RAL 93-036

An unsymmetric-pattern multifrontal method for sparse LU factorization*

Timothy A. Davis† and Iain S. Duff

June 1993

Abstract

Sparse matrix factorization algorithms are typically characterized by irregular memory access patterns that limit their performance on parallel-vector supercomputers. For symmetric problems, methods such as the multifrontal method replace irregular operations with dense matrix kernels. However, no efficient LU factorization algorithm based primarily on dense matrix kernels exists for matrices whose pattern is very unsymmetric. A new unsymmetric-pattern multifrontal method based on dense matrix kernels is presented. Frontal matrices are rectangular instead of square, and the assembly tree is replaced with a directed acyclic graph. As in the classical multifrontal method, advantage is taken of repetitive structure in the matrix by amalgamating nodes in the directed acyclic graph, giving it high performance on parallel-vector supercomputers. The performance of three sequential versions is compared with the classical multifrontal method and other unsymmetric solvers on a Cray YMP-8/128.

Keywords

LU factorization, unsymmetric sparse matrices, parallel algorithms, multifrontal methods

AMS (MOS) subject classifications: 65F50, 65W05, 65F05, 65-04, 68-04.

*also appeared as Report TR/PA/93/98, CERFACS, 42 av. G. Coriolis, 31057 Toulouse Cedex, France and as Technical Report TR-93-018 Computer and Information Sciences Department University of Florida Gainesville, FL, 32611 USA

†Computer and Information Sciences Department, University of Florida, Gainesville, Florida, USA

Abbreviated title. Unsymmetric-pattern multifrontal method

Contents

List of Figures

List of Tables

1 Introduction

Conventional sparse matrix factorization algorithms rely heavily on indirect addressing. This gives them an irregular memory access pattern that limits their performance on typical parallel-vector supercomputers. In contrast, the multifrontal method of Duff and Reid [16] is designed with regular memory access in the innermost loops. Its kernel is one or more steps of LU factorization within each square, dense frontal matrix defined by the nonzero pattern of a pivot row and column. These steps of LU factorization compute a submatrix of update terms that are held within the frontal matrix until they are *assembled* (added) into the frontal matrix of its parent in the assembly tree. The assembly tree (a variant of the elimination tree [26]) controls parallelism across multiple frontal matrices, while dense matrix operations [6] provide parallelism and vectorization within each frontal matrix.

However, this method is based on an assumption of a symmetric nonzero pattern, and so has a poor performance on matrices whose patterns are very unsymmetric. If this assumption is not made, the frontal matrices are rectangular instead of square, a directed acyclic graph (called the *assembly dag*) replaces the assembly tree, and frontal matrices are no longer assembled by a single parent.

A new *unsymmetric-pattern* multifrontal approach respecting these constraints is presented [4]. It builds the assembly dag either during factorization or in a preprocessing phase. As in the symmetric multifrontal case, advantage is taken of repetitive structure in the matrix by amalgamating nodes in the assembly dag. Thus the algorithm uses dense matrix kernels in its innermost loops, giving it high performance on parallel-vector supercomputers.

1.1 Previous work

The D2 algorithm [5] is based on a non-deterministic parallel pivot search that constructs a set of independent pivots (m, say) followed by a parallel rank-m update of the active submatrix. Near the end of the factorization, the active submatrix is considered as a dense matrix (and factorized with dense matrix kernels).

The multifrontal method of Duff and Reid (MA37) [2, 9, 10, 16] will be referred to as the *classical* multifrontal method. The method takes more advantage of dense matrix kernels than D2, but is unsuitable when the pattern of the matrix is very unsymmetric. Many methods for symmetric matrices use dense kernels; a survey may be found in [25].

Most recently, Gilbert and Liu [23] and Eisenstat and Liu [18] have presented symbolic factorization algorithms for unsymmetric matrices, assuming that the pivot ordering is known a priori. The algorithms are based on the *elimination directed acyclic graph (dag)* and its reductions, which are similar to the assembly dag presented in this paper. Moreover, we also indicate how our graphs can be applied to the case where the pivot ordering is not known a priori. Also, Matstoms [28] has recently developed a multifrontal QR factorization

algorithm.

1.2 Outline

The following sections present the unsymmetric-pattern multifrontal method and three sequential algorithms based on the method. Section 2 describes LU factorization in terms of general frontal matrices. Section 3 demonstrates that both the elimination tree and assembly tree are unsuitable for the unsymmetric-pattern multifrontal method. Section 4 presents the assembly graph $\mathcal{D}$ and assembly dag $\mathcal{G}$, and discusses amalgamation and edge reduction on these graphs. The assembly dag $\mathcal{G}$ is to the unsymmetric-pattern multifrontal method as the assembly tree is to the classical multifrontal method. Section 5 discusses an analysis-only algorithm, which computes a symbolic factorization when the pivot sequence is not known in advance. Section 6 presents a factor-only algorithm based on $\mathcal{D}$ and highlights several open problems such as the effects of numerical pivoting. It requires a previous analysis phase, either from the analysis-only algorithm or the combined analysis-factor algorithm described in Section 7. The performance of three analysis-factor algorithms is illustrated in Section 8. Finally, Section 9 summarizes the unsymmetric-pattern multifrontal method, including open problems and future research.

2 LU factorization with general frontal matrices

The LU factorization of an n-by-n matrix A into the product of a lower triangular matrix L (with unit-diagonal) times an upper triangular matrix U consists of n major steps. In the outer-product formulation of Gaussian elimination, A is transformed into the product $L^{[k]}A^{[k]}U^{[k]}$ after step $k-1$ and just before step k ($1 \leq k \leq n, A^{[1]} = A, L^{[n+1]} = L, U^{[n+1]} = U$). Throughout this paper, the notation $X^{[k]}$ will refer to the state of X just before step k, where X is a matrix, set, graph, scalar, etc. The state of X after LU factorization is complete is denoted by $X^{[n+1]}$.

The *active submatrix*, $A^{[k]}_{k..n,k..n}$, is the portion of the matrix $A^{[k]}$ that has still to be reduced after step $k-1$ has finished. We can write

$$L^{[k]}A^{[k]}U^{[k]} = \begin{bmatrix} L^{[k]}_{1..k-1,1..k-1} & 0 \\ L^{[k]}_{k..n,1..k-1} & I_{n-k+1} \end{bmatrix} \cdot \begin{bmatrix} I_{k-1} & 0 \\ 0 & A^{[k]}_{k..n,k..n} \end{bmatrix} \cdot \begin{bmatrix} U^{[k]}_{1..k-1,1..k-1} & U^{[k]}_{1..k-1,k..n} \\ 0 & I_{n-k+1} \end{bmatrix} \quad (1)$$

where matrices $L^{[k]}_{1..k-1,1..k-1}$ and $U^{[k]}_{1..k-1,1..k-1}$ are $(k-1)$-by-$(k-1)$ lower and upper triangular matrices, respectively, I_{k-1} is the $(k-1)$-by-$(k-1)$ identity matrix, and I_{n-k+1} is the $(n-k+1)$-by-$(n-k+1)$ identity matrix.

When we include pivoting to preserve sparsity and maintain numerical accuracy, the matrix PAQ (instead of A) is factorized into LU. The permutation matrices P and Q define

the row and column permutations performed during factorization or during a preprocessing phase. However, to simplify the notation used to describe the method, we will assume that permutations rename rows and columns of $L^{[k]}$, $A^{[k]}$, and $U^{[k]}$. Thus, the partial factorization shown in Equation 1 will be considered as the factorization of $P^{[k]}AQ^{[k]}$, where $P^{[k]}$ and $Q^{[k]}$ are the permutations applied through step $k-1$.

An *entry* in row i and column j of a sparse matrix A is a single value a_{ij} that is symbolically represented in the sparse data structure for A. An entry is typically numerically nonzero, but explicit zero entries might be included if the pattern represents a class of matrices to which the given A belongs. Numerical cancellation is ignored during factorization, so entries in $A^{[k]}$ might become numerically zero. Entries are interchangeably referred to as nonzeros in this paper, with the understanding that such a "nonzero" might actually be numerically zero.

Step k selects a single pivot entry $a_{ij}^{[k]}$ from the active submatrix $A^{[k]}_{k..n,k..n}$, and interchanges row i and column j with the k-th row and column of $A^{[k]}_{k..n,k..n}$. Step k then computes $L^{[k+1]}$, $A^{[k+1]}$, and $U^{[k+1]}$ from $L^{[k]}$, $A^{[k]}$, and $U^{[k]}$.

