots, T[c_s]$. Therefore, we can express $\overline{U}_j$ as an aggregate of outer-product updates from columns in $T[c_1], \cdots, T[c_s]$. But for each $T[c_t]$ $(1 \leq t \leq s)$, by (4)

in Theorem 3.3, its outer-product updates to F_j is given by U_{c_t}. Therefore, all updates from columns in $T[j] - \{j\}$ are included in $U_{c_1}, \cdots, U_{c_s}$. The result then follows. □

It should be noted that the submatrix $U_{c_1} \dotplus \cdots \dotplus U_{c_s}$ in (5) may have fewer rows/columns than F_j. But if it is properly extended to conform with the subscript set of F_j, it becomes the update matrix $\overline{U}_j$. Therefore, the submatrix $U_{c_1} \dotplus \cdots \dotplus U_{c_s}$ contains all the outer-product contributions of $\overline{U}_j$ to F_j and is essentially the same as $\overline{U}_j$. Theorem 4.1 provides this important relation.

This result is the basis of the multifrontal method of Duff and Reid [20]. They refer to the process of forming the jth frontal matrix F_j from A_{*j} and the update matrices of its tree children as the frontal matrix *assembly* operation. Indeed, they call the tree structure, on which the assembly operations are based, the *assembly tree*. There is also an interesting discussion on the formation of frontal matrices in some unpublished lecture notes by Leiserson and Lewis [30] in 1987.

4.2. Formulation of sparse Cholesky factorization. We can now reformulate the Cholesky factorization algorithm in terms of the frontal matrices $\{F_j\}$ and the update matrices $\{U_j\}$.

ALGORITHM 4.1. Cholesky factorization via frontal and update matrices.

for column $j := 1$ **to** n **do**

Let $j, i_1, \cdots, i_r$ be the locations of nonzeros in L_{*j};

Let $c_1, \cdots, c_s$ be the children of j in the elimination tree;

Form the update matrix $\overline{U} = U_{c_1} \dotplus \cdots \dotplus U_{c_s}$;

$$F_j := \begin{pmatrix} a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{i_1,j} & & & \\ \vdots & & & \\ a_{i_r,j} & & & 0 \end{pmatrix} \dotplus \overline{U};$$

Factor F_j into

$$\begin{pmatrix} \ell_{j,j} & 0 \\ \ell_{i_1,j} & \\ \vdots & I \\ \ell_{i_r,j} & \end{pmatrix} \begin{pmatrix} 1 & 0 \\ & \\ & \\ 0 & U_j \end{pmatrix} \begin{pmatrix} \ell_{j,j} & \ell_{i_1,j} & \cdots & \ell_{i_r,j} \\ & & & \\ 0 & & I & \end{pmatrix};$$

end for

Algorithm 4.1 captures the essence of the multifrontal method. Note that the matrix $\overline{U}$ is used only for clarity. Indeed, the frontal matrix F_j can be formed directly from A_{*j} and the children update matrices can be formed by using (5) of Theorem 4.1.

To illustrate the result of Theorem 4.1 in the context of Algorithm 4.1, we consider F_6 for the matrix example of Fig. 3.1. Although there are four contributing columns L_{*2}, L_{*3}, L_{*4}, and L_{*5} to the subtree update matrix $\overline{U}_6$, the node 6 has only two children 4 and 5. Therefore, by Theorem 4.1, F_6 can be formed by using A_{*6} and the update matrices U_4 and U_5. The matrix U_4 contains update contributions to F_6 from columns 2 and 4,

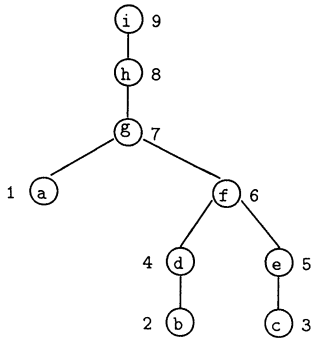

FIG. 4.1. *One possible assembly tree for the matrix example in Fig.* 3.1.

and U_5 from columns 3 and 5. Indeed, we have

$$U_4 = -\begin{pmatrix} \ell_{62} \\ 0 \\ 0 \end{pmatrix}\begin{pmatrix} \ell_{62} & 0 & 0 \end{pmatrix} - \begin{pmatrix} \ell_{64} \\ \ell_{84} \\ \ell_{94} \end{pmatrix}\begin{pmatrix} \ell_{64} & \ell_{84} & \ell_{94} \end{pmatrix},$$

$$U_5 = -\begin{pmatrix} 0 \\ \ell_{83} \end{pmatrix}\begin{pmatrix} 0 & \ell_{83} \end{pmatrix} - \begin{pmatrix} \ell_{65} \\ \ell_{85} \end{pmatrix}\begin{pmatrix} \ell_{65} & \ell_{85} \end{pmatrix}.$$

Since the subscript sets of F_6 and $U_4 \dot{+} U_5$ are the same, it is easy to see that $\overline{U}_6 = U_4 \dot{+} U_5$.

4.3. General assembly trees. The elimination tree is used to guide the assembly of frontal matrices from update matrices in Theorem 4.1. It is interesting to note that an assembly tree for the multifrontal method can take a more general form than that provided by the elimination tree. We can define an assembly tree formally as follows. A tree of the n nodes $\{1, \cdots, n\}$ is an *assembly tree* if for any node j with parent node p, the off-diagonal structure of L_{*j} is a subset of the structure of L_{*p}, and $p > j$.

It is easy to verify that the elimination tree satisfies this condition. But, there are other tree structures that also have this property. For example, if we define the assembly tree such that the node p is the parent of j if and only if

$$p = \min \{i > j \mid \text{off-diagonal structure of } L_{*j} \subseteq \text{ structure of } L_{*i}\}.$$

Taking this new definition, we note that for the matrix example of Fig. 3.1, the parent of node 6 in this assembly tree will be 7 instead of 8 in the elimination tree (see Fig. 3.2). We illustrate this tree in Fig. 4.1.

Any assembly tree can be used to guide the matrix assembly process for the multifrontal method. This is because the structure of the matrix F_j can be used to accommodate all nonzero update entries from columns associated with nodes under j in the tree. Obviously, the subtree update matrix $\overline{U}_j$ and the frontal matrix F_j have to be defined in terms of the assembly tree instead of the elimination tree.

In practice, this general form of assembly tree does not offer much advantage over an elimination tree. We shall consider one possible advantage in §5, when we study the overhead storage requirements. For definiteness, unless otherwise specified, we shall assume that the elimination tree is used for matrix assembly.

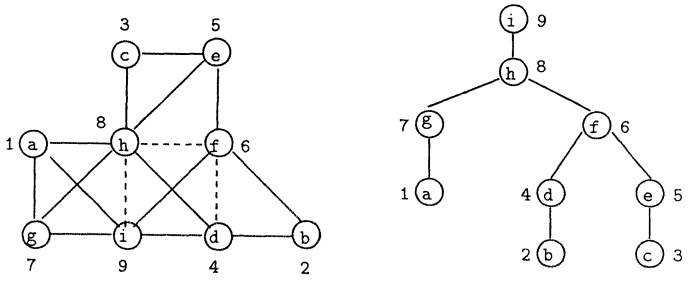

FIG. 4.2. *The undirected graph and elimination tree for the matrix example in Fig.* 3.1.

