Electronic Transactions on Numerical Analysis.
Volume 21, pp. 1-19, 2005.
Copyright © 2005, Kent State University.
ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

A TREE-BASED DATAFLOW MODEL FOR THE UNSYMMETRIC MULTIFRONTAL METHOD[*]

STANLEY C. EISENSTAT[†] AND JOSEPH W.H. LIU[‡]

Dedicated to Alan George on the occasion of his 60th birthday

Abstract. This paper introduces a new model to describe the flow of data from update to frontal matrix in the unsymmetric multifrontal method for solving sparse linear systems. The model is based on the elimination tree of an unsymmetric matrix and consists of the edges in this tree together with some cross edges.

By symmetrically renumbering the rows and columns of the coefficient matrix using a tree-based postordering, we can permute the matrix into a bordered block triangular form while preserving the elimination tree. The model simplifies when the matrix has this form, which suggests that a multifrontal implementation based on it should be simpler as well.

We also extend the model to handle pivoting for stability; compare it with others used in the unsymmetric multifrontal method; and point out the implications for parallelism.

Key words. sparse matrix factorization, sparse LU decomposition, elimination tree

AMS subject classifications. 65F05, 65F50

1. Introduction.

The unsymmetric multifrontal method is a powerful scheme for solving a linear system with a large, sparse, unsymmetric coefficient matrix [4, 6, 9, 13]. Here we explore the use of the elimination tree of an unsymmetric matrix [11] to model the flow of updates in the method. Our approach is a generalization of the symmetric case [18] and is somewhat similar to that in [13, 14, 15], which offer a more ad hoc treatment. The paper is organized as follows.

In §2 we review the elimination tree and tree-based, bordered block triangular (BBT) postorderings [11]. In §3 we review the symmetric multifrontal method and point out a fundamental dichotomy in extensions to unsymmetric matrices.

In §4 we describe how to split the update matrix at each step of the multifrontal method and introduce an augmented elimination tree to model the flow of update submatrices. We also show how updates can be bundled to reduce their total number.

In §5 we discuss the major simplifications that arise when the matrix is already upper BBT ordered: update matrices are always decomposed by columns, and each update submatrix goes either to its parent in the elimination tree or to a descendant of an older sibling. These should make implementation much more straight-forward.

In §6 we show how to extend the model when pivoting is required to maintain stability. The extension (which applies only to BBT ordered matrices) easily accommodates both delayed elimination and pivoting within a fully summed block, the techniques used in existing implementations of the unsymmetric multifrontal method.

In §7 we compare our dataflow model with two existing multifrontal models: the symmetric pattern approach [4, 9], which uses the elimination tree of the symmetrized matrix $A + A^t$; and the task-dag/data-dag approach [13, 14, 15], which is more closely related to that considered here.

In §8 we present some concluding remarks.

[*]Received April 8, 2004. Accepted for publication June 7, 2005. Recommended by S. Oliveira.

[†]Department of Computer Science, Yale University, P. O. Box 208285, New Haven, CT 06520-8285 (Stanley.Eisenstat@yale.edu).

[‡]Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3 (Joseph@cs.yorku.ca). This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A5509.

$$A = \begin{pmatrix}
a & & \bullet & & & & & & & \\
\bullet & b & & & \bullet & & & \bullet & & \bullet \\
\bullet & & c & \bullet & & & & & & \\
 & & \bullet & d & & & & & \bullet & \bullet \\
 & & & \bullet & e & \bullet & & \bullet & & \\
 & \bullet & & & & f & & & & \\
 & & & \bullet & & & g & & & \\
\bullet & \bullet & \bullet & & & & & h & & \\
 & & & & & & & & i & \bullet \\
 & & & & \bullet & & \bullet & & & j
\end{pmatrix}$$

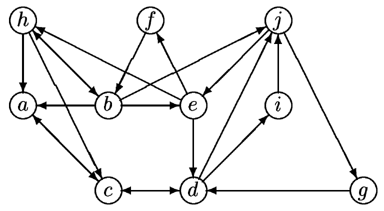


FIG. 2.1. *A sparse matrix and its directed graph.*

$$A^+ = \begin{array}{c|cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & a & & \bullet & & & & & & & \\
2 & \bullet & b & \circ & & \bullet & & & \bullet & & \bullet \\
3 & \bullet & & c & \bullet & & & & & & \\
4 & & & \bullet & d & & & & & \bullet & \bullet \\
5 & & & & \bullet & e & \bullet & & \bullet & \circ & \circ \\
6 & & \bullet & \circ & \circ & \circ & f & & \circ & \circ & \circ \\
7 & & & & \bullet & & & g & & \circ & \circ \\
8 & \bullet & \bullet & \bullet & \circ & \circ & \circ & & h & \circ & \circ \\
9 & & & & & & & & & i & \bullet \\
10 & & & & & \bullet & \circ & \bullet & \circ & \circ & j
\end{array}$$

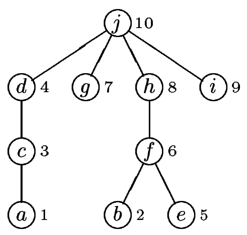


FIG. 2.2. *The filled matrix and elimination tree for the matrix in Figure 2.1.*

2. Background. The authors introduced the elimination tree of a sparse unsymmetric matrix and studied tree-based, bordered block-triangular orderings in [11]. We briefly review these ideas here.

2.1. Graph notation. We let $G(M)$ denote the directed graph of a given $n \times n$ sparse matrix M, and let $r \overset{M}{\longmapsto} c$ (resp., $r \overset{M}{\Longrightarrow} c$) indicate the existence of a directed edge (resp., path[1]) from vertex r to vertex c in that graph. If M is clear from context, we shall sometimes use the abbreviated form $r \mapsto c$ (resp., $r \Rightarrow c$).

Figure 2.1 contains a 10×10 unsymmetric matrix that will be used as an example throughout the paper. Each • represents an off-diagonal nonzero; each vertex in the directed graph is labeled by its corresponding diagonal entry.

2.2. The elimination tree of an unsymmetric matrix. Let A be a nonsingular, sparse, unsymmetric $n \times n$ matrix with a nonzero diagonal and the factorization $A = LU$, where L is unit lower triangular and U is upper triangular; and let $A^+ = L + U - I$ denote its *filled matrix*. We define the *elimination tree* $T(A)$ of A by the parent function

$$\rho[k] = \min\{x \mid x > k \text{ and } x \overset{L}{\Longrightarrow} k \overset{U}{\Longrightarrow} x\}.$$

By definition we have $\rho[n] = \infty$, so that vertex n is a root. In general $T(A)$ consists of one or more trees,[2] and each vertex k with $\rho[k] = \infty$ is a root.

Figure 2.2 contains the filled matrix A^+ and the elimination tree $T(A)$ for the matrix A in Figure 2.1. Each ∘ represents an off-diagonal fill-in; each vertex in the tree is labeled by its diagonal entry and its row/column index.

[1] Paths and cycles need not be *simple*; that is, they may visit a vertex more than once.
[2] There is only one tree when A is irreducible [11, Corollary 3.7].

An efficient algorithm for finding the elimination tree is given in [10]. Its complexity is $O(nm)$, where m is the number of nonzeros in A, but in practice it runs at least as fast as a symbolic factorization code.

The following result characterizes the ancestor relation in $T(A)$ in terms of paths in $G(L)$ and $G(U)$.

THEOREM 2.1. [11, Theorem 3.2] *Vertex q is an ancestor of vertex k in the elimination tree $T(A)$ if and only if there exists a cycle $q \overset{L}{\Longrightarrow} k \overset{U}{\Longrightarrow} q$.*

For a vertex k in the elimination tree $T(A)$, we let $\mathcal{T}[k]$ denote the subtree of $T(A)$ rooted at k as well as the set of vertices in this subtree. We also define the *immediate family members* of k to be the vertices in the set $(\mathcal{T}[\rho[k]] \setminus \{\rho[k]\}) \setminus \mathcal{T}[k]$, which consists of all descendants of $\rho[k]$ other than k and its descendants. We use the term *older immediate family members* to refer to those immediate family members that are older than k with respect to the numbering.

For example, in the elimination tree of Figure 2.2, vertex b has only one immediate family member, namely vertex e, which is also an older immediate family member. On the other hand, the immediate family members of vertex h are vertices a, c, d, g, and i, but its only older immediate family member is vertex i.

2.3. BBT postorderings. A matrix M is said to be upper *bordered-block-triangular* (BBT) if it can be written in the form

$$M = \begin{pmatrix} M_{1,1} & M_{1,2} & \dots & M_{1,s} & M_{1,s+1} \\ 0 & M_{2,2} & \dots & M_{2,s} & M_{2,s+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & M_{s,s} & M_{s,s+1} \\ M_{s+1,1} & M_{s+1,2} & \dots & M_{s+1,s} & M_{s+1,s+1} \end{pmatrix},$$

where the diagonal blocks M_{ii} are square. If we can find a permutation matrix P such that the reordered matrix PAP^t is in BBT form, we can take advantage of this structure by working with PAP^t instead of A.

Let $c_1, \dots, c_t$ be the children of vertex k in the elimination tree $T(A)$. In a *postordering* of $T(A)$ the vertices within each subtree $\mathcal{T}[c_i]$ are numbered consecutively, and k is numbered immediately after its descendants. Postordering preserves the structure of $T(A)$; that is, letting P denote the corresponding permutation matrix, the elimination tree $T(PAP^t)$ is identical to $T(A)$ up to the numbering of the vertices [11, Theorem 5.1].

To achieve a (recursive) BBT form, an *upper BBT* (resp., *lower BBT*) postordering imposes a further condition on the order in which the subtrees $\mathcal{T}[c_1], \dots, \mathcal{T}[c_t]$ are numbered: an edge in $G(A)$ from a vertex in one subtree to a vertex in another must always be directed from the lower-numbered vertex to the higher-numbered one (resp., higher to lower). An efficient algorithm for finding such orderings is given in [10].