For a sparse matrix A, define the row and column *structure* (or *pattern*) as the index pattern of the entries in rows or columns of A, viz.

$$Struct\ (A_{i*}) = \{j \mid a_{ij} \neq 0\}$$

$$Struct\ (A_{*j}) = \{i \mid a_{ij} \neq 0\}.$$

The *row degree* $r_i^{[k]}$ $(i \geq k)$ is the number of entries in row i of $A^{[k]}$, and the *column degree* $c_j^{[k]}$ $(j \geq k)$ is the number of entries in column j of $A^{[k]}$, so that

$$r_i^{[k]} = |Struct\ (A_{i*}^{[k]})|$$

$$c_j^{[k]} = |Struct\ (A_{*j}^{[k]})|.$$

The LU factorization can be described in terms of general frontal matrices. An *element* or *general frontal matrix* E_k is a dense, rectangular ($c_j^{[k]}$-by-$r_i^{[k]}$) submatrix that corresponds to the pivot ($a_{ij}^{[k]}$) selected at step k. The columns in E_k are defined by the set $\mathcal{U}_k$, which is the set of column indices of entries in the pivot row i selected at step k. Similarly, the rows in E_k are defined by the set $\mathcal{L}_k$, which is the set of row indices of entries in the pivot column j selected at step k. That is,

$$\mathcal{U}_k = Struct\ (A_{i*}^{[k]})$$

$$\mathcal{L}_k = Struct\ (A_{*j}^{[k]}).$$

The sets $\mathcal{L}_k$ and $\mathcal{U}_k$ are referred to as the row and column pattern, respectively, of the frontal matrix E_k.

Let us assume that another $g_k - 1$ pivots ($g_k \geq 1$) have the same pivot row and column pattern (excluding earlier pivots) as the first pivot in the frontal matrix. In this case, their

pivot rows and columns are also in the element E_k. Entries not in the first g_k rows or columns of E_k form the *contribution block*, D_k, of update terms that are added to $A^{[k]}$ to obtain the reduced matrix $A^{[k+g_k]}$. The elements E_{k+1} to E_{k+g_k-1} are not explicitly represented, having been *amalgamated* into the single element E_k. The g_k major steps of LU factorization within a single element E_k can be performed with dense matrix kernels. Element E_k is referred to as a *supernode* with a pivot block of rank g_k if $g_k > 1$, or as a *simple node* if $g_k = 1$. Two or more rows with identical pattern are referred to as a *super-row*, and columns with identical column pattern are referred to as a *super-column*.

The row and column pattern of E_k ($\mathcal{L}_k$ and $\mathcal{U}_k$) divide into two disjoint subsets:

$$\mathcal{L}_k = \mathcal{L}'_k \cup \mathcal{L}''_k$$

(where $\mathcal{L}'_k$ is the set of g_k pivot rows in E_k) and

$$\mathcal{U}_k = \mathcal{U}'_k \cup \mathcal{U}''_k$$

(where $\mathcal{U}'_k$ is the set of g_k pivot columns in E_k). The sets $\mathcal{L}'_k$, $\mathcal{L}''_k$, $\mathcal{U}'_k$, and $\mathcal{U}''_k$ divide the element E_k into four submatrices: F_k (the g_k-by-g_k *pivot block*), B_k, C_k, and D_k (the contribution block),

$$E_k = \begin{array}{c} \\ \mathcal{L}'_k \\ \mathcal{L}''_k \end{array} \begin{array}{c} \begin{array}{cc} \mathcal{U}'_k & \mathcal{U}''_k \end{array} \\ \begin{bmatrix} F_k & B_k \\ C_k & D_k \end{bmatrix} \end{array}, \tag{2}$$

where the matrix is shown before factorization. The matrices F_k, B_k and C_k are *fully assembled* before the factorization of this frontal matrix begins. That is, after assembly no other nonzeros remain in the g_k pivot rows and columns except those in E_k. However, the submatrix D_k may only hold a partial summation of the original matrix entries and update terms from previous elements.

The numerical factorization within this frontal matrix computes the LU factorization of F_k ($F_k = L'_k U'_k$), computes the block column L''_k of L and the block row U''_k of U, and updates the contribution block with the Schur complement

$$D_k \leftarrow D'_k = D_k - L''_k U''_k.$$

Any entries in F_k can be selected as pivots, as long as they are numerically acceptable. Numerical considerations require pivoting within the pivot block F_k and might limit the number of pivots to less than the maximum (in which case g_k is taken to be the actual number of pivots found in E_k). The matrices L'_k and L''_k define columns k through $k + g_k - 1$ of L, and U'_k and U''_k define rows k through $k + g_k - 1$ of U. The factorized frontal matrix is

$$E_k = \begin{array}{c} \\ \mathcal{L}'_k \\ \mathcal{L}''_k \end{array} \begin{array}{c} \begin{array}{cc} \mathcal{U}'_k & \mathcal{U}''_k \end{array} \\ \begin{bmatrix} L'_k \backslash U'_k & U''_k \\ L''_k & D'_k \end{bmatrix} \end{array}. \tag{3}$$

Entries in the lower-right $(n-k+1)$-by-$(n-k+1)$ submatrix of $P^{[k]}AQ^{[k]}$ that are not yet assembled in a frontal matrix form the *active part* of A, denoted as $M^{[k]}$, where

$$(M^{[k]})_{ij} = \begin{cases} 0 & \text{if } i < k \vee j < k, \\ 0 & \text{if } (P^{[k]}AQ^{[k]})_{ij} \text{ has been assembled into some } E_t \ (1 \le t < k), \\ (P^{[k]}AQ^{[k]})_{ij} & \text{otherwise.} \end{cases} \tag{4}$$

The active part of A is simply the set of original entries that have not yet been assembled into some frontal matrix. The assembly of original entries of A is done when a row or column becomes pivotal, or earlier as a by-product of the degree update. The active submatrix is represented as a sum of elements created during factorization, plus the active part of A,

$$A^{[k]}_{k..n,k..n} = M^{[k]} + \sum_{t \in \mathcal{V}^{[k]}} E_t^{[k]}, \tag{5}$$

where $\mathcal{V}^{[k]}$ is the set of all frontal matrices created through step $k-1$ of the factorization. The active part of an element E_t just before step k (denoted as $E_t^{[k]}$) is a submatrix of E_t formed by rows and columns that are non-pivotal just before step k (where $1 \le t < k$), and also have not been assembled into subsequent frontal matrices. The row and column pattern of $E_t^{[k]}$ are denoted as $\mathcal{L}_t^{[k]}$ and $\mathcal{U}_t^{[k]}$, respectively. Note that $\mathcal{L}_k'' \equiv \mathcal{L}_k^{[k+g_k]}$, $\mathcal{U}_k'' \equiv \mathcal{U}_k^{[k+g_k]}$, and $D_k' \equiv E_k^{[k+g_k]}$.

2.1 Example matrix

Consider the matrix

$$A = \begin{bmatrix} p_1 & \cdot & \cdot & \times & \times & \cdot & \cdot \\ \times & p_2 & \times & \cdot & \times & \cdot & \times \\ \times & \times & p_3 & \cdot & \cdot & \cdot & \times \\ \times & \cdot & \cdot & p_4 & \times & \cdot & \cdot \\ \cdot & \times & \times & \cdot & \times & \times & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \times & \times \\ \times & \times & \cdot & \cdot & \times & \cdot & \times \end{bmatrix} \tag{6}$$

with the partially factorized matrix $L^{[4]}A^{[4]}U^{[4]}$

$$\left[\begin{array}{ccc|cccc} p_1 & \cdot & \cdot & \times & \times & \cdot & \cdot \\ \times & p_2 & \times & \times & \times & \cdot & \times \\ \times & \times & p_3 & \times & \times & \cdot & \times \\ \hline \times & \cdot & \cdot & p_4 & \times & \cdot & \cdot \\ \cdot & \times & \times & \times & \times & \times & \times \\ \cdot & \cdot & \cdot & \cdot & \cdot & \times & \times \\ \times & \times & \times & \times & \times & \cdot & \times \end{array}\right]$$

just before the fourth step of LU factorization. An entry is denoted as $\times$, a zero is a small dot, and p_k denotes the k-th pivot entry. The pivots are assumed to lie on the diagonal in order. The two elements already created are the dense rectangular matrices, E_1 and E_2.

$$E_1 = \begin{array}{c|c|cc} & 1 & 4 & 5 \\ \hline 1 & p_1 & \times & \times \\ \hline 2 & \times & \times & \times \\ 3 & \times & \times & \times \\ 4 & \times & \times & \times \\ 7 & \times & \times & \times \end{array}$$

$$E_2 = \begin{array}{c|cc|ccc} & 2 & 3 & 4 & 5 & 7 \\ \hline 2 & p_2 & \times & \times & \times & \times \\ 3 & \times & p_3 & \times & \times & \times \\ \hline 5 & \times & \times & \times & \times & \times \\ 7 & \times & \times & \times & \times & \times \end{array}$$

The pivot rows and columns have been delineated from the contribution blocks. The active submatrix just before step 4 is

$$A^{[4]}_{4..7,4..7} = M^{[4]} + E_1^{[4]} + E_2^{[4]}$$

or

$$A^{[4]}_{4..7,4..7} = \begin{array}{c|cccc} & 4 & 5 & 6 & 7 \\ \hline 4 & \times & \times & \cdot & \cdot \\ 5 & \cdot & \times & \times & \cdot \\ 6 & \cdot & \cdot & \times & \times \\ 7 & \cdot & \times & \cdot & \times \end{array} + \begin{array}{c|cc} & 4 & 5 \\ \hline 4 & \times & \times \\ 7 & \times & \times \end{array} + \begin{array}{c|ccc} & 4 & 5 & 7 \\ \hline 5 & \times & \times & \times \\ 7 & \times & \times & \times \end{array}.$$

Note that E_3 is not generated, since it lies entirely within E_2. If the entry $a^{[4]}_{4,4}$ (labeled as p_4) is selected at step 4 as the fourth pivot, then the resulting element E_4 would be

$$E_4 = \begin{array}{c|c|c} & 4 & 5 \\ \hline 4 & p_4 & \times \\ \hline 5 & \times & \times \\ 7 & \times & \times \end{array}.$$

Also note that element E_1 makes a contribution to the pivot row of E_4 which cannot be assembled into or represented by the intermediate element E_2, even though E_1 affects portions of E_2. This would not occur if the pattern of A was symmetric.