4.4. Graph-theoretic interpretation. In this section, we will provide graph-theoretic interpretations of the notions of frontal and update matrices. We assume that the readers are familiar with the connection of graph theory and sparse matrices. This subsection can be skipped without affecting the continuity of the paper.

Let G be the undirected graph associated with the given n-by-n symmetric sparse matrix A. Without loss of generality, we assume that G is connected. Consider any connected subset C of nodes in the graph. The *front* of the subset C is defined to be the adjacent set of C in the graph G. We shall use $\mathrm{Adj}_G(C)$ to denote this front.

It is known that for each node j in the elimination tree of the matrix A, the subtree $T[j]$ rooted at j defines a connected subgraph in G [39]. The row/column subscripts of the frontal matrix F_j and the update matrix U_j are given precisely by the sets $\{j\} \cup \mathrm{Adj}_G(T[j])$ and $\mathrm{Adj}_G(T[j])$, respectively.

Take the example in Fig. 4.2, which is the graph associated with the matrix in Fig. 3.1. A solid edge represents a nonzero "•" in the original matrix A, and a dotted edge a fill "○" in the Cholesky factor. We have also included its elimination tree in the figure for convenience. For the node 5, $\mathrm{Adj}_G(T[5]) = \mathrm{Adj}_G(\{3,5\}) = \{6,8\}$, which is the subscript set in the update matrix U_5. Take node 6 with $T[6] = \{2,3,4,5,6\}$. We have $\mathrm{Adj}_G(T[6]) = \{8,9\}$, which is the subscript set of U_6.

The matrix elimination process can be viewed from the graph as proceeding in multiple fronts. In the example of Fig. 4.2, after the elimination of the node 5, we have three subtrees $T[1] = \{1\}$, $T[4] = \{2,4\}$, and $T[5] = \{3,5\}$ in the elimination tree with eliminated nodes. They correspond to three connected subgraphs of eliminated nodes in the graph at this stage of elimination. Their respective adjacent sets are the three fronts in which eliminations have taken place. They are given by $\mathrm{Adj}_G(T[1]) = \{7,8,9\}$, $\mathrm{Adj}_G(T[4]) = \{6,8,9\}$, and $\mathrm{Adj}_G(T[5]) = \{6,8\}$.

Consider the step to form the frontal matrix F_6 via $\overline{U}_6$. By Theorem 4.1, we note that

$$\overline{U}_6 = U_4 \dot{+} U_5.$$

We can interpret this as the merging of the two fronts given by $\mathrm{Adj}_G(T[4])$ and $\mathrm{Adj}_G(T[5])$. The union of these two fronts gives {6,8,9}. Upon eliminating the node 6, this creates a new front with the set {8,9}, which is simply $\mathrm{Adj}_G(T[6])$.

Overall, we can therefore view the elimination as being performed in multiple fronts. An elimination step on a front F_j associated with a node j in the elimination tree leads to the creation of a new front in one of two ways. If j is a leaf node in the elimination tree, the new front $\mathrm{Adj}_G(T[j])$ will be added to the set of fronts. Otherwise, the new front is obtained by the merging of the fronts of all the children of j. The multifrontal method is a novel scheme that reorganizes the numerical computation and has this graph-theoretic interpretation.

5. Storage and manipulation of frontal/update matrices.

5.1. Use of tree postorderings. The basic sparse factorization scheme of Algorithm 4.1 uses frontal and update matrices. Since the columns are processed in increasing order from 1 to n, the update matrices $\{U_j\}$ are also produced in that sequence. If the update matrix U_j is not used immediately in the formation of F_{j+1}, temporary storage space must be allocated to store it so that it can be used for later frontal matrices.

For the matrix example of Fig. 3.1, in step 4 when the frontal matrix F_4 is being processed, it only uses the update matrix U_2. This means we have to store the update matrices U_1 and U_3 produced in previous steps into some temporary locations. After U_4 is generated from F_4, this update matrix has to be kept so that F_5 can be processed. Furthermore, the update matrix U_3 has to be retrieved from the working storage to be used in the formation of F_5.

It is clear from this discussion that an orderly storage and retrieval of the update matrices is necessary for an effective implementation of the multifrontal method. In this section, we consider ways in which we can reorder the matrix to achieve an orderly manipulation of the update matrices and to reduce the amount of working overhead storage required to implement the multifrontal method. The reorderings will be based on the elimination tree.

Let A be a given sparse symmetric positive definite matrix. Two matrix orderings on A are said to be *equivalent* if the reordered matrices have identical filled graphs. It is known that any reordering that numbers a node before its parent in the elimination tree is equivalent to the original ordering [39]. Such an ordering has been referred to as a *topological ordering* of the tree. This therefore provides some flexibility in finding equivalent reorderings that are appropriate for the multifrontal method.

A *postordering* on a tree is a topological ordering such that the nodes in every subtree are numbered consecutively. It follows from definition that a postordering on the elimination tree is equivalent to the original ordering of the given matrix. In the context of the multifrontal method, it has the following additional desirable property: the update matrices can be managed in a *last-in-first-out* basis. Since each update matrix is symmetric and full, we can therefore manipulate the sequence of update matrices using a *stack* data structure [1] of full triangular matrices. When an update matrix is generated from its associated frontal matrix, it will be *pushed* into the stack. On the other hand, when an update matrix is needed in the formation of the current frontal matrix, one such matrix will be *popped* out from the stack of update matrices. This important observation on the use of postordering and stack in the multifrontal method is from the original paper by Duff and Reid [20]. It is also interesting to note that Speelpenning [46] has also used topological sorting in his out-of-core generalized element method; but his motivation is to reduce data I/O traffic.

Consider the matrix example of Fig. 3.1 with its elimination tree in Fig. 3.2. A postordering of this tree is given in Fig. 5.1 together with its reordered matrix and Cholesky factor.

In this example with the new reordering, we can see that the multifrontal method can be performed in an orderly fashion with the use of a stack of update matrices. In Fig. 5.2, we illustrate the contents of this stack for each step of computation. In the display, the box to the left of each F_j represents the contents of the stack before the frontal matrix F_j is processed. Each arrow represents the popping of an update matrix from the stack storage to form the next frontal matrix. For example, the formation of F_7 requires the popping of two update matrices from the stack; whereas F_1, F_3, and F_5 do not require any update matrix from the stack. As the update matrix U_j is formed from F_j, it is being pushed into the stack. The contents of the stack after the push operation are depicted by the box to the right of F_j.