For example, the ordering $g, b, e, f, h, a, c, d, i, j$ is an upper BBT postordering of the elimination tree in Figure 2.2. The permuted matrix PAP^t is given in Figure 2.3. Note that the reordering does *not* preserve the filled graph. However, it does preserve the set of diagonal elements of U [11, Theorem 5.2] and thus is arguably just as stable numerically.

We shall say that a matrix A is upper (resp., lower) *BBT ordered* if the natural ordering is an upper (resp., lower) BBT postordering of $T(A)$.

Let ℓ_{rc} (resp., u_{rc}) denote the (r, c) entry of the lower (resp., upper) triangular factor L (resp., U). The following result states a key property of BBT ordered matrices.

THEOREM 2.2. [11, Theorem 6.1] *If A is upper (resp., lower) BBT ordered, then $\ell_{\rho[k],k} \neq 0$ (resp., $u_{k,\rho[k]} \neq 0$).*

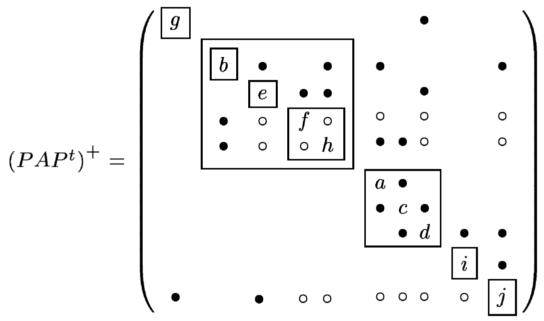


FIG. 2.3. *The filled matrix of an upper BBT postordering of the matrix in Figure 2.1.*

3. The multifrontal method. In this section we review the symmetric multifrontal method and discuss how to extend this approach to unsymmetric problems. We first establish an important property of the rows and columns of the factor matrices.

3.1. A nesting property. Let struct(v) denote the nonzero structure of the vector v, that is, the set of indices i where $v_i \neq 0$. The following nesting property was originally proved for symmetric factorization [18, Theorem 3.1]. We now extend it to the unsymmetric case.

THEOREM 3.1. *Let vertex $p = \rho[k]$ be the parent of vertex k in the elimination tree $T(A)$. Ignoring accidental cancellation we have*

$$\mathrm{struct}((\ell_{pk}, \ldots, \ell_{nk})^t) \subseteq \mathrm{struct}((\ell_{pp}, \ldots, \ell_{np})^t)$$

and

$$\mathrm{struct}((u_{kp}, \ldots, u_{kn}) \subseteq \mathrm{struct}((u_{pp}, \ldots, u_{pn})).$$

Proof. Since p is the parent of k, by definition we have $p \overset{L}{\Longrightarrow} k \overset{U}{\Longrightarrow} p$. If $u_{ij} \neq 0$, a multiple of column i is added to column j during the LU factorization. Thus, by a simple induction argument, for each j on the path $k \overset{U}{\Longrightarrow} p$ we must have

$$\mathrm{struct}((\ell_{pk}, \ldots, \ell_{nk})^t) \subseteq \mathrm{struct}((\ell_{pj}, \ldots, \ell_{nj})^t).$$

In particular, this holds for column p. The second result follows by a similar argument based on the path $p \overset{L}{\Longrightarrow} k$. ☐

This result is the cornerstone of the symmetric multifrontal method: it ensures that each update matrix can be assembled into the frontal matrix of its parent in the symmetric elimination tree. The subtle difference here is that all of the entries $\ell_{k+1,k}, \ldots, \ell_{\rho[k]-1,k}$ in the kth column of L are zero in the symmetric case, whereas some may be nonzero in the unsymmetric case (as may the analogous entries in the kth row of U).

For example, in the elimination tree of Figure 2.2, since vertex 6 is the parent of vertex 2, we have

$$\mathrm{struct}((\ell_{62}, \ell_{72}, \ldots, \ell_{10,2})^t) = \{6, 8\} \subseteq \{6, 8, 10\} = \mathrm{struct}((\ell_{66}, \ell_{76}, \ldots, \ell_{10,6})^t)$$

and

$$\mathrm{struct}((u_{26}, u_{27}, \ldots, u_{2,10})) = \{8, 10\} \subseteq \{6, 8, 9, 10\} = \mathrm{struct}((u_{66}, u_{67}, \ldots, u_{6,10})).$$

Moreover, unlike the symmetric case, we have $u_{26} = 0$ and $u_{23} \neq 0$ and $u_{25} \neq 0$.

```
for $k := 1$ to $n$ do
    Assemble the frontal matrix $\mathcal{F}_k$ from the nonzeros in $[a_{ik}]_{k\le i\le n}$
        and the update matrices $\mathcal{U}_c$ for $c$ a child of $k$ in $T(A)$
    Factor $\mathcal{F}_k$ into $\begin{pmatrix} \ell_{kk} & \\ [\ell_{ik}]_{k<i\le n,\, \ell_{ik}\ne 0} & I \end{pmatrix} \begin{pmatrix} 1 & \\ & \mathcal{U}_k \end{pmatrix} \begin{pmatrix} \ell_{kk} & [\ell_{jk}]^t_{k<j\le n,\, \ell_{jk}\ne 0} \\ & I \end{pmatrix}$
end for
```

FIG. 3.1. *Symmetric multifrontal factorization.*

3.2. The symmetric multifrontal method. The *classic multifrontal method*, first described by Speelpenning, is a powerful approach to solving a linear system with a large, sparse, symmetric coefficient matrix [8, 19]. Each step in the algorithm (see Figure 3.1) consists of the assembly of a frontal matrix $\mathcal{F}_k$ and its partial factorization, yielding the nonzeros in the kth column of L and an update matrix $\mathcal{U}_k$ that (if non-null) will be assembled into a later frontal matrix. In practice only the lower triangles of $\mathcal{F}_k$ and $\mathcal{U}_k$ are formed.

The assembly of $\mathcal{F}_k$ must include every unassembled[3] update matrix that contributes to the kth column of L; that is, every $\mathcal{U}_m$ with $m < k$ that has a nonzero entry in column k and has not already been assembled into another frontal matrix. In the symmetric case these are precisely the updates $\mathcal{U}_c$ with c a child of k in the elimination tree $T(A)$. Thus the flow of data from update matrix to frontal matrix can be modeled in terms of these child-parent edges [18, Theorem 4.1].

The rows of $\mathcal{F}_k$ correspond to those rows i where $\ell_{ik} \ne 0$. Similarly, the columns of $\mathcal{F}_k$ correspond to those columns j where $\ell_{jk} \ne 0$. We have $\ell_{ik} \ne 0$ structurally (i.e., ignoring accidental cancellation) if $a_{ik} \ne 0$ for some $i \ge k$; and by Theorem 3.1 or [18, Theorem 3.1] we have both $\ell_{ik} \ne 0$ and $\ell_{jk} \ne 0$ structurally if the (i, j) entry in the update $\mathcal{U}_c$ from a child c of k is nonzero for some $i \ge j \ge k$. Thus there is an entry in $\mathcal{F}_k$ to accommodate each nonzero assembled into it.

During the assembly process we add each nonzero in $[a_{ik}]_{k\le i\le n}$ and each nonzero in an update $\mathcal{U}_c$ to the corresponding entry in $\mathcal{F}_k$, which was initialized to zero. Then we perform the partial factorization of $\mathcal{F}_k$ in place as a dense matrix operation. The nonzeros in the kth column of L now comprise the first column and row of $\mathcal{F}_k$; and the nonzeros in $\mathcal{U}_k$ comprise the remainder of $\mathcal{F}_k$. This is the key to the efficiency of the symmetric multifrontal method.

3.3. The unsymmetric multifrontal method. The multifrontal approach has been extended to unsymmetric problems [4, 6, 9, 13]. By analogy with the symmetric case, we might expect the algorithm to be that shown in Figure 3.2. The assembly of $\mathcal{F}_k$ includes every unassembled update matrix that contributes to the kth column of L or the kth row of U; that is, every $\mathcal{U}_m$ with $m < k$ that has a nonzero entry in column k or row k and has not already been assembled into another frontal matrix. The partial factorization of $\mathcal{F}_k$ yields the nonzeros in the kth column of L and kth row of U and an update matrix $\mathcal{U}_k$ that (if non-null) will be assembled into a later frontal matrix.

This algorithm makes two assumptions, which are satisfied in the symmetric case:

a) The rows (resp., columns) of $\mathcal{F}_k$ correspond to those rows i where $\ell_{ik} \ne 0$ (resp., columns j where $u_{kj} \ne 0$).

b) Each update matrix $\mathcal{U}_m$ can be assembled into a single frontal matrix.

But if the nonzero structure of A is unsymmetric, an update $\mathcal{U}_m$ to $\mathcal{F}_k$ may contain a nonzero entry in a position (i, j) where $\ell_{ik} = 0$ or $u_{kj} = 0$. For example, for the matrix in Figure 2.1

[3]If a previously assembled update matrix contains a contribution to column k, that contribution will be part of the update from the frontal matrix into which it was assembled.

```
for $k := 1$ to $n$ do
    Assemble the frontal matrix $\mathcal{F}_k$ from the nonzeros in $[a_{ik}]_{k \le i \le n}$,
        $[a_{kj}]_{k \le j \le n}$, and some update matrices $\mathcal{U}_m$ with $m < k$
    Factor $\mathcal{F}_k$ into $\begin{pmatrix} \ell_{kk} & \\ [\ell_{ik}]_{k<i\le n,\, \ell_{ik}\ne 0} & I \end{pmatrix} \begin{pmatrix} 1 & \\ & \mathcal{U}_k \end{pmatrix} \begin{pmatrix} u_{kk} & [u_{kj}]_{k<j\le n,\, u_{kj}\ne 0} \\ & I \end{pmatrix}$
end for
```

FIG. 3.2. *Unsymmetric multifrontal factorization.*

the update

$$\mathcal{U}_1 = \begin{array}{c|c} & 3 \\ \hline 2 & \circ \\ 3 & \circ \\ 8 & \circ \end{array}$$

to $\mathcal{F}_2$ contains a nonzero entry in position $(3,3)$ but $\ell_{32} = 0$. Thus the assumptions are incompatible in general.