Row 7 of E_1 can be assembled into E_2, since $\mathcal{U}_1^{[2]} \subset \mathcal{U}_2$, even though the contribution that E_1 makes to row 7 is not needed to factorize E_2. If this assembly is performed, row 7 is removed from $\mathcal{L}_1^{[4]}$ since the notation $E_1^{[4]}$ refers only to portions of E_1 that are not yet assembled into subsequent frontal matrices just before step 4. This assembly is performed by the edge reductions described in Section 4.2.

3 Elimination and assembly trees

The *elimination tree*, $\mathcal{T}$, [26] and its variants (such as the *assembly tree* [15]) are used either explicitly or implicitly in most parallel sparse matrix algorithms [25]:

$$\mathcal{T} = (\mathcal{T}_{\mathcal{V}}, \mathcal{T}_{\mathcal{E}})$$

$$\mathcal{T}_{\mathcal{V}} = 1 \ldots n$$

$$\mathcal{T}_{\mathcal{E}} = \{\langle i, j \rangle \mid j = \mathit{parent}\ (i)\}$$

$$\mathit{parent}\ (i) = \min\{j \mid i < j, l_{ji} \neq 0\}.$$

Starting with the elimination tree, $\mathcal{T}$, an assembly tree is constructed by amalgamating a connected (node-induced) subgraph into a single supernode. The minimum label of the nodes in the subgraph becomes the label of the new supernode. Edges contained in a subgraph are removed, and any edges incident on only one node in the subgraph become incident on the resulting supernode. The process repeats until the desired assembly tree is obtained. Elimination and assembly trees are typically defined only for symmetric-patterned LU factors, with the exception of partial-pivoting methods [21, 22]. A more general graph is needed for the unsymmetric-pattern multifrontal method.

The classical multifrontal method [2, 9, 10, 15, 16] is based on the assembly tree. It has a similar formulation as the general frontal matrix formulation described in the previous section, except that the analysis is performed on the pattern of $A + A^T$. Frontal matrices are square. The method is usually divided into two phases: symbolic analysis and numerical factorization. The symbolic phase finds a suitable pivot ordering using a sparsity preserving heuristic such as minimum degree [20], determines the assembly tree, and finds the patterns of L and U. Each node k in the assembly tree represents the work associated with element E_k. The assembly tree describes the large-grain parallelism between the nodes.

The column pattern $\mathcal{U}_k$ of the frontal matrix E_k is given by the union of the column pattern $\mathcal{U}_j^{[k]}$ of each child j of node k and the pattern of row k of the upper triangular part of A. A node can be amalgamated with one of its children if the column pattern of the parent is identical to the child (excluding the index of the child's pivot column). That is, if $\mathcal{U}_k = \mathcal{U}_j^{[k]}$ for a single child j of node k. Additional amalgamation may be allowed if the patterns are not quite identical, in which case extra fill-in occurs.

The numerical factorization phase uses the assembly tree to compute the LU factorization. The tree guides the assembly process and construction of new elements and describes the precedence between nodes. At node k, the frontal matrices $E_j^{[k]}$ of each child node j are assembled into E_k. Because of the symmetric-pattern assumption, the pattern $\mathcal{U}_k$ is a superset of the pattern $\mathcal{U}_j^{[k]}$ of the contribution block of each child j. The entire contribution block D_j of a child can always be assembled into its parent, since all the rows and columns that are affected by D_j are present in E_k. The method takes advantage of the dense matrix

kernels [6, 7] to factorize E_k: the Level-2 BLAS for simple nodes ($g_k = 1$) or the Level-3 BLAS for supernodes ($g_k > 1$).

The classical multifrontal method is not the only method based on the elimination tree or its variants. In the sparse column-Cholesky factorization of George et al. [19], the work at node k in the elimination tree is the computation of column k of L. The work at node k modifies column k with columns corresponding to a subset of the descendants of node k in the elimination tree. This is in contrast to the classical multifrontal method, in which data is assembled only from the children of a node.

However, the assembly tree is not appropriate in a multifrontal method if the frontal matrices have unsymmetric patterns. This is due to the incomplete assembly that takes place. Consider the inter-relationships between three elements, E_i, E_j, and E_k ($i < j < k$ and $g_i = g_j = g_k = 1$) that results from the following 3-by-3 submatrix of $L\backslash U$ formed from the pivot rows and columns i, j, and k:

$$\begin{bmatrix} p_i & u_{ij} & u_{ik} \\ l_{ji} & p_j & u_{jk} \\ l_{ki} & l_{kj} & p_k \end{bmatrix}$$

(the notation $L\backslash U$ when applied to matrices refers to two matrices packed in the same array). Assume that there are other nonzeros in these rows and columns of U and L, respectively. If, for example, $u_{ik} = u_{jk} = u_{ij} = l_{kj} = 0$, the inter-relationships become

$$\begin{bmatrix} p_i & \cdot & \cdot \\ l_{ji} & p_j & \cdot \\ l_{ki} & \cdot & p_k \end{bmatrix}. \tag{7}$$

In this case, a contribution from E_i is assembled into both E_j and E_k (assuming that there are other nonzeros in row i), but the work at nodes j and k may proceed in parallel after the work at node i completes. The corresponding fragment of a possible subgraph that characterizes these relationships is shown in Figure 1. An assembly tree is not suitable since node i has more than one parent; the parallelism between j and k cannot be described by a tree rooted at node n, and the contributions of a child (node i) are not assembled by a single parent. The contribution that E_i makes to row k *cannot* be assembled into E_j, but the contribution that E_i makes to row j *must* be assembled into E_j.

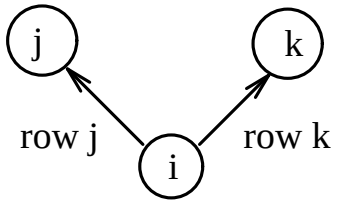

Figure 1: A possible subgraph that characterizes Equation 7

If the assembly tree is unsuitable, what kind of graph can guide the unsymmetric-pattern multifrontal method? The elimination dag [18, 23] is one possibility. Define $\mathcal{G}(L)$ as the directed graph associated with L. That is, $\langle i, j \rangle$ is an edge of $\mathcal{G}(L)$ if and only if l_{ji} is nonzero. Similarly, $\langle i, j \rangle$ is an edge of $\mathcal{G}(U^T)$ if and only if u_{ij} is nonzero. Several reductions to these graphs are described in [18, 23] (such as transitive reduction, which leads to the elimination dag). If the graphs are constructed for a symmetric-patterned matrix and a simple reduction is applied (namely, *symmetric reduction*), then the graphs become equivalent to the elimination tree. The next section extends the results in [18, 23] by allowing for arbitrary sparsity preserving and numerically acceptable pivoting during the factorization. Further extensions are discussed in Section 7.

4 The assembly graph and assembly dag

The *assembly graph*

$$\mathcal{D} = (\mathcal{A}, \mathcal{G}, \mathcal{F})$$

is constructed during the factorization as the pivot sequence is determined. Edge reductions are applied as it is constructed. The notation $\mathcal{D}^{[k]}$, $\mathcal{A}^{[k]}$, $\mathcal{G}^{[k]}$, and $\mathcal{F}^{[k]}$ refer to the state of each of these graphs or edge-sets just before step k of the factorization.

The composite graph consists of three types of nodes in two types of graphs. Conceptually, the bipartite graph

$$\mathcal{A}^{[k]} = (\mathcal{A}_{\mathcal{V}}^{[k]}, \mathcal{A}_{\mathcal{E}}^{[k]})$$

represents the unassembled form of the active submatrix, $A_{k..n,k..n}^{[k]}$, where,

$$\mathcal{A}_{\mathcal{V}}^{[k]} = \mathcal{A}_{\mathcal{R}}^{[k]} \cup \mathcal{A}_{\mathcal{C}}^{[k]},$$

$$\mathcal{A}_{\mathcal{R}}^{[k]} = \{\text{row } k, \ldots, \text{row } n\},$$

$$\mathcal{A}_{\mathcal{C}}^{[k]} = \{\text{column } k, \ldots, \text{column } n\},$$

$$\mathcal{A}_{\mathcal{E}}^{[k]} = \{\langle i, j \rangle \mid i \in \mathcal{A}_{\mathcal{R}}^{[k]} \wedge j \in \mathcal{A}_{\mathcal{C}}^{[k]} \wedge a_{ij}^{[k]} \neq 0\}.$$

The edges $\mathcal{A}_{\mathcal{E}}^{[k]}$ in the bipartite graph $\mathcal{A}^{[k]}$ are not explicitly stored. Rather, they are represented as the active part of A plus any unassembled contribution blocks (as in Equation 5).