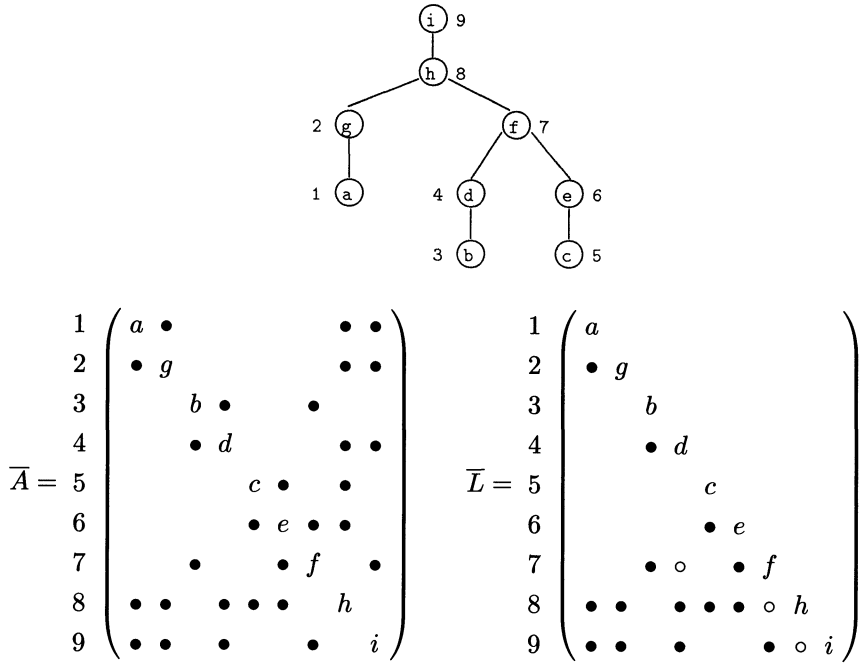

FIG. 5.1. *A postordering and its reordered matrices for the example in Fig.* 3.1.

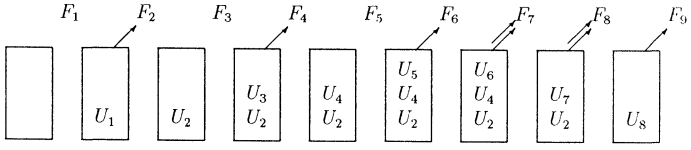

FIG. 5.2. *The stack contents for the postordering in Fig.* 5.1.

5.2. Optimal postorderings for working storage requirement. The use of postordering of the elimination or assembly tree is fundamental in the multifrontal method. Not only does it allow an orderly manipulation of the update matrices, it also has the important property that additional fronts are created only when necessary. In the remainder of this paper, we shall always assume that a postordering is used. In other words, the ordering in the matrix A is already a postordering of some fill-reducing ordering.

It is clear that the execution of the multifrontal method requires additional overhead storage for the stack of update matrices and for the current frontal matrix. The maximum such requirement over the course of the algorithm represents the minimum working storage requirement for the successful completion of the method. In the matrix example of Fig. 5.1, the most amount of working storage needed is during the processing of the frontal matrix F_6. At that step, since the stack update matrices U_2 and U_4 are of sizes 3 and 6, respectively, and the frontal matrix F_6 of size 6, we require a working storage of 15 entries. (Here, we assume that the storage for F_6 overlaps with that for U_5.)

In [33], this author considers the problem of minimizing the working storage requirement for the multifrontal method over the class of postorderings on a given elimination tree. Note that a postordering requires all nodes within a subtree to be numbered consecutively. But for multiple subtrees under the same parent in the elimination tree, we have the freedom of ordering the nodes in one subtree before the others. An op-

timal arrangement of the children in the elimination tree to minimize working storage requirement is given in [33]. We shall express the result in the terms used here.

We shall use the notation $|F_j|$ and $|U_j|$ to represent the number of nonzeros in the lower triangular part of the frontal matrix F_j and the update matrix U_j, respectively (that is, the number of storage locations). For a node j in the elimination tree, define $MinWstore(j)$ to be the minimum working storage required to execute the multifrontal method on the nodes/columns associated with the subtree $T[j]$. It is clear that if j is a leaf node in the elimination tree,

$$MinWstore(j) = |F_j|.$$

If j is not a leaf, let $c_1, \cdots, c_s$ be the children of j in the tree.

THEOREM 5.1. [33] *An arrangement of the subtrees $T[c_1], \cdots, T[c_s]$ under the node j in decreasing order of*

$$\max\{MinWstore(c_1), |F_j|\} - |U_{c_1}| \geq \cdots \geq \max\{MinWstore(c_s), |F_j|\} - |U_{c_s}|,$$

is an optimal (in terms of storage requirement) children sequence for the node j and

$$MinWstore(j) = \max_{1 \leq k \leq s} \left\{ \max\{MinWstore(c_k), |F_j|\} + \sum_{i=1}^{k-1} |U_{c_i}| \right\}. \tag{6}$$

It should be pointed out that in Theorem 5.1, in the formation of the frontal matrix F_j from A_{*j} and the update matrices $U_{c_1}, \cdots, U_{c_s}$, we assume that the storage used by F_j overlaps with that used by the last update matrix U_{c_s}. A simple algorithm can be devised based on the above theorem to determine an optimal postordering for the elimination tree that requires the least amount of working storage for the execution of the multifrontal method. In this algorithm, the nodes are processed in increasing order from 1 to n. Since the original ordering is already a topological ordering of the elimination tree, the $MinWstore$ values of children are always determined before that of the parent value in the following algorithm.

ALGORITHM 5.1. Optimal postordering.

for $j := 1$ **to** n **do**

  if j is a leaf in the elimination tree

    then $MinWstore(j) = |F_j|$;

    else

    Rearrange the children $c_1, \cdots, c_s$ of j in descending order of

      $$\max\{MinWstore(c_k), |F_j|\} - |U_{c_k}|;$$

    Compute $MinWstore(j)$ using equation (6) of Theorem 5.1

  end if;

end for;

Generate the new postordering based on the new children sequences.

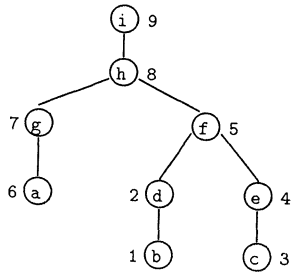

FIG. 5.3. *An optimal postordering for the elimination tree in Fig.* 5.1.

The sorting of the children is the most time-consuming step in Algorithm 5.1. Since in practice, the number of children of most nodes in an elimination tree is bounded by two, an optimal postordering can be determined in time proportional to the size of the matrix.

It is instructive to apply this algorithm to the postordering in Fig. 5.1. It is straightforward to verify that

$$\max\{MinWstore(4), |F_7|\} - |U_4| = \max\{10, 6\} - 6 = 4;$$

$$\max\{MinWstore(6), |F_7|\} - |U_6| = \max\{6, 6\} - 3 = 3;$$

so that the two subtrees under the node 7 are already in proper sequence. Also, by Theorem 5.1, we have $MinWstore(7) = 12$. However, for the two subtrees $T[2]$ and $T[7]$ under the node 8,

$$\max\{MinWstore(2), |F_8|\} - |U_2| = \max\{10, 3\} - 3 = 7;$$

$$\max\{MinWstore(7), |F_8|\} - |U_7| = \max\{12, 3\} - 3 = 9.$$

Therefore, the nodes in $T[7]$ should be numbered before those in $T[2]$ for an optimal postordering.