The usual approach [4, 9], which satisfies (b) but not (a), is based on the elimination tree $T(A + A^t)$ of the symmetrized matrix $A + A^t$.

Let $\tilde{\ell}_{rc}$ denote the (r,c) entry in the symmetric factor $\tilde{L}$ of $A + A^t$. If $\ell_{ik} \ne 0$ (resp., $u_{kj} \ne 0$), then $\tilde{\ell}_{ik} \ne 0$ (resp., $\tilde{\ell}_{jk} = (\tilde{L}^t)_{kj} \ne 0$), structurally if not numerically. That is, the graph $G(L)$ (resp., $G(U)$) is a subgraph of $G(\tilde{L})$ (resp., $G(\tilde{L}^t)$); and the structure of A^+ is contained in that of the filled matrix $(A + A^t)^+$.

Thus we can apply the algorithm in Figure 3.2 if we treat an entry of A as structurally nonzero whenever the corresponding entry of $A + A^t$ is structurally nonzero; and use the larger frontal matrix $\tilde{\mathcal{F}}_k$ whose rows and columns correspond to those rows i with $\tilde{\ell}_{ik} \ne 0$ and those columns j with $\tilde{\ell}_{jk} \ne 0$. As in the symmetric case, the updates $\mathcal{U}_m$ all come from the children of k in $T(A + A^t)$.

However, even when we suppress rows or columns that are structurally zero [4], in general some frontal matrices will have nonzero entries in rows (resp., columns) whose leftmost (resp., topmost) entry is zero, violating property (a). For example, using this approach, the frontal matrix $\tilde{\mathcal{F}}_2$ for the matrix in Figure 2.1 is given by

$$\tilde{\mathcal{F}}_2 = \begin{array}{c|ccccc} & 2 & 3 & 5 & 8 & 10 \\ \hline 2 & b & \circ & \bullet & \bullet & \bullet \\ 3 & & \circ & & & \\ 6 & \bullet & & & & \\ 8 & \bullet & \circ & & & \end{array} = \begin{array}{c|cccc} & 2 & 5 & 8 & 10 \\ \hline 2 & b & \bullet & \bullet & \bullet \\ 6 & \bullet & & & \\ 8 & \bullet & & & \end{array} \oplus\!\!\!\!+ \; \mathcal{U}_1, \quad \text{where} \quad \mathcal{U}_1 = \begin{array}{c|c} & 3 \\ \hline 2 & \circ \\ 3 & \circ \\ 8 & \circ \end{array}.$$

(Here rows 5 and 10 and column 6 are structurally zero and have been suppressed; and $\oplus\!\!\!\!+$ denotes the extend-add operator, whereby the two operand submatrices are extended to conform with the row and column subscript sets formed by the union of their respective sets and added together [18].) Note the structural zero in the first column.

An alternative[4] is to impose property (a), but to break each update matrix into disjoint pieces that are assembled into *different* frontal matrices [13, 14]. The assembly process is more complicated, but only dense matrix operations are performed throughout the factorization as in the symmetric multifrontal method. We explore this idea in the next section.

[4] Another alternative is to order for sparsity while carrying out the multifrontal method, determining the frontal matrix dynamically [6].

```
Let $p = \rho[k]$, the parent of vertex $k$
Set $T = \mathcal{U}_k$, the update matrix from vertex $k$
while $T$ is non-null do
    Set $s$ = smallest row or column index in $T$
    if $s \geq p$ then
        Forward all of $T$ to vertex $p$ and set $T$ = null
    else if $s$ is a row index then
        Remove row $s$ from $T$ and forward it to vertex $s$
    else {$s$ is a column index}
        Remove column $s$ from $T$ and forward it to vertex $s$
    end if
end while
```

FIG. 4.1. *A simple algorithm to decompose the update matrix $\mathcal{U}_k$.*

4. A dataflow model.

4. A dataflow model. In this section we show how to use the elimination tree to split the update matrices and model the flow of data in the unsymmetric multifrontal method.

4.1. Simple decompositions of update matrices. After forming the frontal matrix $\mathcal{F}_k$ associated with vertex k, one step of elimination on $\mathcal{F}_k$ generates the kth column of L, the kth row of U, and the update matrix $\mathcal{U}_k$.

Theorem 3.1 ensures that the frontal matrix $\mathcal{F}_{\rho[k]}$ of the parent $\rho[k]$ of k can accommodate every nonzero entry in $\mathcal{U}_k$ whose row and column indices are both greater than or equal to $\rho[k]$. Thus the submatrix with these entries can be sent directly to $\rho[k]$. The remainder, consisting of entries whose row index or column index is less than $\rho[k]$, must be split and the pieces sent to other vertices. The simplest approach is the algorithm in Figure 4.1, which "peels" rows or columns off the update matrix until the parent row/column is reached.[5]

We use the matrix in Figure 2.2 to illustrate this process. In the first step the frontal matrix $\mathcal{F}_1$ and update matrix $\mathcal{U}_1$ are given by

$$\mathcal{F}_1 = \begin{array}{c|cc} & 1 & 3 \\ \hline 1 & a & \bullet \\ 2 & \bullet & \\ 3 & \bullet & \\ 8 & \bullet & \end{array} \quad \text{and} \quad \mathcal{U}_1 = \begin{array}{c|c} & 3 \\ \hline 2 & \circ \\ 3 & \circ \\ 8 & \circ \end{array}.$$

The update $\mathcal{U}_1$ is split into two parts,

$$\mathcal{U}_1^2 = \begin{array}{c|c} & 3 \\ \hline 2 & \circ \end{array} \quad \text{and} \quad \mathcal{U}_1^{\rho[1]} \equiv \mathcal{U}_1^3 = \begin{array}{c|c} & 3 \\ \hline 3 & \circ \\ 8 & \circ \end{array},$$

with $\mathcal{U}_1^2$ sent to vertex 2 and $\mathcal{U}_1^{\rho[1]} \equiv \mathcal{U}_1^3$ sent to vertex 3, the parent of vertex 1 in the elimination tree.

In the second step the assembly of the frontal matrix $\mathcal{F}_2$ includes the update $\mathcal{U}_1^2$, giving

$$\mathcal{F}_2 = \begin{array}{c|ccccc} & 2 & 3 & 5 & 8 & 10 \\ \hline 2 & b & \circ & \bullet & \bullet & \bullet \\ 6 & \bullet & & & & \\ 8 & \bullet & & & & \end{array} = \begin{array}{c|cccc} & 2 & 5 & 8 & 10 \\ \hline 2 & b & \bullet & \bullet & \bullet \\ 6 & \bullet & & & \\ 8 & \bullet & & & \end{array} \oplus \mathcal{U}_1^2 \quad \text{and} \quad \mathcal{U}_2 = \begin{array}{c|cccc} & 3 & 5 & 8 & 10 \\ \hline 6 & \circ & \circ & \circ & \circ \\ 8 & \circ & \circ & \circ & \circ \end{array}.$$

[5] This is an improvement over [13], which uses in place of $\rho[k]$ the parent function

$$\rho_{LU}[k] = \min\{x \mid x \overset{L}{\Longrightarrow} k \overset{U}{\longmapsto} x \text{ or } x \overset{L}{\longmapsto} k \overset{U}{\Longrightarrow} x\}.$$

Note that $\rho[k] \leq \rho_{LU}[k]$ in general, but that $\rho[k] = \rho_{LU}[k]$ when A is BBT ordered, by Theorem 2.2.

The update matrix $\mathcal{U}_2$ is split into three parts,

$$\mathcal{U}_2^3 = \begin{array}{r@{}l} & \begin{matrix} 3 \end{matrix} \\ \begin{matrix} 6 \\ 8 \end{matrix} & \begin{pmatrix} \circ \\ \circ \end{pmatrix} \end{array}, \quad \mathcal{U}_2^5 = \begin{array}{r@{}l} & \begin{matrix} 5 \end{matrix} \\ \begin{matrix} 6 \\ 8 \end{matrix} & \begin{pmatrix} \circ \\ \circ \end{pmatrix} \end{array}, \quad \text{and} \quad \mathcal{U}_2^{\rho[2]} \equiv \mathcal{U}_2^6 = \begin{array}{r@{}l} & \begin{matrix} 8 & 10 \end{matrix} \\ \begin{matrix} 6 \\ 8 \end{matrix} & \begin{pmatrix} \circ & \circ \\ \circ & \circ \end{pmatrix} \end{array}.$$

In the third step the frontal matrix $\mathcal{F}_3$ includes the two updates $\mathcal{U}_1^3$ and $\mathcal{U}_2^3$, giving

$$\mathcal{F}_3 = \begin{array}{r@{}l} & \begin{matrix} 3 & 4 \end{matrix} \\ \begin{matrix} 3 \\ 4 \\ 6 \\ 8 \end{matrix} & \begin{pmatrix} \bullet & \bullet \\ \bullet & \\ \circ & \\ \bullet & \end{pmatrix} \end{array} = \begin{array}{r@{}l} & \begin{matrix} 3 & 4 \end{matrix} \\ \begin{matrix} 3 \\ 4 \end{matrix} & \begin{pmatrix} c & \bullet \\ \bullet & \end{pmatrix} \end{array} \oplus \mathcal{U}_1^3 \oplus \mathcal{U}_2^3 \quad \text{and} \quad \mathcal{U}_3 = \begin{array}{r@{}l} & \begin{matrix} 4 \end{matrix} \\ \begin{matrix} 4 \\ 6 \\ 8 \end{matrix} & \begin{pmatrix} \circ \\ \circ \\ \circ \end{pmatrix} \end{array}.$$

Since the entire update matrix can be sent to the parent vertex, no splitting is necessary; that is, $\mathcal{U}_3^{\rho[3]} \equiv \mathcal{U}_3^4 = \mathcal{U}_3$.