The factorized frontal matrices are described by the *assembly dag* (directed acyclic graph)

$$\mathcal{G}^{[k]} = (\mathcal{V}^{[k]}, \mathcal{E}^{[k]})$$

$$\mathcal{V}^{[k]} = \{t \mid E_t \text{ is a frontal matrix created before step } k\}$$

$$\mathcal{V}^{[k]} \subseteq \{1, \ldots, k-1\}$$

$$\mathcal{E}^{[k]} = \mathcal{E}_L^{[k]} \cup \mathcal{E}_U^{[k]}$$

$$\mathcal{E}_L^{[k]} = \{\langle s,t\rangle \quad | \quad s < t < k \wedge s \in \mathcal{V}^{[k]} \wedge t \in \mathcal{V}^{[k]} \wedge \text{ one or more entire rows of } E_s^{[t]} \text{ assembled into } E_t\}.$$

$$\mathcal{E}_U^{[k]} = \{\langle s,t\rangle \quad | \quad s < t < k \wedge s \in \mathcal{V}^{[k]} \wedge t \in \mathcal{V}^{[k]} \wedge \text{ one or more entire columns of } E_s^{[t]} \text{ assembled into } E_t\}$$

Edges in $\mathcal{E}^{[k]}$ are referred to as *inactive L-, U-, or LU-edges*. Inactive edges describe the {L/U/LU}-{parent/child} relationship of factorized frontal matrices. Node s is an *LU-child* of its *LU-parent* node t if $E_s^{[t]}$ is assembled into E_t in its entirety ($\langle s,t\rangle \in \mathcal{E}_L^{[k]} \cap \mathcal{E}_U^{[k]}$). Otherwise, node s is an *L-child* of its *L-parent* node t if $\langle s,t\rangle \in \mathcal{E}_L^{[k]}$, or a *U-child* of its *U-parent* node t if $\langle s,t\rangle \in \mathcal{E}_U^{[k]}$. In the assembly dag, the term *parent* refers to any type of parent, and the term *child* refers to any type of child.

An *active edge* in $\mathcal{F}$ connects a frontal matrix in $\mathcal{G}$ with a row or column in $\mathcal{A}$ for which it holds an unassembled contribution. That is,

$$\mathcal{F}^{[k]} = \mathcal{F}_L^{[k]} \cup \mathcal{F}_U^{[k]}$$

$$\mathcal{F}_L^{[k]} = \left\{\langle t, \text{row } i\rangle \mid i \in \mathcal{L}_t^{[k]}\right\}$$

$$\mathcal{F}_U^{[k]} = \left\{\langle t, \text{column } j\rangle \mid j \in \mathcal{U}_t^{[k]}\right\}.$$

These edges are referred to as *active L-edges* and *active U-edges*, respectively.

4.1 The basic graph

Before factorization starts, the dag $\mathcal{G}^{[1]}$ is empty and $\mathcal{A}^{[1]}$ is the bipartite graph of the original matrix, A.

The pivot search selects a pivot and permutes it to the first row and column (renaming them as row and column one, in our notation). Consecutive pivots with identical pattern are included in the first frontal matrix E_1 which after being factorized is given as

$$E_1 = \begin{array}{c} \\ \mathcal{L}_1' \\ \mathcal{L}_1'' \end{array} \begin{array}{c} \begin{array}{cc} \mathcal{U}_1' & \mathcal{U}_1'' \end{array} \\ \begin{bmatrix} L_1' \backslash U_1' & U_1'' \\ L_2'' & D_1' \end{bmatrix} \end{array},$$

where $g_1 = |\mathcal{L}_1'| = |\mathcal{U}_1'|$ is the number of pivots in the pivot block of E_1.

The first node of $\mathcal{G}$ is constructed. This node refers to the newly factorized frontal matrix E_1. The column pattern $\mathcal{U}_1$ of E_1 is given by the set of edges in $\mathcal{A}^{[1]}$ incident to row nodes 1

through g_1. Similarly, the row pattern of E_1 ($\mathcal{L}_1$) is given by the set of edges in $\mathcal{A}^{[1]}$ incident to column nodes 1 through g_1. Pivot row and column nodes 1 through g_1 (having been renamed) and edges in $\mathcal{A}^{[1]}$ incident to these nodes are then removed from $\mathcal{A}$. The pattern $\mathcal{L}_1''$ defines new active L-edges in $\mathcal{F}_L^{[g_1+1]}$ that are added from node 1 of $\mathcal{G}$ to row nodes in $\mathcal{A}$. Similarly, $\mathcal{U}_1''$ defines new active U-edges that are added from node 1 in $\mathcal{G}$ to column nodes in $\mathcal{A}$. Note that edges from pivots 2 through g_1 are not added, since nodes 2 through g_1 will not appear in $\mathcal{G}$. This is the first case of edge reduction, and is a result of no-fill node amalgamation (additional amalgamation is described in Section 4.3).

New fill-in edges are (implicitly) added to $\mathcal{A}$ for each entry in the contribution block that is not already present in the active submatrix. The resulting bipartite graph is $\mathcal{A}^{[g_1+1]}$, with row and column nodes numbered from $g_1 + 1$ to n. The resulting $\mathcal{G}^{[g_1+1]}$ contains a single node. The factorization of the first frontal matrix satisfies steps 1 through g_1 of the LU factorization. The next step of LU factorization will be step $g_1 + 1$.

An example of a graph $\mathcal{D}^{[2]}$ is shown in Figure 2 for the matrix A in Equation 6 in Section 2.1. The first frontal matrix E_1 consists of only one pivot and is the first node in the graph $\mathcal{G}^{[2]}$. To avoid too many lines, edges in the implicit bipartite graph $\mathcal{A}^{[2]}$ are shown in array form. An original edge between a row i and a column j is shown as an $\times$, a fill-in edge is shown as a $\otimes$, and a zero is shown as a small $\circ$ (for which no edge in $\mathcal{A}$ exists). The two graphs, $\mathcal{G}^{[2]}$ and $\mathcal{A}^{[2]}$, are separated by a dashed line. Edges in $\mathcal{F}^{[k]}$ cross this dashed line, while edges in $\mathcal{E}^{[k]}$ will be placed to the upper-left of this line ($\mathcal{E}^{[2]}$ is empty).

At a subsequent step k, a new pivot row i and column j are selected, and frontal matrix E_k is created. Additional pivots with identical pattern are also included, so that E_k contains pivots k through $k + g_k - 1$. After permutations are made, active edges in $\mathcal{F}^{[k]}$ incident to row and column nodes k through $k + g_k - 1$ in $\mathcal{A}^{[k]}$ represent unassembled numerical contributions to the pivot rows and columns of E_k. These contributions must be assembled before factorizing E_k. Active edges are assembled, removed from $\mathcal{F}^{[k]}$, and combined into inactive edges in $\mathcal{E}^{[k+g_k]}$. Thus,

$$\langle s, \text{row } i\rangle \in \mathcal{F}_L^{[k]} \wedge i \in \mathcal{L}_k' \;\rightarrow\; \langle s, k\rangle \in \mathcal{E}_L^{[k+g_k]} \tag{8}$$

and

$$\langle s, \text{column } j\rangle \in \mathcal{F}_U^{[k]} \wedge j \in \mathcal{U}_k' \;\rightarrow\; \langle s, k\rangle \in \mathcal{E}_U^{[k+g_k]}. \tag{9}$$

If both Equations 8 and 9 hold, then the entire $E_s^{[k]}$ is assembled into E_k (a *symmetric edge reduction*, see Section 4.2).

Pivot row and column nodes k through $k + g_k - 1$ and edges incident to these nodes are removed from $\mathcal{A}$. New active edges are added from node k in $\mathcal{G}$ to row and column nodes in $\mathcal{A}$ defined by $\mathcal{L}_k''$ and $\mathcal{U}_k''$, respectively. New fill-in edges are added (implicitly) to $\mathcal{A}$ in the same manner as fill-in from the first frontal matrix. The factorization of the frontal matrix E_k satisfies steps k through $k + g_k - 1$ of the LU factorization. The next step of LU factorization will be $k + g_k$.