Based on Algorithm 5.1, we can therefore generate an optimal postordering for the given elimination tree to minimize the working storage requirement for the multifrontal method. We display in Fig. 5.3 such an optimal postordering. For this new postordering, the minimum working storage requirement can be determined to be 13 (instead of 15 for the postordering in Fig. 5.1). The readers are referred to [33] for experimental results on some practical problems with more substantial savings in working storage. Over 50 percent reduction in working storage has been reported. This can have significant impact on the out-of-core version of the multifrontal method (see §8.2).

5.3. Further reduction in working storage by tree restructuring. For a given elimination tree, out of the class of all postorderings on this tree, Algorithm 5.1 determines an optimal postordering that minimizes the working storage requirement for the execution of the multifrontal method. The structure of the elimination tree is preserved by the new postordering.

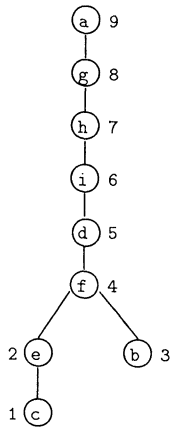

FIG. 5.4. *A different elimination tree of an equivalent reordering for Fig.* 4.2.

Further reduction in the working storage is possible if we allow the structure of the tree to alter. For example, if we number the nodes in the graph of Fig. 4.2 in the following order:

$$c, e, b, f, d, i, h, g, a,$$

it is easy to verify that this is an ordering equivalent to the one in Fig. 4.2 (that is, the set of filled edges remains unchanged). However, the structure of the elimination tree is quite different; we illustrate it in Fig. 5.4. Furthermore, the amount of working storage requirement for the multifrontal method using this equivalent ordering is 10.

In [36], the author considers the use of elimination tree *rotations* that will produce an equivalent matrix reordering with a different elimination tree structure. In particular, the original tree structure can be transformed to an unbalanced form, which is more desirable for the multifrontal method in reducing working storage. The readers are referred to [36] for the details in elimination tree rotations and their practical effect in reducing working storage.

In §4.3, we consider a formal definition of an assembly tree for the multifrontal method. This form is more general than an elimination tree. We note that for the multifrontal method, the use of a different assembly tree structure sometimes requires less working storage when compared with the standard elimination tree. We shall not pursue this further since the elimination tree is mainly used in practice for the multifrontal assembly. Instead, we simply provide an example.

Consider the star graph, where the center node is numbered last. It is easy to verify that the tree consisting of a chain of nodes, numbered in increasing order from the leaf to the root, satisfies the assembly tree requirement. This is certainly different from its elimination tree and it requires less working storage in the multifrontal method than the elimination tree.

6. Supernodal multifrontal methods.

6.1. Overview of the multifrontal method. Conceptually, the multifrontal method reorganizes the computation associated with sparse Cholesky factorization. It transforms the task of factoring a large sparse matrix into a sequence of subtasks, each involving the partial factorization of a *smaller dense* frontal matrix:

$$F_1 \Rightarrow L_{*1}, \quad F_2 \Rightarrow L_{*2}, \cdots, \quad F_n \Rightarrow L_{*n}.$$

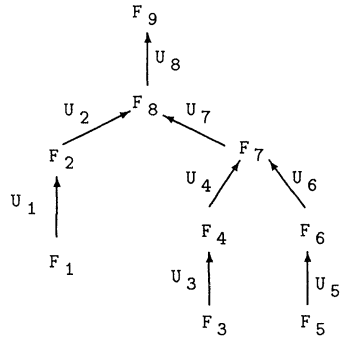

FIG. 6.1. *The assembly/factor tree structure for the matrix example in Fig.* 5.1.

In the last two sections, we have provided preprocessing details before the numerical factorization in the multifrontal method. The structural preprocessing may consist of the following major steps: (1) determine the elimination tree structure of the given matrix; (2) restructure the tree to reduce working storage requirement by finding an equivalent reordering; (3) determine an optimal postordering of the restructured tree (Algorithm 5.1); (4) form the new elimination tree and perform the symbolic factorization based on this optimal postordering.

After the preprocessing, the computation of the factor matrix by the multifrontal method can be carried out using Algorithm 4.1 as the basis. The elimination tree provides the assembly information to form each frontal matrix F_j, which can be computed from A_{*j} and the update matrices using (5) of Theorem 4.1. Indeed, in the elimination tree, if each node j is replaced by its frontal matrix F_j, and the edge from j to its parent is labeled by the update matrix U_j, the entire assembly/factor process in the multifrontal method can be conveniently represented by the resulting tree structure. The assembly/factor structure for the matrix example of Fig. 5.1 is illustrated in Fig. 6.1.

A stack of full triangular matrices is used to manipulate the update matrices in an orderly fashion. We shall provide an expanded version of Algorithm 4.1 in the next section when we include the use of supernodes in the description.

6.2. Supernodal version. A practical improvement to the multifrontal method is the use of *supernodes* [4], [8], [20] or their variants. The notion of supernode appears in many other contexts of sparse matrix computation. It has been referred to in the literature as *supervariable* [20] and *indistinguishable node* [27].

Generally speaking, columns/nodes are grouped together to form a supernode if they can be treated as one computational unit in the course of sparse Cholesky factorization. We now provide a formal definition of this notion in terms of the elimination tree. A *supernode* is a maximal set of contiguous nodes $\{j, j+1, \cdots, j+t\}$ such that

$$\mathrm{Adj}_G(T[j]) = \{j+1, \cdots, j+t\} \cup \mathrm{Adj}_G(T[j+t]).$$

It follows from this definition that in a supernode $\{j, j+1, \cdots, j+t\}$, for $1 \le k \le t$, the node $j+k$ is the parent of $j+k-1$ in the elimination tree.

In matrix terms, $\mathrm{Adj}_G(T[j])$ gives the subscript set of nonzeros in column L_{*j}. Therefore, a supernode corresponds to a maximal block of contiguous columns in the Cholesky factor, where the corresponding diagonal block is full triangular, and these columns all

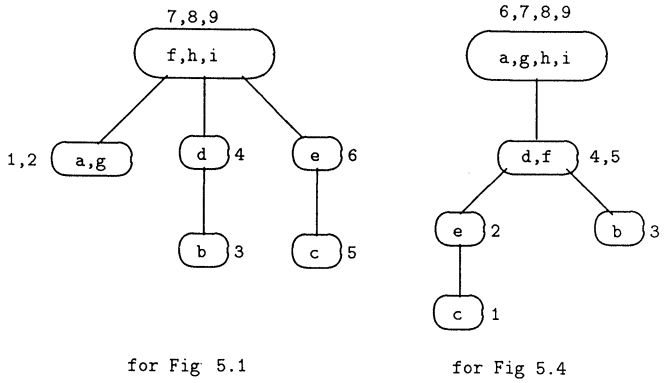

FIG. 6.2. *The supernodal elimination trees for the examples in Figs.* 5.1 *and* 5.4.

have identical off-block-diagonal column structures. For the matrix example in Fig. 5.1, it is easy to see that with the given ordering, we obtain six supernodes, given by:

$$\{1,2\},\{3\},\{4\},\{5\},\{6\},\{7,8,9\}.$$

It should be noted that a slight variant of supernode can be defined without using the contiguous condition. However, we keep it for its simple matrix interpretation. Furthermore, most practical use of supernodes as storage or computational units will require this contiguous condition. With our definition, the set of supernodes depends crucially on the ordering of the matrix/graph due to the contiguous condition. Postorderings on the elimination tree will help in producing larger supernodes.