In the fourth step the frontal matrix $\mathcal{F}_4$ includes the update $\mathcal{U}_3^4$, giving

$$\mathcal{F}_4 = \begin{array}{r@{}l} & \begin{matrix} 4 & 9 & 10 \end{matrix} \\ \begin{matrix} 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} & \begin{pmatrix} \bullet & \bullet & \bullet \\ \bullet & & \\ \circ & & \\ \bullet & & \\ \circ & & \end{pmatrix} \end{array} = \begin{array}{r@{}l} & \begin{matrix} 4 & 9 & 10 \end{matrix} \\ \begin{matrix} 4 \\ 5 \\ 7 \end{matrix} & \begin{pmatrix} d & \bullet & \bullet \\ \bullet & & \\ \bullet & & \end{pmatrix} \end{array} \oplus \mathcal{U}_3^4 \quad \text{and} \quad \mathcal{U}_4 = \begin{array}{r@{}l} & \begin{matrix} 9 & 10 \end{matrix} \\ \begin{matrix} 5 \\ 6 \\ 7 \\ 8 \end{matrix} & \begin{pmatrix} \circ & \circ \\ \circ & \circ \\ \circ & \circ \\ \circ & \circ \end{pmatrix} \end{array}.$$

The update matrix $\mathcal{U}_4$ is split into four parts,

$$\mathcal{U}_4^5 = \begin{array}{r@{}l} & \begin{matrix} 9 & 10 \end{matrix} \\ 5 & \begin{pmatrix} \circ & \circ \end{pmatrix} \end{array}, \quad \mathcal{U}_4^6 = \begin{array}{r@{}l} & \begin{matrix} 9 & 10 \end{matrix} \\ 6 & \begin{pmatrix} \circ & \circ \end{pmatrix} \end{array}, \quad \mathcal{U}_4^7 = \begin{array}{r@{}l} & \begin{matrix} 9 & 10 \end{matrix} \\ 7 & \begin{pmatrix} \circ & \circ \end{pmatrix} \end{array}, \quad \text{and} \quad \mathcal{U}_4^8 = \begin{array}{r@{}l} & \begin{matrix} 9 & 10 \end{matrix} \\ 8 & \begin{pmatrix} \circ & \circ \end{pmatrix} \end{array}.$$

The update $\mathcal{U}_4^{\rho[4]} \equiv \mathcal{U}_4^{10}$ to the parent vertex is the null matrix.

In the fifth step the frontal matrix $\mathcal{F}_5$ includes the updates $\mathcal{U}_2^5$ and $\mathcal{U}_4^5$, giving

$$\mathcal{F}_5 = \begin{array}{r@{}l} & \begin{matrix} 5 & 6 & 8 & 9 & 10 \end{matrix} \\ \begin{matrix} 5 \\ 6 \\ 8 \\ 10 \end{matrix} & \begin{pmatrix} e & \bullet & \bullet & \circ & \circ \\ \circ & & & & \\ \circ & & & & \\ \bullet & & & & \end{pmatrix} \end{array} = \begin{array}{r@{}l} & \begin{matrix} 5 & 6 & 8 \end{matrix} \\ \begin{matrix} 5 \\ 10 \end{matrix} & \begin{pmatrix} e & \bullet & \bullet \\ \bullet & & \end{pmatrix} \end{array} \oplus \mathcal{U}_2^5 \oplus \mathcal{U}_4^5 \quad \text{and} \quad \mathcal{U}_5 = \begin{array}{r@{}l} & \begin{matrix} 6 & 8 & 9 & 10 \end{matrix} \\ \begin{matrix} 6 \\ 8 \\ 10 \end{matrix} & \begin{pmatrix} \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \end{pmatrix} \end{array}.$$

No splitting is necessary, so $\mathcal{U}_5^{\rho[5]} \equiv \mathcal{U}_5^6 = \mathcal{U}_5$.

We will consider one more step. The frontal matrix $\mathcal{F}_6$ includes the updates $\mathcal{U}_2^6$, $\mathcal{U}_4^6$, and $\mathcal{U}_5^6$, giving

$$\mathcal{F}_6 = \begin{array}{r@{}l} & \begin{matrix} 6 & 8 & 9 & 10 \end{matrix} \\ \begin{matrix} 6 \\ 8 \\ 10 \end{matrix} & \begin{pmatrix} \bullet & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \end{pmatrix} \end{array} = \begin{array}{r@{}l} & 6 \\ 6 & \begin{pmatrix} f \end{pmatrix} \end{array} \oplus \mathcal{U}_2^6 \oplus \mathcal{U}_4^6 \oplus \mathcal{U}_5^6 \quad \text{and} \quad \mathcal{U}_6 = \begin{array}{r@{}l} & \begin{matrix} 8 & 9 & 10 \end{matrix} \\ \begin{matrix} 8 \\ 10 \end{matrix} & \begin{pmatrix} \circ & \circ & \circ \\ \circ & \circ & \circ \end{pmatrix} \end{array}.$$

Again no splitting is necessary. Note that unlike the nonzeros in the other updates, the nonzeros in $\mathcal{U}_2^6$ and $\mathcal{U}_5^6$ are not limited to the leftmost column and topmost row of $\mathcal{F}_6$.

ETNA
Kent State University
etna@mcs.kent.edu

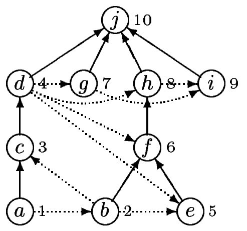


FIG. 4.2. *The dataflow graph for the matrix of Figure 2.2.*

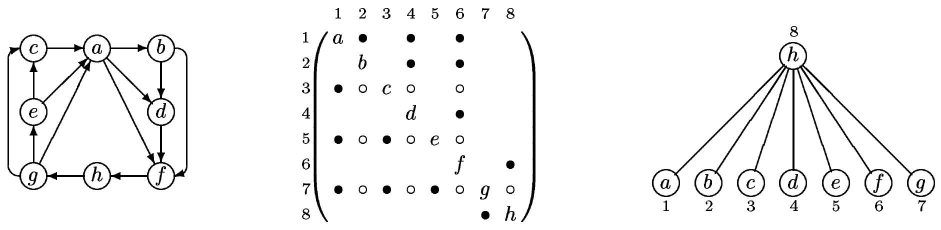


FIG. 4.3. *A chorded cycle and its filled matrix and elimination tree.*

4.2. The dataflow graph. The decomposition drives the flow of data from update matrix to frontal matrix, and this flow can be modeled by a *directed acyclic graph (dag)*. Each vertex corresponds to a frontal matrix and its associated update matrix (i.e., a row/column index); each edge corresponds to the assembly of all or part of an update matrix into a frontal matrix. If we assume that every vertex sends a (possibly null) update to its parent, the dag contains every edge in the elimination tree, directed from child to parent. In addition there is an edge from k to s for each non-null update $\mathcal{U}_k^s$ with $k < s < \rho[k]$. Each such edge corresponds to a nonzero in A^+.

Figure 4.2 gives this *dataflow graph* for the example of Figure 2.2, with the nine non-tree edges represented by dotted arrows. As we will see in §5, the situation is much simpler for BBT ordered matrices.

4.3. Length-two reductions. There is more freedom in splitting an update matrix than the simple algorithm in Figure 4.1 suggests [13, p. 537].

Consider the matrix in Figure 4.3, which corresponds to a cycle with six chords. Its first frontal and update matrices are given by

$$\mathcal{F}_1 = \begin{array}{c} \\ 1 \\ 3 \\ 5 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 4 & 6 \end{array} \\ \begin{pmatrix} a & \bullet & \bullet & \bullet \\ \bullet & & & \\ \bullet & & & \\ \bullet & & & \end{pmatrix} \end{array} \quad \text{and} \quad \mathcal{U}_1 = \begin{array}{c} \\ 3 \\ 5 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 4 & 6 \end{array} \\ \begin{pmatrix} \circ & \circ & \circ \\ \circ & \circ & \circ \\ \circ & \circ & \circ \end{pmatrix} \end{array}.$$

Since vertex 8 is the parent of vertex 1, the update matrix $\mathcal{U}_1$ must be decomposed before being sent to other vertices. The algorithm in Figure 4.1 will split $\mathcal{U}_1$ as

$$\begin{array}{c} \\ 3 \\ 5 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 4 & 6 \end{array} \\ \begin{pmatrix} 2 & 3 & 3 \\ 2 & 4 & 5 \\ 2 & 4 & 6 \end{pmatrix} \end{array},$$

```
Let $p = \rho[k]$, the parent of vertex $k$
Set $T = \mathcal{U}_k$, the update matrix from vertex $k$
**while** $T$ is non-null **do**
    Set $s$ = smallest row or column index in $T$
    **if** $s \geq p$ **then**
        Forward all of $T$ to vertex $p$ and set $T$ = null
    **else if** $s$ is a row index **then**
        Remove from $T$ row $s$ and *some* set of other rows $i$ that are length-two
            reducible to it (i.e., $i \overset{L}{\longmapsto} s \overset{L}{\longmapsto} k$) and forward them to vertex $s$
    **else** $\{s$ is a column index$\}$
        Remove from $T$ column $s$ and *some* set of other columns $j$ that are length-two
            reducible to it (i.e., $k \overset{U}{\longmapsto} s \overset{U}{\longmapsto} j$) and forward them to vertex $s$
    **end if**
**end while**
```

FIG. 4.4. *An algorithm to decompose the update matrix* $\mathcal{U}_k$.

where each entry specifies the index of the vertex to which it is sent.