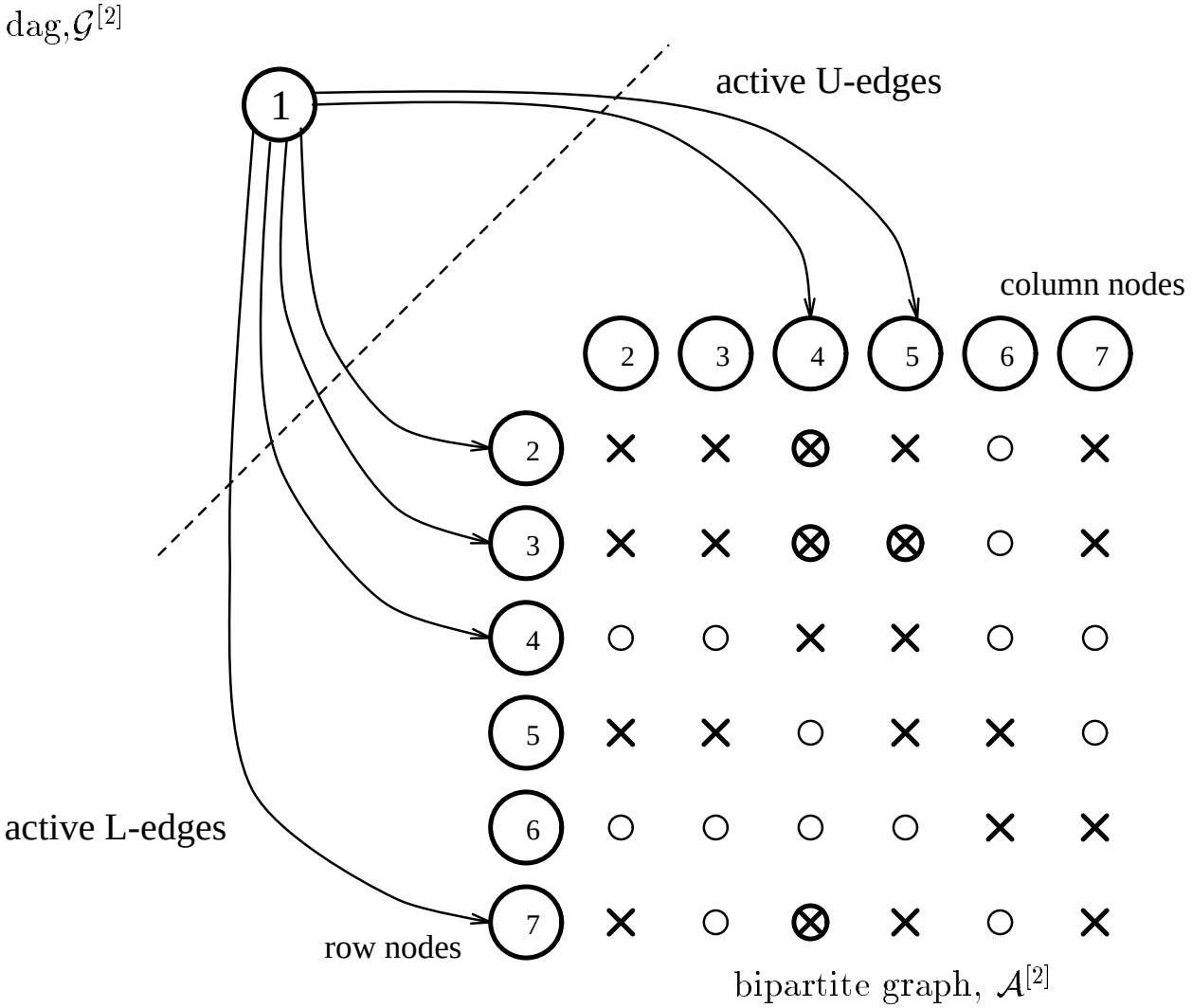

Figure 2: Assembly graph $\mathcal{D}^{[2]} = (\mathcal{A}^{[2]}, \mathcal{G}^{[2]}, \mathcal{F}^{[2]})$ for matrix A in Equation 6

The graphs $\mathcal{A}^{[k]}$ and $\mathcal{G}^{[k]}$ correspond to specific submatrices of the partial LU factors in Equation 1. $\mathcal{G}^{[k]}$ is an assembly dag for the $L^{[k]}_{1..k-1,1..k-1}$ and $U^{[k]}_{1..k-1,1..k-1}$ matrices. The active L-edges are a subset of the nonzero pattern of $L^{[k]}_{k..n,1..k-1}$, and the active U-edges are a subset of the nonzero pattern of $U^{[k]}_{k..n,1..k-1}$.

Figure 3 continues the example shown in Figure 2, where the second frontal matrix E_2 has been factorized. The single inactive L-edge $\langle 1,2\rangle$ in $\mathcal{G}^{[4]}$ is shown in bold (node 2 is an L-parent of its L-child node 1). This L-edge is due to the nonzero entries $l_{2,1}$ and $l_{3,1}$, and (at this point in the discussion) represents the assembly of two rows from the contribution block of E_1 into the pivot rows of E_2. The resulting bipartite graph $\mathcal{A}^{[4]}$ is also shown.

Figure 4 shows the graph $\mathcal{D}^{[5]} = (\mathcal{A}^{[5]}, \mathcal{G}^{[5]}, \mathcal{F}^{[5]})$ after the factorization of E_4. The inactive U-edge $\langle 2,4\rangle$ in $\mathcal{G}^{[5]}$ represents the assembly into E_4 of the contribution that E_2 makes to row 4. It is present because of the nonzeros $u_{2,4}$ and $u_{3,4}$. The inactive LU-edge $\langle 1,4\rangle$ represents the assembly into E_4 of the contribution of E_1 to both row and column 4. It is present because of the nonzeros $l_{4,1}$ and $u_{1,4}$. In the next section these inactive edges will be used to represent the assembly of additional contributions, and some of the active edges in Figures 3 and 4 will be removed.

4.2 Edge reductions

Potentially, every off-diagonal nonzero in the LU factors can define a single, unique edge in the assembly dag $\mathcal{G}$. Some of the these potential edges are never created, due to the no-fill amalgamation described in the previous section. Additional edge reductions are described in this section. Some of these are in the style of Eisenstat, Liu, and Gilbert [18, 23], except that we apply them to our partially constructed graph when the pivot order is not fully known. A class of edge reductions that goes beyond transitive reduction is applied during the approximate degree update phase described in Section 7.2.

Up to this point in the discussion, an active L-edge $\langle s, \text{row } i\rangle \in \mathcal{F}_L$ (or an active U-edge $\langle s, \text{column } j\rangle \in \mathcal{F}_U$) represents an unassembled contribution from E_s to row i (or column j) of the active submatrix. This contribution remains in E_s until row i becomes pivotal in E_t, or until E_s is assembled into its LU-parent E_k (where $s < k < t$). With no additional edge reductions, contributions of a frontal matrix remain unassembled until the latest possible step of the factorization. A frontal matrix persists until its contribution block is completely assembled into subsequent frontal matrices. Edge reductions improve the memory requirements of the method by assembling these contributions at the earliest possible step. They also simplify the pivot search and approximate degree update described in Section 7.

Two types of simple edge reductions, L-child and U-child reductions, can be applied at step k. For an L-child edge reduction, consider the lower triangular part of the submatrix formed from rows and columns s to $s + g_s - 1$, k to $k + g_k - 1$, and i of $L^{[k+g_k]} \backslash A^{[k+g_k]}$, where s and

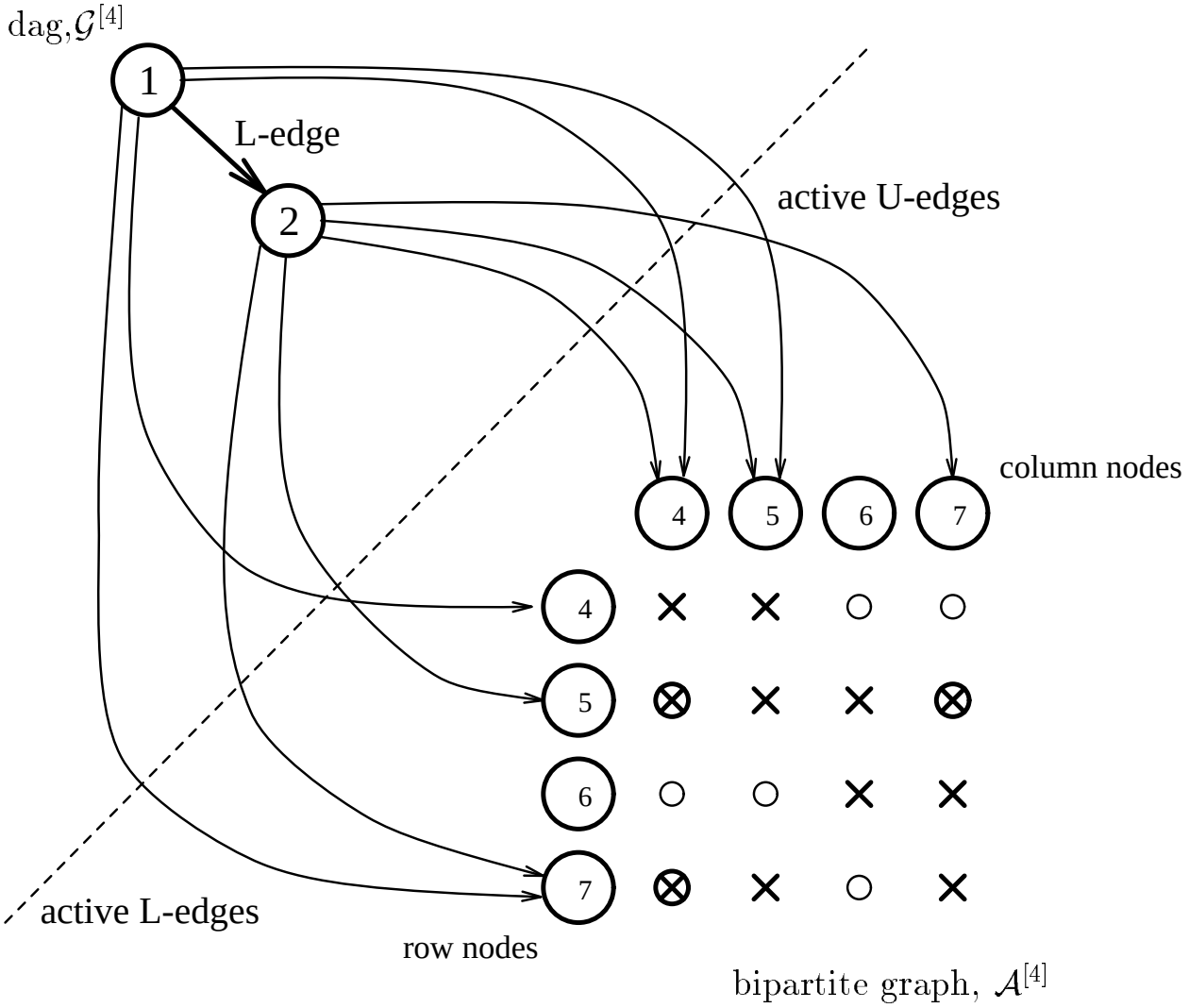

Figure 3: Assembly graph $\mathcal{D}^{[4]} = (\mathcal{A}^{[4]}, \mathcal{G}^{[4]}, \mathcal{F}^{[4]})$ for matrix A in Equation 6