Consider a given set of supernodes. The *supernodal elimination tree* is defined to be the tree consisting of the supernodes, and $\tilde{S}$ is the parent supernode of $S = \{j, \cdots, j+t\}$ if the parent p of $j+t$ in the (nodal) elimination tree belongs to $\tilde{S}$. (The notion of supernodal assembly tree can be similarly defined.) In Fig. 6.2, we display the supernodal elimination trees for the examples in Figs. 5.1 and 5.4.

We can also generalize the notion of a frontal matrix in (3) to a supernodal frontal matrix. Let $S = \{j, j+1, \cdots, j+t\}$ be a supernode, and $j+t, i_1, \cdots, i_r$ be the row subscripts of nonzeros in L_{*j+t}, the last column of S. The *supernodal frontal matrix* $\mathcal{F}_j$ associated with S is defined to be

$$\mathcal{F}_j = \begin{pmatrix} a_{j,j} & a_{j,j+1} & \cdots & a_{j,j+t} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{j+1,j} & a_{j+1,j+1} & \cdots & a_{j+1,j+t} & a_{j+1,i_1} & \cdots & a_{j+1,i_r} \\ \vdots & \vdots & \cdots & \vdots & & & \\ a_{j+t,j} & a_{j+t,j+1} & \cdots & a_{j+t,j+t} & a_{j+t,i_1} & \cdots & a_{j+t,i_r} \\ a_{i_1,j} & a_{i_1,j+1} & \cdots & a_{i_1,j+t} & & & \\ \vdots & \vdots & \cdots & \vdots & & 0 & \\ a_{i_r,j} & a_{i_r,j+1} & \cdots & a_{i_r,j+t} & & & \end{pmatrix} + \overline{\mathcal{U}}_j,$$

where

$$\overline{U}_j = - \sum_{\substack{k \in T[j+t] \\ -\{j,\cdots,j+t\}}} \begin{pmatrix} \ell_{j,k} \\ \ell_{j+1,k} \\ \vdots \\ \ell_{j+t,k} \\ \ell_{i_1,k} \\ \vdots \\ \ell_{i_r,k} \end{pmatrix} \begin{pmatrix} \ell_{j,k} & \ell_{j+1,k} & \cdots & \ell_{j+t,k} & \ell_{i_1,k} & \cdots & \ell_{i_r,k} \end{pmatrix}.$$

A supernodal version of Theorem 4.1 can be established, relating the supernodal frontal matrix $\mathcal{F}_j$ and the update matrices. In the matrix extend-add formation of $\mathcal{F}_j$, we include all the update matrices U_c, where c is the highest-numbered node of each child supernode of $\{j, j+1, \cdots, j+t\}$. This provides the basis for supernodal frontal matrix assembly. For example, in Fig. 5.1, to form $\mathcal{F}_7$ for the supernode {7,8,9}, we need the update matrices U_2, U_4, and U_6.

In this context, it is important to realize that for a supernode $S = \{j, j+1, \cdots, j+t\}$, the frontal matrix F_j and the supernodal frontal matrix $\mathcal{F}_j$ both have the same size and subscript sets. Therefore, in the space allocated for F_j, we can actually assemble $\mathcal{F}_j$. Furthermore, once $\mathcal{F}_j$ is formed, its first $t+1$ columns have all been completely updated from columns before j, and hence can be eliminated together as a unit.

We shall now formulate a supernodal version of the multifrontal method. It is an expanded version of the basic Algorithm 4.1 with the incorporation of the stack and the use of supernodes. Here, we assume that the ordering is the final postordering of the restructured tree from the preprocessing step, and that the supernodes are defined in terms of this ordering.

ALGORITHM 6.1. Cholesky factorization by the supernodal multifrontal method.

Initialize a stack of full triangular matrices;

for each supernode S in increasing order of first column subscript **do**

Let $S = \{j, j+1, \cdots, j+t\}$;

Let $j+t, i_1, \cdots, i_r$ be the locations of nonzeros in $L_{*,j+t}$;

$$\mathcal{F}_j = \begin{pmatrix} a_{j,j} & a_{j,j+1} & \cdots & a_{j,j+t} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{j+1,j} & a_{j+1,j+1} & \cdots & a_{j+1,j+t} & a_{j+1,i_1} & \cdots & a_{j+1,i_r} \\ \vdots & \vdots & \cdots & \vdots & & & \\ a_{j+t,j} & a_{j+t,j+1} & \cdots & a_{j+t,j+t} & a_{j+t,i_1} & \cdots & a_{j+t,i_r} \\ a_{i_1,j} & a_{i_1,j+1} & \cdots & a_{i_1,j+t} & & & \\ \vdots & \vdots & \cdots & \vdots & & 0 & \\ a_{i_r,j} & a_{i_r,j+1} & \cdots & a_{i_r,j+t} & & & \end{pmatrix};$$

nchildren := no. of children of S in supernodal elimination tree;

for $c := 1$ **to** *nchildren* **do**

Pop an update matrix U from the stack and form $\mathcal{F}_j := \mathcal{F}_j \dotplus U$

end for;

Perform $t+1$ steps of elimination on $\mathcal{F}_j$ to give

the factor columns $L_{*j}, \cdots, L_{*,j+t}$ and U_{j+t};

Push the update matrix U_{j+t} into the stack

end for;

Implicit in this formulation is the assumption that when an update matrix U is popped out of the stack, we have to know its structure (that is, the subscript set) so that the matrix extend-add operation can be performed to merge it with $\mathcal{F}_j$. It can be achieved by simply remembering the column that generates this update matrix.

The structural preprocessing steps of §6.1 must now be formulated in terms of a supernodal elimination tree. In particular, the optimal postordering result of Theorem 5.1 has to be modified to account for the use of supernodes. In [4], Ashcraft provides the generalization of Theorem 5.1 for the supernodal version. The formula for minimum working storage requirement in the theorem can also be expressed recursively based on the supernodal elimination tree.

It should be pointed out that the supernodal elimination tree is closely related to a tree structure, called the *clique tree* [31], [42]. Clique trees can be used to determine supernodes and to guide the assembly process in the formation of supernodal frontal matrices. Interested readers are referred to [31], [42] for more details on clique trees.

7. Implementation techniques for the multifrontal method. In this section, we attempt to identify various implementation techniques proposed or used in the literature for the multifrontal method. We shall restrict ourselves to techniques related to the numerical factorization phase, and not the structural part. A very thorough description of a supernodal multifrontal implementation has been given by Ashcraft [4].

7.1. Full matrix techniques. Except for the matrix extend-add operations, the entire numerical computation of the multifrontal method can be carried out in full matrix form. This is the key to the success of the method, and was first recognized by Duff and Reid [14], [20].