Because the $(2, 4)$ entry of A^+ is nonzero, the frontal matrix $\mathcal{F}_2$ can accommodate column 4 of $\mathcal{U}_1$. Thus we could send that column to vertex 2 for assembly into $\mathcal{F}_2$, from which this contribution to $\mathcal{F}_4$ would be sent to vertex 4. This gives the split

$$\begin{array}{c} \\ 3 \\ 5 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 4 & 6 \end{array} \\ \begin{pmatrix} 2 & 2 & 3 \\ 2 & 2 & 5 \\ 2 & 2 & 6 \end{pmatrix} \end{array}.$$

Similarly, because the $(5, 3)$ and $(7, 3)$ entries of A^+ are nonzero, the frontal matrix $\mathcal{F}_3$ can accommodate rows 5 and 7 of $\mathcal{U}_1$. Thus we could send these rows to vertex 3, leading to the split

$$\begin{array}{c} \\ 3 \\ 5 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 4 & 6 \end{array} \\ \begin{pmatrix} 2 & 3 & 3 \\ 2 & 3 & 3 \\ 2 & 3 & 3 \end{pmatrix} \end{array}.$$

More generally, let $k < s < t < \rho[k]$. If $u_{ks} \neq 0$, $u_{kt} \neq 0$, and $u_{st} \neq 0$ (resp., $\ell_{sk} \neq 0$, $\ell_{tk} \neq 0$, and $\ell_{ts} \neq 0$), then the frontal matrix $\mathcal{F}_s$ can accommodate column (resp., row) t of the update matrix $\mathcal{U}_k$. Thus, provided that no update has been bundled into the update $\mathcal{U}_k^t$ and that the update $\mathcal{U}_k^s$ has not been bundled into another update, $\mathcal{U}_k^t$ could be bundled with $\mathcal{U}_k^s$ and sent to vertex s; and this contribution to the frontal matrix $\mathcal{F}_t$ would eventually arrive as part of the update $\mathcal{U}_s^t$ from s.

We shall call this bundling *length-two reduction* since the path $k \overset{U}{\longmapsto} s \overset{U}{\longmapsto} t$ (resp., $t \overset{L}{\longmapsto} s \overset{L}{\longmapsto} k$) of length two in the graph $G(U)$ (resp., $G(L)$) replaces the edge $k \overset{U}{\longmapsto} t$ (resp., $t \overset{L}{\longmapsto} k$), which corresponds to the direct update from k to t. This process lowers the *number* of non-null update matrices (but not the volume of data sent), which could be significant in a parallel context.

The splitting algorithm shown in Figure 4.4 processes nonzeros top-to-bottom and left-to-right, and thus automatically satisfies the conditions above. Its 18 possible decompositions of $\mathcal{U}_1$ for the chorded cycle are shown in Figure 4.5.

ETNA
Kent State University
etna@mcs.kent.edu

$$
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 2 & 2 \\ 5 & 2 & 2 & 2 \\ 7 & 2 & 2 & 2 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 2 & 3 \\ 5 & 2 & 2 & 3 \\ 7 & 2 & 2 & 3 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 2 & 3 \\ 5 & 2 & 2 & 3 \\ 7 & 2 & 2 & 6 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 2 & 3 \\ 5 & 2 & 2 & 5 \\ 7 & 2 & 2 & 3 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 2 & 3 \\ 5 & 2 & 2 & 5 \\ 7 & 2 & 2 & 5 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 2 & 3 \\ 5 & 2 & 2 & 5 \\ 7 & 2 & 2 & 6 \end{array}
$$

$$
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 2 \\ 5 & 2 & 3 & 2 \\ 7 & 2 & 3 & 2 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 2 \\ 5 & 2 & 3 & 2 \\ 7 & 2 & 4 & 2 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 2 \\ 5 & 2 & 4 & 2 \\ 7 & 2 & 3 & 2 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 2 \\ 5 & 2 & 4 & 2 \\ 7 & 2 & 4 & 2 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 3 \\ 5 & 2 & 3 & 3 \\ 7 & 2 & 3 & 3 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 3 \\ 5 & 2 & 3 & 3 \\ 7 & 2 & 4 & 4 \end{array}
$$

$$
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 3 \\ 5 & 2 & 3 & 3 \\ 7 & 2 & 4 & 6 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 3 \\ 5 & 2 & 4 & 4 \\ 7 & 2 & 3 & 3 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 3 \\ 5 & 2 & 4 & 5 \\ 7 & 2 & 3 & 3 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 3 \\ 5 & 2 & 4 & 4 \\ 7 & 2 & 4 & 4 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 3 \\ 5 & 2 & 4 & 5 \\ 7 & 2 & 4 & 5 \end{array}
\quad
\begin{array}{c|ccc} & 2 & 4 & 6 \\ \hline 3 & 2 & 3 & 3 \\ 5 & 2 & 4 & 5 \\ 7 & 2 & 4 & 6 \end{array}
$$

FIG. 4.5. *All possible decompositions of the update matrix $\mathcal{U}_1$ of Figure 4.3. Each entry specifies the vertex to which it is forwarded.*

$$
\begin{array}{c|cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \hline
1 & g & & & & & & & \bullet & & \\
2 & & b & \bullet & & \bullet & \bullet & & & & \bullet \\
3 & & & e & \bullet & \bullet & & & \bullet & & \\
4 & & \bullet & \circ & f & \circ & \circ & & \circ & & \circ \\
5 & & \bullet & \circ & \circ & h & \bullet & \bullet & \circ & & \circ \\
6 & & & & & & a & \bullet & & & \\
7 & & & & & & \bullet & c & \bullet & & \\
8 & & & & & & & \bullet & d & \bullet & \bullet \\
9 & & & & & & & & & i & \bullet \\
10 & \bullet & & \bullet & \circ & \circ & \circ & \circ & \circ & \circ & j
\end{array}
$$

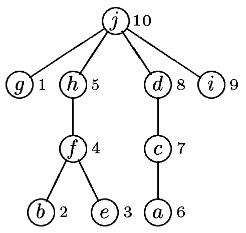


FIG. 5.1. *The BBT ordered matrix of Figure 2.3 and its elimination tree.*

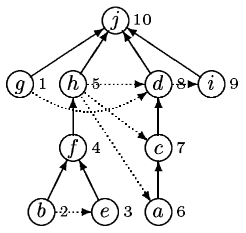
5. BBT ordered matrices. When a matrix is BBT ordered, the decomposition of the update matrices and the dataflow graph are much simplified.[6] In this section we assume that the matrix is upper BBT ordered. We will use as an example the matrix of Figure 2.3, given again in Figure 5.1 along with its elimination tree, which we have rearranged to better reflect the sequence in which the subtrees are numbered.

5.1. Decompositions and the dataflow graph. Let k be a vertex and let $p = \rho[k]$ be its parent. Since an upper BBT postordering of the elimination tree numbers the vertices in each subtree consecutively, the vertices between k and p must be descendants of p but not descendants of k; that is, they must be older immediate family members of k in $T(A)$. And since in an upper BBT postordering there are no edges from an older immediate family member to k, we have $\ell_{ik} = 0$ for $k < i < p$. Finally, we have $\ell_{pk} \neq 0$ by Theorem 2.2. Thus the decomposition of the update matrices in the algorithms in Figures 4.1 and 4.4 will always be by *columns*.

This uniformity simplifies both the dataflow graph and the implementation of the unsymmetric multifrontal algorithm. In particular the update $\mathcal{U}_k^p$ will contain the columns of the update matrix $\mathcal{U}_k$ with indices between p and n. The remaining columns will be split into groups; and there will be an update $\mathcal{U}_k^j$ containing column j for each $k < j < p$ with $u_{kj} \neq 0$, unless that update has been bundled with another. Thus in the dataflow model for an upper BBT ordered matrix with no length-two reductions, there is a tree edge from k to p and an extra *cross edge* from k to each $k < j < p$ with $u_{kj} \neq 0$.

[6] A BBT postordering of the elimination tree can increase the fill and (to a lesser extent) the work required [10]. However, it also enables a variation of the usual, stack-based implementation [2, 9].

ETNA
Kent State University
etna@mcs.kent.edu



$$\begin{array}{c|cccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & g & & & & & & & \bullet & & \\
2 & & b & \bullet & & \cdot & \cdot & & & & \cdot \\
3 & & & e & \cdot & \cdot & & & \cdot & & \\
4 & & \bullet & \circ & f & \cdot & \cdot & & \cdot & & \cdot \\
5 & & \cdot & \cdot & \circ & h & \bullet & \bullet & \circ & & \cdot \\
6 & & & & & & a & \cdot & & & \\
7 & & & & & & \bullet & c & \cdot & & \\
8 & & & & & & & \bullet & d & \bullet & \cdot \\
9 & & & & & & & & & i & \cdot \\
10 & \bullet & & \cdot & \cdot & \circ & \cdot & \cdot & \circ & \circ & j
\end{array}$$

FIG. 5.2. *Two views of the dataflow graph of the matrix of Figure 5.1.*

$$\begin{pmatrix}
1 & \bullet & \bullet & \bullet & \bullet & \bullet \\
 & 2 & \bullet & & & \bullet \\
 & & 3 & \bullet & \bullet & \bullet \\
 & & & 4 & \bullet & \bullet \\
 & & & & 5 & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & 6
\end{pmatrix}
\quad
\begin{pmatrix}
1 & \bullet & \bullet & \bullet & \bullet & \cdot \\
 & 2 & \bullet & & & \cdot \\
 & & 3 & \bullet & \bullet & \cdot \\
 & & & 4 & \bullet & \cdot \\
 & & & & 5 & \cdot \\
\bullet & \bullet & \bullet & \bullet & \bullet & 6
\end{pmatrix}
\quad
\begin{pmatrix}
1 & \bullet & \cdot & \bullet & \bullet & \cdot \\
 & 2 & \bullet & & & \cdot \\
 & & 3 & \bullet & \cdot & \cdot \\
 & & & 4 & \bullet & \cdot \\
 & & & & 5 & \cdot \\
\bullet & \bullet & \bullet & \bullet & \bullet & 6
\end{pmatrix}
\quad
\begin{pmatrix}
1 & \bullet & \cdot & \bullet & \cdot & \cdot \\
 & 2 & \bullet & & & \cdot \\
 & & 3 & \bullet & \cdot & \cdot \\
 & & & 4 & \bullet & \cdot \\
 & & & & 5 & \cdot \\
\bullet & \bullet & \bullet & \bullet & \bullet & 6
\end{pmatrix}$$

Original matrix	No length-two reductions	All practical reductions	Maximum number of reductions

FIG. 5.3. *A matrix and several possible dataflow models.*

The dataflow graph for the matrix in Figure 5.1 is given in Figure 5.2. The six cross edges corresponding to non-parental updates are represented by dotted arrows.