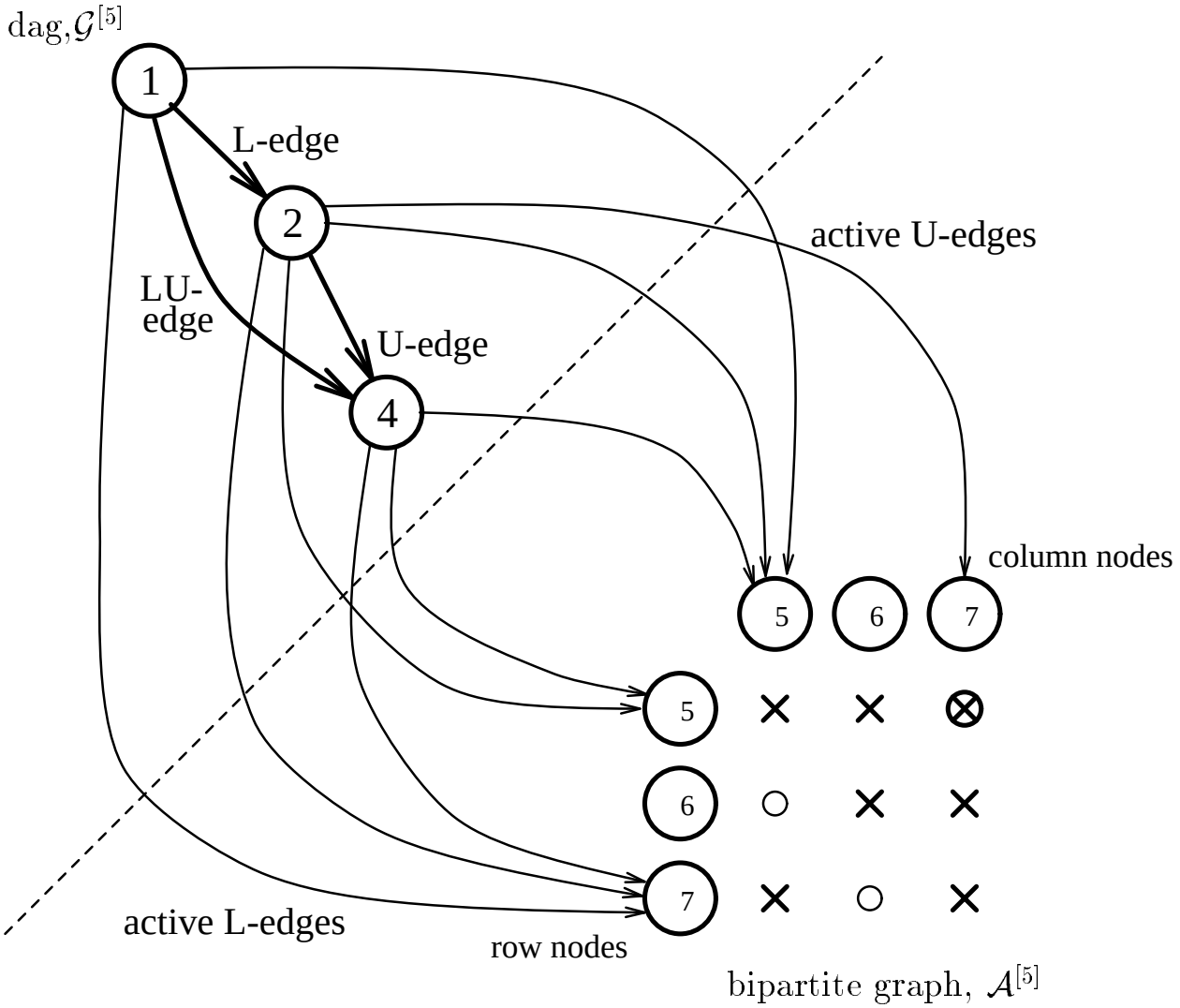

Figure 4: Assembly graph $\mathcal{D}^{[5]} = (\mathcal{A}^{[5]}, \mathcal{G}^{[5]}, \mathcal{F}^{[5]})$ for matrix A in Equation 6

k are nodes in $\mathcal{G}^{[k+g_k]}$ and $i \geq k + g_k$.

$$\begin{bmatrix} L'_s & & \\ L_{ks} & L'_k & \\ L_{is} & L_{ik} & a_{ii}^{[k+g_k]} \end{bmatrix} \tag{10}$$

This reflects the status of the factorization just before step $k + g_k$. It is not yet known when row i will become pivotal. In Equation 10, matrix $L_{k..k+g_k-1,s+g_s-1}$ is denoted as L_{ks}, $L_{i,s..s+g_s-1}$ is denoted as L_{is}, and $L_{i,k..k+g_k-1}$ is denoted as L_{ik}.

If $L_{ks} \neq 0$, then s is an L-child of k. Fill-in from E_s causes

$$\mathcal{U}_s^{[k]} \subseteq \mathcal{U}_k. \tag{11}$$

If E_s and E_k both contribute to a common row i, the contribution that element E_s makes to row i (with pattern $\mathcal{U}_s^{[k]}$) is assembled into E_k at step k *before* row i becomes pivotal. The active edge $\langle s, \text{row } i \rangle$ is removed, reducing the size of the active edge-set $\mathcal{F}_L$. The entry for row i is removed from $\mathcal{L}_s^{[k+g_k]}$. Row i can be found by scanning all the active edges incident on all rows in $\mathcal{L}_k''$. If an L-child of k is found in this scan, the active edge can be removed.

U-child edge reduction is the transpose of the L-child case. Both reductions are applied if a node s is an LU-child of a new node k, forming a *symmetric edge reduction*. In this case, the entire contribution block of E_s is assembled into E_k, and all active edges from node s are removed.

Removal of active edges also results in fewer inactive edges in $\mathcal{G}$. If all active L-edges from node s to pivot rows of E_k are removed before step k, then no inactive L-edge $\langle s, k \rangle$ appears in $\mathcal{G}$.

As an example, refer again to matrix A in Equation 6, and to Figure 3. When E_2 is created, its column pattern

$$\mathcal{U}_2 = \{2, 3, 4, 5, 7\}$$

is a superset of the pattern

$$\mathcal{U}_1^{[2]} = \{4, 5\}$$

because of fill-in. Row $i = 7$ is in both row patterns of $\mathcal{L}_1^{[2]}$ and $\mathcal{L}_2''$. Thus, the contribution that E_1 makes to row 7 is assembled into E_2 (before row 7 becomes pivotal). The active L-edge from node 1 to row 7 is assembled and deleted.

Similarly, in Figure 4, node $4 \in \mathcal{G}$ is a U-parent of node 2. Node 2 contributes to column 5; this contribution is assembled and the active U-edge $\langle 2, \text{column } 5 \rangle$ is removed. Node 4 is also an LU-parent of node 1. All contributions from E_1 are assembled into E_4, and all active edges originating at node 1 are removed.

The edge reductions presented here do not result in a graph, $\mathcal{G}$, with the minimal number of edges. However, we can perform these edge reductions as a byproduct of the approximate

degree update phase of the analysis-factor algorithm, because the active edges terminating at rows and columns affected by a newly factorized E_k are scanned to compute bounds on the number of nonzeros in those rows and columns of the active submatrix. Transitive reduction [1] could remove more edges, but would be too costly.

4.3 Additional amalgamation

No-fill amalgamation has already been described. Additional amalgamation can improve the unsymmetric-pattern multifrontal method by increasing the ratio of Level-3 BLAS to Level-2 BLAS operations and by simplifying the symbolic computations. If a parent node k and child node s in $\mathcal{G}$ are amalgamated into a single supernode r, the amalgamated element E_r has row pattern $\mathcal{L}_s \cup \mathcal{L}_k$ and column pattern $\mathcal{U}_s \cup \mathcal{U}_k$.

If k is an LU-parent of s, then extra fill-in can occur only in the pivot rows and columns of E_s, because

$$\mathcal{U}_s^{[k]} \subseteq \mathcal{U}_k$$

and

$$\mathcal{L}_s^{[k]} \subseteq \mathcal{L}_k.$$

Rows and columns that are pivotal through step $k + g_k - 1$ are removed from these patterns, maintaining the subset relationship:

$$\mathcal{U}_s^{[k+g_k]} \subseteq \mathcal{U}_k^{[k+g_k]} \tag{12}$$

$$\mathcal{L}_s^{[k+g_k]} \subseteq \mathcal{L}_k^{[k+g_k]} \tag{13}$$

as shown in Figure 5 (where $k' = k + g_k$).