Once the frontal or supernodal frontal matrix is formed, the elimination step can be performed as a full matrix elimination. This has the significant advantage of reducing the indexing overhead required for sparse elimination. Furthermore, standard techniques used in dense matrix factorization, such as loop unrolling [12], can be employed.

It is also important to point out that for many practical problems, a good percentage of the factorization operations in the multifrontal method is associated with the partial factorization of the last few full matrices. For example, consider the k-by-k regular model problem with nine-point difference operator ordered by the nested dissection ordering [25]. The total number of arithmetic operations for factorization is bounded by $829k^3/84$ [27]. If the multifrontal method is used to factor the corresponding matrix problem, it is easy to verify that the last 6 supernodal frontal matrices require approximately $61k^3/24$ multiplicative operations for their partial elimination. In other words, over 25 percent of the operations are performed in the partial factorization of the last 6 full supernodal frontal matrices.

It is fair to say that for most practical sparse matrix problems, the overall performance of the multifrontal method depends crucially on the code for small full matrix partial factorization. In [4], Ashcraft provides some interesting statistics on the percentage of full matrix operations in the multifrontal method for a number of practical sparse problems.

7.2. Local indices. The matrix extend-add operator is required in the formation of the frontal matrices, as given by Theorem 4.1. Each child update matrix has to be extended and then added to the current frontal matrix. These operations have also been referred to as the *sparse operations*. To facilitate such operations, the use of *local indices* has been suggested by Ashcraft [4]. The notion of local indices is first described by Schreiber [45] when he introduces a relative indexing storage scheme for sparse Cholesky factors.

Instead of storing the explicit row subscripts of nonzeros in the factor column, the locations of its nonzeros relative to the frontal matrix of its parent node in the elimination tree are stored. In this way, nonzeros from the update matrix can be added directly into the frontal matrix without having to perform overhead operations involving the explicit row subscripts.

For example, in Fig. 5.1, the locations of off-diagonal nonzeros in column 5 of the factor matrix are given by the row indices 6 and 8. But they correspond to rows 1 and 3, respectively, of the parent frontal matrix F_6. Instead of keeping the explicit indices 6 and 8, the relative indices of 1 and 3 are stored. In this way, the formation of F_6 from U_5 can be performed using only the relative indices.

The sizes of these local indices will be no greater than the size of the largest frontal matrix (that is, the maximum number of off-diagonal nonzeros in the factor columns). Often, they can be stored using short integers. Thus, the use of local indices has the potential advantage of saving overhead storage if short integers are to be employed.

7.3. Relaxed supernodes. The notion of supernode, described in §6.2, groups a block of contiguous columns as one computational unit. These columns can be eliminated together from the corresponding dense supernodal frontal matrix. There are computational advantages in having large supernodes, especially in vectorization of the multifrontal method. Duff and Reid [20] suggest the amalgamation of nodes/supernodes to allow for larger supernodes. Some logical zeros are treated as nonzeros thereby increasing the sizes of some supernodes.

In [7], Ashcraft and Grimes provide an algorithm based on the elimination tree structure for finding supernodes to enhance the performance of the multifrontal method. They have used the term *relaxed supernodes* to refer to such extension of supernodes.

For example, in Fig. 5.1, by treating the zero entry ℓ_{75} as nonzero, we can group $\{5, 6\}$ as a relaxed supernode. In some situations, the advantages offered by having larger supernodes can more than offset the extra arithmetic operations required to be performed on the logical zeros. The readers are referred to [7] for such experimental results.

7.4. Storage of supernodal frontal matrices. In §5.2, we consider the amount of working storage required for the successful execution of the multifrontal method. We assume implicitly that working storage for the entire frontal matrix F_j is required during its formation and partial elimination. However, it should be clear that the first column of this frontal matrix F_j can overlap with the storage of the corresponding factor column L_{*j}. In other words, we only need the working storage of size given by the update matrix U_j during the processing of F_j.

Of course, the amount of working storage saved is rather minimal for the basic multifrontal scheme. However, for the supernodal multifrontal method, we can take advantage of the entire block of supernode columns. Let $\mathcal{F}_j$ be the supernodal frontal matrix associated with the supernode $\{j, j+1, \cdots, j+t\}$. We can split the storage of the supernodal frontal matrix $\mathcal{F}_j$ into two portions. The supernode columns in the supernodal frontal matrix will be stored in their respective factor columns $L_{*j}, L_{*,j+1}, \cdots, L_{*,j+t}$; whereas the remaining columns will be kept in a working array of size equal to that of the update matrix U_{j+t}.

This storage scheme also reduces the amount of data movement. After the partial elimination of the supernodal frontal matrix, the factor columns of the supernode will be already in place so that further data movement is not necessary. The management of the working stack storage can also be simplified by using this storage scheme. After the partial elimination of the supernode columns from the supernodal frontal matrix $\mathcal{F}_j$, the remaining full update matrix U_{j+t} will be residing on the "top" of the stack storage area. Again the amount of data movement can be reduced.

8. Multifrontal method in different computational environments.

8.1. Virtual-memory paging systems. The appropriateness of the multifrontal method for paged virtual-memory system is discussed by the author in [38]. In the multifrontal method, contributions from columns associated with the subtree rooted at the node j are all aggregated into the update matrix U_j of column j. Subsequent access of these column update information will be localized in the storage space for U_j. Therefore, good data locality behavior is exhibited in the multifrontal method.

On the other hand, in the conventional sparse Cholesky factorization algorithm by columns (see, for example, [18], [27]), updates to column j are applied together. This implies that the factor columns that affect column j need to be accessed to compute their contributions to column j. The required columns are usually not consecutive and are hence scattered throughout the data storage vector for L. The execution of a sparse column-Cholesky factorization code in a paging environment incurs a significant amount of page faults.

In [38], a tenfold reduction in the number of page faults is reported for the multifrontal method over the conventional column-Cholesky scheme. This can result in as much as a 20 percent reduction in factorization time. This good data locality property of the multifrontal method is also important for sparse matrix factorization on machines with cache memory [44].

8.2. Limited core-memory systems. The frontal method was originally proposed by Irons [28] as an efficient computational scheme on machines with limited core memory. It is not surprising to see that the multifrontal method also adapts well in such environments.

As noted before, in the multifrontal method, contributions from columns associated with the subtree rooted at the node j are all aggregated into the update matrix U_j of column j. This implies that it is no longer necessary to access all the factor columns associated with this subtree. This forms the basis for the out-of-core multifrontal method.

In the supernodal multifrontal method of Algorithm 6.1, if we output the factor columns as they are being computed to an external file, and we only maintain the working stack storage in core, we have a simple basic out-of-core multifrontal scheme. In such a case, the amount of in-core storage requirement for the successful completion of the out-of-core scheme will be given by the amount of working storage required by the in-core multifrontal method. It should be noted that for a large class of practical problems,

this working storage requirement is usually a modest amount compared to the number of nonzeros in the factor matrix. As little as ten percent of the number of factor nonzeros for some practical matrix systems has been reported in the literature [38].