In the same figure we use the nonzero structure of the filled matrix A^+ to present another view of the dataflow graph. As noted above we have $\ell_{pk} \neq 0$ for each parental update $\mathcal{U}_k^p$ (tree edge from k to p); and by construction $u_{kj} \neq 0$ for each non-parental update $\mathcal{U}_k^j$ (cross edge from k to j). Thus after we replace every other off-diagonal nonzero in A^+ by a $\cdot$, the remaining nonzeros in the strict lower (resp., upper) triangle represent tree edges (resp., cross edges) and are denoted $\bullet$ or $\circ$, depending on their origin.

5.2. Length-two reductions. Length-two reductions remove edges from the dataflow graph. However, finding an optimum set of reductions seems very difficult; and even a greedy implementation of the algorithm in Figure 4.4 (that is, one that removes all edges reducible to $k \overset{U}{\longmapsto} s$ at each iteration) may be too expensive.

If f_k is the column index of the first off-diagonal nonzero in the kth row of U and $f_k < \rho[k]$, the edge $k \overset{U}{\longmapsto} f_k$ cannot be reduced. Therefore, a practical compromise is to identify only length-two reductions where $k \overset{U}{\longmapsto} f_k \overset{U}{\longmapsto} j$ replaces $k \overset{U}{\longmapsto} j$ for some $k < j$.

The 6×6 matrix in Figure 5.3 illustrates this compromise. Note that column 5 of the update matrix can be bundled with column 4 because of the length-two path $1 \overset{U}{\longmapsto} 4 \overset{U}{\longmapsto} 5$. But this path is not of the form $k \overset{U}{\longmapsto} f_k \overset{U}{\longmapsto} j$, so the edge $1 \overset{U}{\longmapsto} 5$ will be kept in the practical dataflow graph. Moreover, even without this restriction column 4 cannot be bundled with column 3 since the latter has already been bundled with column 2.

All practical reductions can be identified using the algorithm in Figure 5.4. The total time required is $O(nnz(U))$.

6. Pivoting for stability. So far we have assumed that the elimination sequence is fixed. That is, after we assemble the frontal matrix $\mathcal{F}_k$, we perform a step of Gaussian elimination to generate the kth column of L, the kth row of U, and the update matrix $\mathcal{U}_k$. But what if the kth pivot is not acceptable numerically?

```
for $j := 1$ to $n$ do
    Set $exists[j] = 0$
    Set $S[j]$ = empty set
end for
for $k := 1$ to $n$ do
    for each nonzero $u_{kj}$ in row $k$ do
        Set $exists[j] = k$
    end for
    for each row $i$ in $S[k]$ (i.e., each row with $f_i = k$) do
        for each nonzero $u_{ij}$ in row $i$ with $k < j < \rho[i]$ do
            if $exists[j] = k$ (i.e., $u_{kj} \neq 0$) then
                Reduce edge $i \xmapsto{U} j$ to $i \xmapsto{U} k \xmapsto{U} j$
            end if
        end for
    end for
    Add row $k$ to $S[f_k]$ (= set of rows $i$ with $f_i = k$)
end for
```

FIG. 5.4. *Finding all practical length-two reductions for an upper BBT ordered matrix.*

The simplest approach is the one that was used in the GESP (Gaussian elimination with static pivoting) algorithm[7] [16]. If the magnitude of any pivot would be less than $\sqrt{\epsilon}\,\|A\|_1$, where ϵ is the machine precision, then that pivot is replaced by $\sqrt{\epsilon}\,\|A\|_1$. When combined with a preprocessing step that moves large entries onto the main diagonal and an extra stage of iterative refinement that compensates for the perturbation, this scheme seems to work well in practice [17]. And it does not require any changes to the dataflow model.

But as we will show in this section, the model can also handle more traditional approaches to pivoting [2, 6, 9, 13, 14, 15] when the matrix is BBT ordered. We begin by discussing block matrices and supernodes.

6.1. Block matrices and supernodes. Assume that we can symmetrically partition A into a *block matrix* whose diagonal blocks are dense and whose off-diagonal blocks are either dense or zero. Then we can easily extend the unsymmetric multifrontal method to perform *block* elimination on A and thereby compute its *block LU* factorization. We can model the flow of data with a *block* dataflow graph[8] whose vertices correspond to the diagonal blocks of A. And because we are doing block elimination, pivoting *within* a diagonal block has no effect on the graph.

Supernodes are a more general form of blocking that is commonly used to improve the efficiency of symmetric [5] and unsymmetric [7] sparse numerical factorization. Assume that $\ell_{k+1,k}$ and $u_{k,k+1}$ are both nonzero. Then vertex k is a child of vertex $k+1$ in the elimination tree $T(A)$, and the entire update matrix $\mathcal{U}_k$ will be forwarded to vertex $k+1$ and assembled into the frontal matrix $\mathcal{F}_{k+1}$.

Assume further that the nonzero structures of the columns of[9] $A^+(k\!:\!n, k\!:\!k\!+\!1)$ are identical and that the structures of the rows of $A^+(k\!:\!k\!+\!1, k\!:\!n)$ are also identical. Then $\mathcal{U}_k$ will have the same nonzero rows and columns as $\mathcal{F}_{k+1}$.

Thus if we assemble into the frontal matrix $\mathcal{F}_k$ all updates (other than $\mathcal{U}_k$) destined for either $\mathcal{F}_k$ or $\mathcal{F}_{k+1}$ and perform two steps of elimination, we will have computed columns k and $k+1$ of L, rows k and $k+1$ of U, and the update $\mathcal{U}_{k+1}$. There is no need to create $\mathcal{U}_k$ explicitly. The set $\{k, k+1\}$ is said to be a supernode.

[7] Existing implementations of GESP are column-based, not multifrontal.
[8] This graph is also the quotient graph of the dataflow graph for A with respect to the blocking.
[9] We use the MATLAB notations $p\!:\!q$, denoting the sequence $p, p+1, \ldots, q$, and $A^+(P, Q)$, denoting the submatrix of A^+ defined by the row sequence P and the column sequence Q.

More generally the set $K = \{k, k+1, \ldots, m\}$ is a supernode if the diagonal subblock $A^+(K, K)$ is dense; the columns in the submatrix $A^+(k\!:\!n, K)$ have the same nonzero structure; and the rows in the submatrix $A^+(K, k\!:\!n)$ have the same structure. As before we can assemble all updates destined for $\mathcal{F}_k$, $\mathcal{F}_{k+1}$, ..., and $\mathcal{F}_m$ from earlier vertices into the supernodal frontal matrix $\mathcal{F}_K$ and perform $m - k + 1$ steps of elimination (of the vertices in K). This will compute $L(k\!:\!n, K)$, $U(K, k\!:\!n)$, and the update matrix $\mathcal{U}_m$. There is no need to create $\mathcal{U}_k, \mathcal{U}_{k+1}, \ldots$, and $\mathcal{U}_{m-1}$ explicitly.

In terms of the dataflow graph this corresponds to coalescing the vertices in the set K into a single supernode K. To do so we delete the edges between vertices in K; replace by $x \mapsto K$ any edges of the form $x \mapsto i$ with $x < k \leq i \leq m$; and replace by $K \mapsto y$ any edges of the form $m \mapsto y$ with $m < y \leq n$. (Since $k \mapsto \cdots \mapsto m$ is a chain in $T(A)$, there are no edges $i \mapsto y$ with $k \leq i < m$.)

Rather than perform $m - k + 1$ separate steps of elimination, we could instead perform a single $(m - k + 1) \times (m - k + 1)$ block elimination and compute something equivalent to $L(k\!:\!n, K)$ and $U(K, k\!:\!n)$. The advantage of block elimination is that any pivoting within the $(m - k + 1) \times (m - k + 1)$ pivot block, which consists of the *fully summed* rows and columns of $\mathcal{F}_K$, will not affect the supernodal dataflow graph.

6.2. Delayed elimination for upper BBT ordered matrices. Pivoting within supernodal blocks may not be sufficient. Thus when a vertex k is not an acceptable (diagonal) pivot, we delay its elimination until immediately after the elimination of its parent p. (The pivot value would be unchanged if we tried to eliminate k any earlier [11, Theorem 5.2].) The corresponding symmetric permutation P has a local effect on the elimination tree [11, Theorem 7.2] and thus on the dataflow graph.

One drawback of this approach is that there is no guarantee that vertex p is an acceptable pivot; e.g., suppose that every remaining diagonal element is zero. Following [9] we address this problem by treating $\{p, k\}$ as a supernode, which permits pivoting within the supernodal block.[10] The effect on the dataflow graph is again local (see the discussion of supernodes above).

Note that to satisfy the definition of supernode we may have to treat certain zero entries in the filled matrix $(PAP^t)^+$ as nonzero, which constitutes additional fill. However, all such entries lie in the rows and columns corresponding to vertices k and p in the original ordering,[11] and there is no effect on fill during the remainder of the factorization.

Another drawback is that the dataflow graph of PAP^t does not capture all flows. In particular, we do not discover that the vertex k is an unacceptable pivot until after we have received all of the updates to k and have assembled them into the frontal matrix $\mathcal{F}_k$; and we may have to package some of the nonzero entries in $\mathcal{F}_k$ as updates to vertices between k and p in the original elimination order. None of these flows is captured.

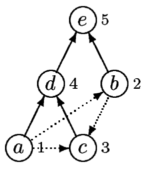
We address this problem by keeping the *original* dataflow graph with vertex p now representing the new supernode and vertex k now representing an *assembly-only* vertex at which:

- All updates are assembled into $\mathcal{F}_k$.
- Row k of $\mathcal{F}_k$ is moved immediately after row p; column k is moved either immediately after column p (if present) or to where column p would be (otherwise).
- The *entire* $\mathcal{F}_k$ is treated as the update matrix; that is, it is decomposed in the usual way with the pieces sent along edges in the dataflow graph to vertices between k and p in the original order.