If k is only an L-parent of s, then the two row patterns $\mathcal{L}_k$ and $\mathcal{L}_s$ are unrelated (Equation 12 holds, but not Equation 13), as shown in Figure 6 (where $k' = k + g_k$). Fill-in can occur in pivot columns $\mathcal{U}'_k$ and $\mathcal{U}'_s$, and in pivot rows $\mathcal{L}'_s$, but not in pivot rows $\mathcal{L}'_k$. Fill-in can also occur in the contribution block of E_r, specifically, the submatrix defined by columns $\mathcal{U}_k^{[k+g_k]} \backslash \mathcal{U}_s^{[k+g_k]}$ and rows $\mathcal{L}_s^{[k+g_k]} \backslash \mathcal{L}_k^{[k+g_k]}$ (outlined with a dashed box in Figure 6). Amalgamation between s and a U-parent node k is the transpose of the L-parent case.

When two nodes k and s are amalgamated into a new node r, the edge $\langle s, k \rangle \in \mathcal{E}$ is removed, any edges that terminated at either k or s now terminate at r, and any edges that start at either k or s now start at r. Duplicate edges are combined.

5 Analysis-only algorithm

In the symbolic analysis phase, we wish to use a sparsity preserving heuristic to generate an ordering and symbolic factorization information so that a subsequent numerical factorization

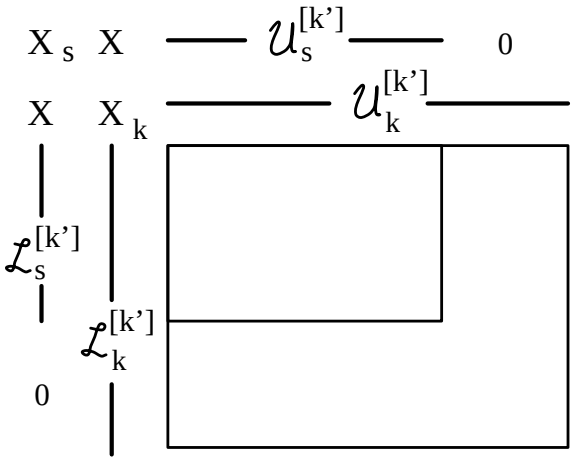

Figure 5: Fill-in due to amalgamation between LU-child and LU-parent

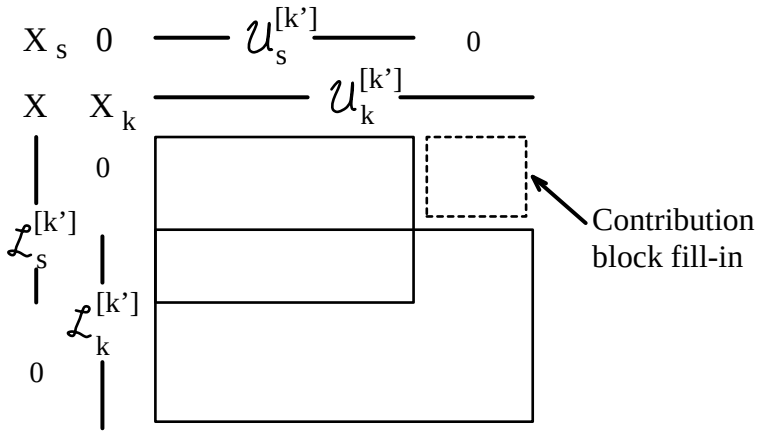

Figure 6: Fill-in due to amalgamation between L-child and L-parent

can use this information to effect an efficient decomposition. Thus, in this symbolic *analysis-only* algorithm, the values of the nonzeros are not taken into account and only the sparsity structure of the matrix is considered. We did start to design such an analysis phase but were not convinced of its utility because of the problems with perturbing the data structures that we mention in Section 6. However, we can apply recent work of Duff and Reid [17] based on algorithms developed by Duff, Gould, Reid, Scott, and Turner [11] to obtain a suitable analysis. We discuss the use of their algorithms in this section. Methods based on the elimination dag are presented in [18, 23].

Duff et al. [11] design algorithms for factorizing symmetric indefinite matrices which use block pivots of order 1 or 2, chosen from the diagonal to preserve symmetry, and are suitable even when there are zero entries on the diagonal. In particular, they have tested their codes on augmented matrices of the form

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix}$$

which arise from the solution of least-squares problems. They also consider the more general case where the upper left identity matrix is replaced by a general symmetric matrix, H, say corresponding to the augmented matrix that occurs when solving constrained nonlinear programming problems. The strategy used to select pivots in their symmetric analysis is that of minimum degree, generalized to handle 2×2 pivots of the form

$$\begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix} \text{ (full pivots),}$$

$$\begin{bmatrix} \times & \times \\ \times & 0 \end{bmatrix} \text{ (tile pivots), and}$$

$$\begin{bmatrix} 0 & \times \\ \times & 0 \end{bmatrix} \text{ (oxo pivots).}$$

Now, if we consider the augmented system

$$\begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix} \tag{14}$$

where B is a nonsingular unsymmetric matrix of order n, the first n components of the solution are just the solution of the set of unsymmetric linear equations

$$Bx = b.$$

Furthermore, if the algorithm of Duff et al. [11] is used to choose pivots from the coefficient matrix of Equation 14, then n oxo pivots will be chosen. In addition, their generalization of the minimum degree criterion will ensure that the off-diagonal entries of the oxo pivot will be the same as the entry in B (and B^T) with the lowest Markowitz count. Thus we can use

the symmetric minimum degree ordering of Duff et al. [11] to obtain a symbolic analysis of an unsymmetric matrix. The code of Duff and Reid [17] will also produce the equivalent of our directed acyclic graph ($\mathcal{G}$) which could, after suitable modification, be used as input to the factor-only algorithm of this paper. Because numerical values were not taken at all into account in the analysis phase, we do not, however, recommend this route because of the large amount of modification to the resulting directed acyclic graph by the subsequent numerical factorization phase.

6 Factor-only algorithm

This section describes the *factor-only* algorithm, an unsymmetric-pattern multifrontal method that factorizes A into LU using the patterns of the LU factors and the final assembly dag, $\mathcal{G}^{[n+1]}$, computed in the analysis-only or analysis-factor algorithm. Numerical pivoting is performed in this phase, so it must be able to handle changes in the predicted patterns of L and U.

The work at node i consists of the following (for $i \in \mathcal{V}$):

1. Wait until all children of node i in $\mathcal{G}$ have finished.
2. Create E_i and assemble into it the contributions represented by the edges in $\mathcal{G}$ that terminate at node i. These are the L-children, U-children, and LU-children of node i. Deallocate elements of children that no longer have unassembled contributions. The row and column pattern of E_i is $\mathcal{L}_i$ and $\mathcal{U}_i$, which were precomputed in the symbolic phase (although they might change due to numerical pivoting considerations as described below).
3. Factorize the pivot block F_i into $L_i' \backslash U_i'$, and compute the block row U_i'' of U and block column L_i'', overwriting them in E_i. Then store them in a separate, statically-allocated data structure for L and U. The symbolic factorization predicted g_i pivots, but this number might be less due to pivot failures within F_i, or more due to pivot failures in children of node i. Set g_i to the number of pivots actually found.
4. Compute the update terms with a rank-g_i update to the contribution block D_i, using the Level-2 BLAS if $g_i = 1$, or Level-3 if $g_i > 1$. Parallelism can occur within these kernels, as well as between independent nodes [24].
5. If node i is the last child to complete for a parent j then enable node j.

Numerical pivoting considerations might not allow the expected number of pivots to be chosen from a pivot block F_i. The work associated with the failed pivots must be performed later. This can be regarded as a forced amalgamation of the failed pivots with one or more parents of i in $\mathcal{G}$ with any fill-in constraints removed.

In the classical multifrontal method, the failed pivots are amalgamated with the single parent node j of i. The g_j steps of LU factorization in node j and that of any failed pivots of its children are attempted. Numerical pivoting causes only local changes in the assembly tree and in the patterns of L and U, although these changes can ripple up if the pivots also fail in the parent node. The changes are limited to the failed pivot rows and columns. This case also occurs in the unsymmetric-pattern multifrontal method if node i has a single LU-parent and no L-parents or U-parents. Otherwise, larger disruptions can occur.

If a node i has either L-parents or U-parents in $\mathcal{G}$ and is found to have numerically unacceptable pivots, the effects are not limited to a single pair of nodes. In the following example, node j is the L-parent of a simple node i with a single failed pivot,

$$\begin{bmatrix} p_i & \cdot & u_{ik} \\ l_{ji} & p_j & u_{jk} \\ l_{ki} & \cdot & p_k \end{bmatrix}.$$

One option is to amalgamate nodes i and j, as was done for numerical pivoting failures in the classical multifrontal method. However, this causes fill-in in the contribution block of E_j, since the pivot rows are not likely to be identical (see Figure 6 and the related discussion). This fill-in causes far-reaching effects in $\mathcal{G}$ in any ancestors of i. Catastrophic fill-in and loss of parallelism can result.