It is interesting to note that the original formulation of the generalized element method by Speelpenning [46] is described as an out-of-core solution method. At a given stage of the algorithm, the pool of *generalized elements* which is simply the set of update matrices generated, will be stored out-of-core. Each generalized element or update matrix will be brought to memory when it is required to form a new generalized element or a new frontal matrix. The use of topological sorting is also suggested to reduce the amount of data traffic.

The TREESOLVE package by Reid [43] is an out-of-core version of the multifrontal method designed to solve huge finite-element problems. The package also allows portion of the working stack storage to be out-of-core when necessary.

In [24], Eisenstat, Schultz, and Sherman propose an interesting variant of sparse Gaussian elimination for machines with limited core; and no auxiliary storage is required. They have called it the *minimal storage sparse elimination* algorithm, which can be best described in terms of the multifrontal method. Except for the last supernodal frontal matrix, factor columns generated from elimination of frontal matrices are discarded as they are computed. When the last supernodal frontal matrix is fully assembled, its associated unknowns are solved by a full matrix factorization and forward/backward substitution. Using these computed components, the overall sparse linear system can then be reduced to two or more smaller independent linear systems. The method can be applied recursively until all unknowns are computed.

It should be noted that the amount of core storage required for this minimal storage scheme is the same as that required by the basic out-of-core multifrontal method. This is usually significantly smaller than the storage requirement for the factor matrix. For more details, the readers are referred to [24] and [39, p. 164].

8.3. Vector machines. The use of the multifrontal method on vector machines has been considered by many researchers. Its important use for such machines has been recognized in the original work of Duff and Reid [20]. The actual performance of multifrontal codes on various generations of Cray machines can be found in [4], [8], [11], [15], [19].

As noted before, the multifrontal method reorganizes the entire sparse matrix factorization into a sequence of partial factorization on smaller dense submatrices. Since vector machines are most suitable for full vector operations, this computational reorganization helps to explain the success of the multifrontal method for these machines.

Ashcraft [4] gives a very detailed description of a vector implementation of the supernodal multifrontal method. In particular, loop unrolling techniques to improve the *multiple-vector* to *multiple-vector* operations [13] in the factor and update portions of the supernodal codes are described. Computational speeds of over 100 megaflops on an Cray X-MP have been achieved for his vectorized supernodal multifrontal code on some practical three-dimensional finite element problems. In [4], Ashcraft also provides many interesting performance statistics and valuable implementation details.

8.4. Parallel machines. The notion of multiple independent fronts in the multifrontal method provides a natural framework for parallel computations. Broadly speaking, there are two levels of parallelism that can be explored in the numerical factorization phase. The first is the processing of the frontal/update matrices associated with independent fronts. Such parallelism is easily identifiable using the elimination tree structure.

Indeed, the processing of columns associated with any two nonoverlapping subtrees can be treated as completely independent tasks. For example, in Fig. 5.1, the three subtrees $T[2]$, $T[4]$, and $T[6]$ correspond to three independent fronts. The processing of the three sets $\{F_1, F_2\}$, $\{F_3, F_4\}$, and $\{F_5, F_6\}$, and their corresponding update matrices can therefore proceed in parallel.

The second level of parallelism comes from the processing of the individual (supernodal) frontal and update matrix. Since the amount of computation involved in the assembly and partial elimination of a frontal matrix can be quite substantial, further subdivision of the task into smaller subtasks seems appropriate.

It is easy to formulate concurrent versions of the multifrontal method for parallel machines. Indeed, the method can be adapted to both shared-memory and local-memory machine architectures. The exploration of the multifrontal method on these high-performance supercomputers is currently a very active research area.

Implementation of the multifrontal method on shared-memory machines can be found in the work of [2] (Cray-2), [9] (ELXSI, Cray-XMP), [16] (HEP), [17] (Alliant FX/8). Amestoy and Duff [2] have used the level-3 BLAS (Basic Linear Algebra Subprograms) kernels in the elimination operations of frontal matrices. They have obtained over 200 megaflops on a CRAY-2 for the multifrontal factorization of some structural problems. Benner, Montry, and Weigand [9] have considered a coarse-grained parallel multifrontal approach for the finite-element method.

In [17], Duff has used finer granularity by grouping the work associated with a block of rows/columns in the frontal/update matrices as a task. Therefore, a number of tasks will be spawned in the assembly and partial elimination of one frontal matrix. He has also suggested the use of a *buddy system* to allocate and manage working storage for the frontal and update matrices.

The implementation of the multifrontal method on local-memory systems has been considered by many researchers. Lucas, in his thesis [40] and [41], has given a detailed discussion of his distributed multifrontal code, and provided performance statistics of his code on an Intel iPSC hypercube. He has also considered the application of the parallel multifrontal code to a numerical device simulation package. The performance of a distributed multifrontal code by Ashcraft on an Intel hypercube has been reported in [6].

9. Concluding remarks. In this paper, we have presented the multifrontal method as a solution scheme for large sparse positive definite symmetric linear systems. The method organizes the numerical factorization into a number of steps, each involving the formation of a dense smaller frontal matrix followed by its partial factorization. There are several advantages of this approach over other factorization algorithms: effective vector processing on dense frontal matrices, better data locality, efficient out-of-core scheme, and its natural adaptability to parallel computations. The price paid is the extra working storage required for the frontal matrices and the extra arithmetic needed to perform the assembly operations.

The use of the multifrontal method on other types of sparse systems has also been reported in the literature. Interestingly, the original motivation of Duff and Reid [20] to design the multifrontal method was for the solution of sparse symmetric indefinite systems. For indefinite systems, it is necessary to pivot for numerical stability. Duff and Reid allow the search of suitable 1-by-1 and 2-by-2 pivots within the fully-formed portions of the frontal matrix. If no appropriate pivot can be found, the current frontal matrix is allowed to be extended to include more columns. In this way, the fully-formed

portions will be enlarged to include more possible pivot candidates. The scheme is very effective as a practical solution method for symmetric indefinite systems.

In [34] and [35], Liu provides some observations on the numerical threshold pivoting strategies for the multifrontal method applied to symmetric indefinite systems. The notion of *delayed elimination* in the choice of suitable numerical block pivots for the method by Duff and Reid [20] is discussed in terms of elimination tree transformations by the author in [37].

In [21], Duff and Reid have also explored the use of the multifrontal method for the solution of sparse unsymmetric linear systems. Only limited success has been reported for such systems.

In [32], this author has introduced the notion of a *row merge tree* for sparse orthogonal factorization of a rectangular matrix M by Givens rotations. This row merging scheme can actually be viewed as the use of the multifrontal method on $A = M^t M$ without ever forming A. The readers are referred to [39, pp. 165–166] for a discussion of the connection between the elimination tree, row merging, and the multifrontal method.

Acknowledgments. The author thanks the referee for many constructive criticisms which contributed greatly in improving the presentation of this paper. He is also grateful to John Gilbert and Phuong Vu for many useful suggestions.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *Data Structures and Algorithms*, Addison–Wesley, Reading, MA, 1983.

[2] P. Amestoy and I. Duff, *Vectorization of a multiprocessor multifrontal code*, Internat. J. Supercomput. Appl., 3 (1989), pp. 41–59.