[10] There is no guarantee that the supernodal block will be an acceptable pivot either, but we can delay all or part of its elimination in the same fashion.

[11] Row/column k may get more fill than in the original ordering, but row/column p cannot.

$$A^+ = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & a & \bullet & & \bullet & \\ 2 & & b & & & \bullet \\ 3 & \bullet & \circ & c & \circ & \circ \\ 4 & & & \bullet & d & \circ \\ 5 & & & & \bullet & e \end{array}$$



$$(PAP^t)^+ = \begin{array}{c|ccccc} & 2 & 3 & 4 & 1 & 5 \\ \hline 2 & b & & & & \bullet \\ 3 & & c & & \bullet & \\ 4 & & \bullet & d & \circ & \\ 1 & \bullet & & \bullet & a & \circ \\ 5 & & & \bullet & \circ & e \end{array}$$

FIG. 6.1. *A filled matrix, its dataflow graph, and the filled matrix after a delayed pivot.*

In terms of an implementation based on this extended model the only modification required is to notify the vertices receiving updates, including p, that k has been permuted to follow p and that the two vertices have been coalesced into a supernode. What makes this scheme work for an upper BBT ordered matrix is that each recipient vertex between k and p in the original order is expecting an update that contains row p; and that p can accommodate an update that contains column k since they have coalesced into a supernode.

For example, for the matrix in Figure 2.3 the frontal matrix of vertex 5 is

$$\mathcal{F}_5 = \begin{array}{c|ccccc} & 5 & 6 & 7 & 8 & 10 \\ \hline 5 & h & \bullet & \bullet & \circ & \circ \\ 10 & \circ & \circ & & \circ & \circ \end{array} = \begin{array}{c|ccc} & 5 & 6 & 7 \\ \hline 5 & h & \bullet & \bullet \end{array} \oplus \mathcal{U}_4^5.$$

If we delay the elimination of vertex 5 until after the elimination of its parent 10, then the "update" matrix is

$$\mathcal{U}_5 = \begin{array}{c|ccccc} & 6 & 7 & 8 & 10 & 5 \\ \hline 10 & \circ & \cdot & \circ & \circ & \circ \\ 5 & \bullet & \bullet & \circ & \circ & h \end{array} = \mathcal{U}_5^6 \oplus \mathcal{U}_5^7 \oplus \mathcal{U}_5^8 \oplus \mathcal{U}_5^{10}$$

where $\cdot$ denotes an entry that would have filled in with the original ordering and

$$\mathcal{U}_5^6 = \begin{array}{c|c} & 6 \\ \hline 10 & \circ \\ 5 & \bullet \end{array}, \quad \mathcal{U}_5^7 = \begin{array}{c|c} & 7 \\ \hline 10 & \cdot \\ 5 & \bullet \end{array}, \quad \mathcal{U}_5^8 = \begin{array}{c|c} & 8 \\ \hline 10 & \circ \\ 5 & \circ \end{array}, \quad \text{and} \quad \mathcal{U}_5^{10} = \begin{array}{c|cc} & 10 & 5 \\ \hline 10 & \circ & \circ \\ 5 & \circ & h \end{array}.$$

Note that the recipient of each of these updates is expecting a contribution to row 10; and that vertex 10 (representing the supernode $\{10, 5\}$) can accommodate a contribution to column 5.

These conditions need not be satisfied if A is not BBT ordered, so the process could break down. For example, for the matrix in Figure 6.1 the frontal matrix of vertex 1 is

$$\mathcal{F}_1 = \begin{array}{c|ccc} & 1 & 2 & 4 \\ \hline 1 & a & \bullet & \bullet \\ 3 & \bullet & & \end{array}$$

If a is an unacceptable pivot and we delay the elimination of vertex 1 until after the elimination of its parent 4, then the "update" matrix is

$$\mathcal{U}_1 = \begin{array}{c|ccc} & 2 & 4 & 1 \\ \hline 3 & \cdot & \cdot & \bullet \\ 1 & \bullet & \bullet & a \end{array} = \mathcal{U}_1^2 \oplus \mathcal{U}_1^3 \oplus \mathcal{U}_1^4$$

where

$$\mathcal{U}_1^2 = \begin{array}{c|c} & 2 \\ \hline 3 & \cdot \\ 1 & \bullet \end{array}, \quad \mathcal{U}_1^3 = \begin{array}{c|cc} & 4 & 1 \\ \hline 3 & \cdot & \bullet \end{array}, \quad \text{and} \quad \mathcal{U}_1^4 = \begin{array}{c|cc} & 4 & 1 \\ \hline 1 & \bullet & a \end{array}.$$

The frontal matrix for vertex 2 is

$$\mathcal{F}_2 = \begin{array}{c|cc} & 2 & 5 \\ \hline 2 & b & \bullet \\ 3 & \cdot & \\ 1 & \bullet & \end{array} = \begin{array}{c|cc} & 2 & 5 \\ \hline 2 & b & \bullet \end{array} \oplus \mathcal{U}_1^2$$

and its update matrix is

$$\mathcal{U}_2 = \begin{array}{c|c} & 5 \\ \hline 3 & \circ \\ 1 & \circ \end{array} = \mathcal{U}_2^3 \oplus \mathcal{U}_2^4,$$

where

$$\mathcal{U}_2^3 = \begin{array}{c|c} & 5 \\ \hline 3 & \circ \end{array} \quad \text{and} \quad \mathcal{U}_2^4 = \begin{array}{c|c} & 5 \\ \hline 1 & \bullet \end{array}.$$

However, there is no edge from vertex 2 to vertex 4 (into which vertex 1 was coalesced) over which to send the update $\mathcal{U}_2^4$, nor can the edge $2 \mapsto 4$ be length-two reduced (which would allow that update to be sent to an intermediate vertex).

This kind of pivoting is similar to what is done in the symmetric indefinite multifrontal method [8] and in unsymmetric multifrontal methods that are based either on the elimination tree $T(A + A^t)$ of the symmetrized matrix $A + A^t$ [2, 3, 9, 12] or on data-dags [13, 14, 15]. The difference is that we work with the elimination tree $T(A)$ of the original matrix and its associated dataflow graph. This has significant advantages, as we show in the next section.

7. Relation to existing multifrontal models.

In this section we investigate how our dataflow model compares with other models of the unsymmetric multifrontal method.

7.1. Model based on $T(A + A^t)$.

As mentioned in §3.3, many current approaches to the unsymmetric multifrontal method use the elimination tree of the symmetrized matrix $A+A^t$ as the model [2, 3, 9, 12]. We first establish some simple relationships between $T(A)$ and $T(A + A^t)$.

THEOREM 7.1. *If vertex $p = \rho[k]$ is the parent of vertex k in the elimination tree $T(A)$, then p is an ancestor of k in the elimination tree $T(A + A^t)$. If $x > k$ and either $\ell_{xk} \neq 0$ or $u_{kx} \neq 0$, then vertex x is an ancestor of k in $T(A + A^t)$.*

Proof. Let $\tilde{L}$ be the symmetric factor of $A + A^t$. As noted in §3.3, the graph $G(L)$ (resp., $G(U)$) is a subgraph of $G(\tilde{L})$ (resp., $G(\tilde{L}^t)$). If p is the parent of k in $T(A)$, there exists a cycle $p \overset{L}{\Longrightarrow} k \overset{U}{\Longrightarrow} p$, whence $p \overset{\tilde{L}}{\Longrightarrow} k \overset{\tilde{L}^t}{\Longrightarrow} p$. Thus p is an ancestor of k in $T(A + A^t)$ by Theorem 2.1. On the other hand, if $x > k$ and $\ell_{xk} \neq 0$, we have $x \overset{L}{\longmapsto} k \overset{L^t}{\longmapsto} x$, whence $x \overset{\tilde{L}}{\Longrightarrow} k \overset{\tilde{L}^t}{\Longrightarrow} x$. Thus x is an ancestor of k in $T(A + A^t)$ by Theorem 2.1. The case $u_{kx} \neq 0$ is similar. $\square$

The second part of Theorem 7.1 does not imply the first since p can be the parent of k even if $\ell_{pk} = 0$ and $u_{kp} = 0$. For example, in the chorded cycle of Figure 4.3, vertex h is the parent of vertex a, yet $\ell_{81} = u_{18} = 0$. However, by Theorem 2.2 this cannot happen if A is BBT ordered.

The data model based on $T(A)$ sends updates from a vertex k to its parent and, via added edges, to certain other vertices x for which $\ell_{xk} \neq 0$ or $u_{kx} \neq 0$. If the data model based on $T(A + A^t)$ were used instead, the entire update from k would be sent to its parent in $T(A + A^t)$. By Theorem 7.1 one part would eventually reach the parent of k in $T(A)$; and, since the edges added to $T(A)$ lead to ancestors in $T(A + A^t)$, the other parts would also eventually reach their destinations.

This suggests that the elimination tree $T(A)$ is a more refined model for the unsymmetric multifrontal method than $T(A + A^t)$. There are three major practical differences:

1. Each update in the $T(A)$ model goes directly to its destination, barring length-two reductions; and even with length-two reductions, the path in $T(A)$ is never longer. Thus there is less data movement/assembly.
2. All updates in the $T(A)$ model are full matrices. Thus there is less arithmetic. Indeed, merely shrinking the update matrices in the $T(A+A^t)$ model by removing zero rows and columns, which does not eliminate all embedded structural zeros, can lead to a significant improvement in both storage and execution time [4].
3. The overhead to manage the updates in the $T(A)$ model is greater since each update matrix is split into submatrices. However, this overhead is significantly lower when the matrix is BBT ordered.

7.2. Model based on $G((L^t)^o + U^o)$. An alternative is to use a *task-dag* to specify the task dependencies in the multifrontal method, and a *data-dag* to guide the assembly of data from update matrix to frontal matrix [13] (see also [14, 15]). Here we explore the relationship of these dags to the elimination tree and the dataflow graph.