[3] P. Amestoy and R. Tilch, *Solving the compressible Navier–Stokes equations with finite elements using a multifrontal method*, Impact of Computing in Science and Engineering, 1 (1989), pp. 93–107.

[4] C. Ashcraft, *A vector implementation of the multifrontal method for large sparse, symmetric positive definite linear systems*, Tech. Report ETA-TR-51, Engineering Technology Applications Division, Boeing Computer Services, Seattle, WA, 1987.

[5] C. Ashcraft, S. Eisenstat and J. Liu, *A fan-in algorithm for distributed sparse numerical factorization*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 593–599.

[6] C. Ashcraft S. Eisenstat, J. Liu and A. Sherman, *The comparison of three column-based distributed sparse factorization schemes*, Tech. Report YALEU/DCS/RR-810, Department of Computer Science, Yale University, New Haven, CT, 1990.

[7] C. Ashcraft and R. Grimes, *The influence of relaxed supernode partitions on the multifrontal method*, ACM Trans. Math. Software, 15 (1989), pp. 291–309.

[8] C. Ashcraft, R. Grimes, J. Lewis, B. Peyton, and H. Simon, *Progress in sparse matrix methods for large linear systems on vector supercomputers*, Internat. J. Supercomput. Appl., 1 (1987), pp. 10–29.

[9] R. E. Benner, G. R. Montry and G. G. Weigand, *Concurrent multifrontal methods: shared memory, cache, and frontwidth issues*, Internat. J. Supercomput. Appl., 1 (1987), pp. 26–44.

[10] A. R. Conn, N. I. M. Gould, M. Lescrenier and Ph. L. Toint, *Performance of a multifrontal scheme for partially separable optimization*, Tech. Report 88/4, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada, 1989.

[11] A. K. Dave and I. S. Duff, *Sparse matrix calculations on the* CRAY-2, Parallel Comput., 5 (1987), pp. 55–64.

[12] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPAK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA.

[13] J. J. Dongarra and S. C. Eisenstat, *Squeezing the most out of an algorithm in* CRAY *Fortran*, ACM Trans. Math. Software, 10 (1894), pp. 219–230.

[14] I. S. Duff, *Full matrix techniques in sparse Gaussian elimination*, in Lecture Notes in Math. (912), G. A. Watson, ed., Springer-Verlag, New York, 1982, pp. 71–84.

[15] ———, *The solution of sparse linear equations on the* CRAY-1, in High Speed Computation, J. S. Kowalik, ed., NATO ASI Series, Vol. F.7, Springer-Verlag, New York, 1984, pp. 293–309.
[16] ———, *Parallel implementation of multifrontal schemes*, Parallel Comput., 3 (1986), pp. 193–204.
[17] ———, *Multiprocessing a sparse matrix code on the Alliant* FX/8, J. Comput. Appl. Math., 27 (1989), pp. 229–239.
[18] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1987.
[19] I. S. DUFF AND J. K. REID, *Experience of sparse matrix codes on the* CRAY-1, Comput. Phys. Comm., 26 (1982), pp. 293–302.
[20] ———, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.
[21] ———, *The multifrontal solution of unsymmetric sets of linear equations*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 633–641.
[22] S. C. EISENSTAT, M. C. GURSKY, M. H. SCHULTZ AND A. H. SHERMAN, *The Yale Sparse Matrix Package* I. *The symmetric code*, Internat. J. Numer. Methods Engrg., 18 (1982), pp. 1145–1151.
[23] S. C. EISENSTAT, M. H. SCHULTZ, AND A. H. SHERMAN, *Applications of an element model for Gaussian elimination*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 85–96.
[24] ———, *Software for sparse Gaussian elimination with limited core memory*, in Sparse Matrix Proceedings, I. S. Duff and G. W. Stewart, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979, pp. 135–153.
[25] J. A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.
[26] J. A. GEORGE, M. HEATH, AND J. W. H. LIU, *Parallel Cholesky factorization on a shared-memory multiprocessor*, Linear Algebra Appl., 77 (1986), pp. 165–187.
[27] J. A. GEORGE AND J. W. H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[28] B. M. IRONS, *A frontal solution program for finite element analysis*, Internat. J. Numer. Methods Engrg., 2 (1970), pp. 5–32.
[29] J. A. G. JESS AND H. G. M. KEES, *A data structure for parallel* L/U *decomposition*, IEEE Trans. Comput., 31 (1982), pp. 231–239.
[30] C. LEISERSON AND J. LEWIS, *Parallel nested dissection*, Advanced Parallel and VLSI Computation Lecture Notes, April 1987.
[31] J. G. LEWIS, B. W. PEYTON, AND A. POTHEN, *A fast algorithm for reordering sparse matrices for parallel factorization*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1146–1173.
[32] J. W. H. LIU, *On general row merging schemes for sparse Givens transformations*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1190–1211.
[33] ———, *On the storage requirement in the out-of-core multifrontal method for sparse factorization*, ACM Trans. Math. Software, 12 (1986), pp. 249–264.
[34] ———, *On threshold pivoting in the multifrontal method for sparse indefinite systems*, ACM Trans. Math. Software, 13 (1987), pp. 250–261.
[35] ———, *A partial pivoting strategy for sparse symmetric matrix decomposition*, ACM Trans. Math. Software, 13 (1987), pp. 173–182.
[36] ———, *Equivalent sparse matrix reordering by elimination tree rotations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 424–444.
[37] ———, *A tree model for sparse symmetric indefinite matrix factorization*, SIAM J. Matrix Anal. Appl., 1 (1988), pp. 26–39.
[38] ———, *The multifrontal method and paging in sparse Cholesky factorization*, ACM Trans. Math. Software, 15 (1989), pp. 310–325.
[39] ———, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.
[40] R. LUCAS, *Solving planar systems of equations on distributed-memory multiprocessor*, Ph.D. thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, 1987.
[41] R. LUCAS, T. BLANK, AND J. TIEMANN, *A parallel solution method for large sparse systems of equations*, IEEE Trans. Computer-Aided Design, CAD-6, 6 (1987), pp. 981–991.
[42] B. W. PEYTON, *Some applications of clique trees to the solution of sparse linear systems*, Ph.D. thesis, Clemson University, Clemson, South Carolina, 1986.
[43] J. K. REID, TREESOLVE, *a Fortran package for solving large sets of linear finite element equations*, Tech. Report CSS 155, Computer Sciences and Systems Division, AERE Harwell, Oxfordshire, UK, 1984.

[44] E. ROTHBERG AND A. GUPTA, *Fast sparse matrix factorization on modern workstation*, Tech. Report STAN-CS-89-1286, Department of Computer Science, Stanford University, Stanford, CA, 1989.

[45] R. SCHREIBER, *A new implementation of sparse Gaussian elimination*, ACM Trans. Math. Software, 8 (1982), pp. 256–276.

[46] B. SPEELPENNING, *The generalized element method*, Tech. Report UIUCDCS-R-78-946, Department of Computer Science, University of Illinois, Urbana-Champaign, IL, 1978.