The task-dag is the union $G((L^t)^o + U^o)$ of the transitive reductions[12] of the graphs $G(L^t)$ and $G(U)$. In general the elimination tree is not a subgraph of the task-dag. For example, in the chorded cycle matrix of Figure 4.3, vertex h is the parent of vertex a in $T(A)$, but there is no direct edge in $G(L^t + U)$ from a to h. However the situation is different for BBT ordered matrices.

THEOREM 7.2. *If A is BBT ordered, then the elimination tree $T(A)$ is a subgraph of the task-dag.*

Proof. If A is upper BBT ordered, the graph $G((L^t)^o)$ is the elimination tree $T(A)$ with its edges directed from child to parent [11, Theorem 6.3]. The case when A is lower BBT ordered is similar. $\square$

The data-dag is an extension of the task-dag that is formed by adding every edge $k \xmapsto{U} j$ satisfying[13]

a) $j < \rho_{LU}[k] \equiv \min\{x \mid x \xRightarrow{L} k \xmapsto{U} x \text{ or } x \xmapsto{L} k \xRightarrow{U} x\}$
b) there does not exist an edge $k \xmapsto{U} s$ in the data-dag such that $s \xmapsto{U} j$
c) there exists a vertex $t > j$ such that $t \xmapsto{L} k$

and every edge in $G(L^t)$ that satisfies the analogous conditions [13, Theorem 4.1]. By contrast the reduced dataflow graph is formed by pruning every length-two reducible edge from the unreduced dataflow graph. In general neither the dataflow graph nor the data-dag is a subgraph of the other. However the situation is different for BBT ordered matrices.

THEOREM 7.3. *Let A be irreducible and upper BBT ordered, and assume that edges $k \xmapsto{U} j$ are considered for pruning from the unreduced dataflow graph (via reduction) or adding to the task-dag (via extension) in left-to-right order within each row. Then the reduced dataflow graph is the data-dag.*

Proof. As noted in the proof of Theorem 7.2, the transitive reduction $G((L^t)^o)$ is the elimination tree $T(A)$ with edges directed from child to parent.

[12]A *transitive reduction* of a directed graph $G(M)$ is a subgraph $G(M^o)$ of $G(M)$ such that there is a path from vertex u to vertex v in $G(M)$ if and only if there is a path from u to v in $G(M^o)$, and no subgraph with this property has fewer edges. The matrix M^o is its representation. The transitive reduction of a dag is unique [1].

[13] The order in which edges are considered for inclusion is not specified in [13], so the data-dag need *not* be unique.

Let vertex $p = \rho[k]$ be the parent[14] of vertex k in $T(A)$, so that $p \stackrel{L}{\Longrightarrow} k \stackrel{U}{\Longrightarrow} p$. As noted at the start of §5.1, we have $\ell_{pk} \neq 0$ and $\ell_{ik} = 0$ for $k < i < p$. Since $p \stackrel{L}{\longmapsto} k \stackrel{U}{\Longrightarrow} p$, we have $\rho_{LU}[k] = \rho[k]$. By construction no edge $k \stackrel{L^t}{\longmapsto} i$ can be a cross edge in the dataflow graph; and by the analog of property (a) above no such edge can be in the extension to the data-dag. Thus it suffices to show that the dataflow graph and the data-dag have the same subset of edges of the form $k \stackrel{U}{\longmapsto} j$ with $j \neq p$. (The tree edges have already been accounted for.)

Assume that $k \stackrel{U}{\longmapsto} j$ with $j > p$. By construction this edge is not in the unreduced dataflow graph; by property (a) above it will not be added to the task-dag; and since $p \stackrel{U}{\longmapsto} j$ by Theorem 3.1, we have $k \stackrel{U}{\Longrightarrow} p \stackrel{U}{\longmapsto} j$, so that it is not in the transitive reduction $G(U^o)$. Thus it suffices to consider the set of edges of the form $k \stackrel{U}{\longmapsto} j$ with $j < p$ that are not in $G(U^o)$. (The edges in $G(U^o)$ are not reducible and must be in both the dataflow graph and the data-dag.)

Because edges are considered for pruning or adding in left-to-right order, the condition to add such an edge to the task-dag is just

There does not exist an s with $k < s < j$ such that $k \stackrel{U}{\longmapsto} s$ is already in the data-dag and $s \stackrel{U}{\longmapsto} j$.

(Since $j < p$ and $\ell_{pk} \neq 0$ by Theorem 2.2, the other conditions are already satisfied.) But this is precisely the condition that this edge *not* be pruned from the unreduced dataflow graph. $\square$

There is an analogous result for lower BBT ordered matrices.

Dynamic pivoting of the kind described in §6 requires a second data-dag D^P (see [13]). This dag is constructed from the original data-dag D^N by adding each edge in the elimination tree $T(A)$, if it is not already present, and any additional edges $k \stackrel{U}{\longmapsto} j$ (or $k \stackrel{L^t}{\longmapsto} j$) that satisfy the three conditions in [13, Theorem 4.5], the first of which is

$k < j < \rho[k]$ and there exists a child s of j such that $k \stackrel{L}{\Longrightarrow} s$ (or $s \stackrel{U}{\Longrightarrow} k$).

But in a BBT ordered matrix, if $k < j < \rho[k]$, then any child s of j must satisfy $k < s < j$. Thus this condition is never satisfied, and Theorem 7.3 holds for D^P as well as D^N.

For BBT ordered matrices the dataflow model is identical to both data-dag models, although their derivations are quite dissimilar. However, the elimination tree is needed to define BBT ordering; and without it neither the simplified decomposition nor the collapse of the $D^N \subseteq D^P$ hierarchy is possible. Thus the dataflow model is arguably more refined.

8. Concluding remarks.

Our dataflow model for the unsymmetric multifrontal method is based on the elimination tree of a sparse unsymmetric matrix. Update matrices are split and flow along either tree edges or certain cross edges in an augmented tree. When the sparse matrix has symmetric structure, the model reduces to that for the symmetric multifrontal method.

The model is particularly simple when the sparse matrix is in BBT form, and this suggests that an implementation based on it should also be simpler. But any unsymmetric matrix can be permuted to such a form efficiently [10].

Another advantage of BBT form is robustness under pivoting. When a pivot is unacceptable the model adapts easily to the combination of pivoting within supernodal blocks and delayed elimination that is used in unsymmetric multifrontal codes based on the elimination tree $T(A + A^t)$ or on the data-dag. However, it is more refined.

Finally, because the dataflow graph captures all dependencies of frontal matrices on update matrices, it also reveals whatever high-level parallelism is available. And since it is more

[14] Since A is irreducible, only vertex n lacks a parent; but there are no edges in either the dataflow graph or the data-dag directed from that vertex.

refined than the $T(A + A^t)$ model, it exposes more such parallelism.

Acknowledgments. We would like to thank the referees for their careful reading of the manuscript and their suggested improvements.

REFERENCES

[1] A. V. AHO, M. R. GAREY, AND J. D. ULLMAN, *The transitive reduction of a directed graph*, SIAM J. Comput., 1 (1972), pp. 131–137.

[2] P. R. AMESTOY AND I. S. DUFF, *Vectorization of a multiprocessor multifrontal code*, International Journal of Supercomputer Applications, 3 (1989), pp. 41–59.

[3] P. R. AMESTOY, I. S. DUFF, J.-Y. L'EXCELLENT, AND X. S. LI, *Analysis and comparison of two general sparse solvers for distributed memory computers*, ACM Trans. Math. Software., 27 (2001), pp. 388–421.

[4] P. R. AMESTOY AND C. PUGLISI, *An unsymmetrized multifrontal LU factorization*, SIAM J. Matrix Anal. Appl., 24 (2002), pp. 553–569.

[5] C. ASHCRAFT, R. GRIMES, J. LEWIS, B. PEYTON, AND H. SIMON, *Progress in sparse matrix methods for large sparse linear systems on vector supercomputers*, Intern. J. of Supercomputer Applications, 1 (1987), pp. 10–30.

[6] T. A. DAVIS AND I. S. DUFF, *An unsymmetric-pattern multifrontal method for sparse LU factorization*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 140–158.

[7] J. W. DEMMEL, S. C. EISENSTAT, J. R. GILBERT, X. S. LI, AND J. W. H. LIU, *A supernodal approach to sparse partial pivoting*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 720–755.

[8] I. S. DUFF AND J. K. REID, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software., 9 (1983), pp. 302–325.

[9] ———, *The multifrontal solution of unsymmetric sets of linear equations*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 633–641.

[10] S. C. EISENSTAT AND J. W. H. LIU, *Algorithmic aspects of elimination trees for sparse unsymmetric matrices*, tech. rep., Department of Computer Science, York University, Toronto, Canada, 2005.

[11] ———, *The theory of elimination trees for sparse unsymmetric matrices*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 686–705.

[12] K. A. GALLIVAN, B. A. MARSOLF, AND H. A. G. WIJSHOFF, *Solving large nonsymmetric sparse linear systems using* MCSPARSE, Parallel Comput., 22 (1996), pp. 1291–1333.

[13] A. GUPTA, *Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices*, SIAM J. Matrix Anal. Appl., 24 (2002), pp. 529–552.

[14] S. M. HADFIELD, *On the LU Factorization of Sequences of Identically Structured Sparse Matrices Within a Distributed Memory Environment*, PhD thesis, University of Florida, Apr. 1994.

[15] S. M. HADFIELD AND T. A. DAVIS, *Lost pivot recovery for an unsymmetric-pattern multifrontal method*, Tech. Rep. TR-94-029, University of Florida, 1994.

[16] X. S. LI AND J. W. DEMMEL, *Making sparse Gaussian elimination scalable by static pivoting*, in Proceedings of Supercomputing '98, Nov. 1998.

[17] ———, *SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems*, ACM Trans. Math. Software., 29 (2003), pp. 110–140.

[18] J. W. H. LIU, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Review, 34 (1992), pp. 82–109.

[19] B. SPEELPENNING, *The generalized element method*, Tech. Rep. UILCDCS-R-78-946, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, Nov. 1978.