The second option for recovering from numerical pivoting failures is to amalgamate node i with its single LU-parent k, assuming it exists. This has the effect of reordering the matrix so the pivot p_i follows p_k,

$$\begin{bmatrix} p_j & u_{jk} & u_{ji} \\ \cdot & p_k & u_{ki} \\ \cdot & l_{ik} & p_i \end{bmatrix}.$$

Limited fill-in occurs in the intermediate nodes between node i and node k that are L-parents and U-parents of node i. In this example, the column pattern $\mathcal{U}_j$ of element E_j is augmented by including the failed pivot column i (because of entry u_{ji}). The fill-in in column i from E_j (assuming p_j is not a column singleton) is

$$\mathcal{L}_i \leftarrow \mathcal{L}_i \cup \mathcal{L}''_j.$$

If node j was instead a U-parent of node i, then the row pattern $\mathcal{L}_j$ of element E_j would be augmented by the single failed pivot row i. The fill-in in row i from E_j would be

$$\mathcal{U}_i \leftarrow \mathcal{U}_i \cup \mathcal{U}''_j.$$

Nodes i and j are not amalgamated, so catastrophic fill-in does not occur in the contribution block of E_j, as in the first option. Instead, fill-in in any contribution block is limited to row or column i. Fill-in also occurs in both row and column i when it is amalgamated with k.

In general, node i is shifted past each intervening L-parent and U-parent node, augmenting each L-parent by column i and each U-parent by row i. Let m be the final position of pivot

i in the modified pivot order. Each L-parent j causes fill-in with pattern $\mathcal{L}_j^{[m]}$ in column i, and each U-parent j causes fill-in with pattern $\mathcal{U}_j^{[m]}$ in row i. Finally, node i is amalgamated with its single LU-parent, node k, augmenting both $\mathcal{L}'_k$ and $\mathcal{U}'_k$ by i (where $k \leq m < k + g_k$). Amalgamation with the LU-parent k does not cause fill-in in the contribution block of E_k because $\mathcal{L}_k$ already contains the pattern $\mathcal{L}_j^{[m]}$ of any L-parent node j of i, even without amalgamation. This can be seen in the example above (prior to reordering). If j is an L-parent of i, and k is an LU-parent of i, then u_{jk} is nonzero (because of fill-in) and $\mathcal{L}_j^{[m]} \subseteq \mathcal{L}_j^{[k]} \subseteq \mathcal{L}_k$. Similarly, $\mathcal{U}_j^{[m]} \subseteq \mathcal{U}_j^{[k]} \subseteq \mathcal{U}_k$, for any U-parent node j of i.

The graph $\mathcal{G}$ is only locally modified, but this option breaks down if node i does not have an LU-parent k. In this case, node i is delayed until its last L-parent or U-parent. Fill-in is caused in row and column i by every ancestor of node i until that point. The failed pivot row and column might easily become dense, leading to a high level of fill-in if many numerically unacceptable pivots are encountered.

To summarize, the factor-only algorithm presented in this section can take advantage of the Level-3 BLAS to factorize a sparse matrix with an unsymmetric pattern. This method, however, is "fragile" with respect to numerical pivoting perturbations in the numerical phase. The graphs and the patterns of L and U can change drastically from those found by the symbolic phase. The changes are less drastic in the assembly tree and the patterns of L and U for the classical multifrontal method. Limiting the perturbations caused by numerical pivoting is the most important open problem facing the development of a practical factor-only algorithm, and we have suggested a possible first step in this direction. The next section presents an alternative that bypasses this problem by combining the symbolic and numerical phases into a single phase.

7 Analysis-factor algorithm

Combining the symbolic and numerical factorization into a single phase is more typical of conventional factorization algorithms for unsymmetric sparse matrices. The advantage is that the numerical values are available during the pivot search. No pivot preordering is assumed. Pivots are chosen as the algorithm progresses via some sparsity preserving and numerical criteria. Unsymmetric permutations are allowed, and probably required, since no assumption is made about a zero-free diagonal or the positive-definiteness of the original matrix. The disadvantage to this approach is the lack of a precomputed assembly dag to guide the construction of new elements, the assembly process, and the exploitation of parallelism. The algorithm must compute the graph $\mathcal{D}$ and the patterns of L and U during factorization. Some form of the active submatrix must be maintained to allow for arbitrary pivoting and the degree of each row and column of the active submatrix must be maintained.

Three approaches for performing additional amalgamation are described below (the *AFup*, *AFdown*, and *AFstack* algorithms). Before using any of these algorithms, we perform

an (optional) preprocessing stage that scales the matrix and permutes it to upper-block-triangular form (the diagonal blocks are factorized independently) [10]. The performance of these algorithms is presented in Section 8.

7.1 Data structures

All data structures are allocated out of a pair of one-dimensional real and integer arrays. The original matrix A is stored in both row and column form, followed by various workspaces and data structures of size n. The two arrays hold a stack containing the pattern and numerical values of the LU factors.

Portions of $\mathcal{D} = (\mathcal{A}, \mathcal{G}, \mathcal{F})$ are held implicitly. The bipartite graph $\mathcal{A}^{[k]}$ is an implicit representation of the active submatrix. The active submatrix $A^{[k]}_{k..n,k..n}$ is represented as the unassembled entries of the active part of A and all unassembled portions of contribution blocks (see Equation 5). The dag, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, is explicitly stored with the LU factors. The edge set $\mathcal{E}$ is not needed in the analysis-factor algorithm, but it is required as input to the factor-only algorithm.

The active L-edges in $\mathcal{F}_L^{[k]}$ are held as a set of *L-child lists*. Similarly, the active U-edges in $\mathcal{F}_U^{[k]}$ are held as a set of *U-child lists*. The L-child list for a row i in $\mathcal{A}^{[k]}$ holds a set of tuples,

$$\text{L-child list } i = \{(t, f) \mid \langle t, \text{row } i\rangle \in \mathcal{F}_L^{[k]} \wedge f = \text{ offset of row } i \text{ in } E_t\},$$

one for each active L-edge that terminates at row i (similarly, U-child list j holds tuples for active U-edges that terminate at column j of $\mathcal{A}^{[k]}$). A tuple (t, f) in an L-child list i contains a reference to a node t, and an offset f to the contribution to row i in E_t.

The space between the original matrix and fixed arrays and the LU factors holds the frontal matrices and L-child and U-child lists. The frontal matrices and the L-child and U-child lists are allocated when elements are created and deallocated when their corresponding contribution blocks have been completely assembled. These form the main dynamic data structures in the algorithm.

7.2 Pivot search, degree update, and edge reductions

The three algorithms use a similar global pivot search strategy and degree update phase (although AFstack combines the numerical assembly with this phase), and perform similar edge reductions.

The pivot search is based on Markowitz' strategy [27], which selects the pivot $a_{ij}^{[k]}$ with minimum upper bound on fill-in (or *cost*),

$$(r_i^{[k]} - 1)(c_j^{[k]} - 1). \tag{15}$$

The rows and columns of the active matrix are not held explicitly, rather, they are held as a set of contribution blocks and entries from the original matrix A. Scanning a row i of $A^{[k]}_{k..n,k..n}$ involves scanning row i of the active part of A and tuples in the L-child list i. Adding these terms gives the pattern and numerical values of row i in $A^{[k]}_{k..n,k..n}$. Columns are scanned similarly. Many candidate pivots will need to be searched, so this is an expensive operation for only calculating the degree. To avoid this, only upper and lower bounds of the degree of each row and column are computed. Upper and lower bounds of a degree r are denoted as $upper(r)$ and $lower(r)$, respectively. The initial upper and lower bounds are simply the true degrees of the rows and columns of A. If the true degree is calculated during factorization, the two bounds are set equal to the true degree. Only the first few columns with minimum upper bound degree are searched (not unlike the truncated pivot search options in [14] and [31]), and the true degrees of these columns are computed. The pivot search is assisted by a set of n linked lists. The d-th linked list holds those columns with upper bound degree d, that is $\{j \mid upper(c^{[k]}_j) = d\}$.

The pivot $a^{[k]}_{ij}$ at step k must also satisfy the threshold partial-pivoting criterion [10]:

$$|a^{[k]}_{ij}| \geq u \cdot \max_{k \leq s \leq n} |a^{[k]}_{sj}|, \;\; 0 < u \leq 1. \tag{16}$$

The candidate pivot is the numerically acceptable entry $a^{[k]}_{ij}$ with lowest approximate Markowitz cost using the true column degree and the upper bound row degree,

$$(upper(r^{[k]}_i) - 1)(c^{[k]}_j - 1). \tag{17}$$

The approximate degree update finds the upper and lower bounds of the degrees of each row $i \in \mathcal{L}''_k$ and column $j \in \mathcal{U}''_k$ by scanning their L-child and U-child lists, respectively. The new lower bound on the row degree, for example, cannot be smaller than the upper bound on fill-in in row i or the maximum number of unassembled columns of each contribution block affecting row i (these can be found in the L-child list i). The new upper bound on the row degree can be computed in a similar way. Only a short scan of each L-child list and U-child list in the affected rows ($\{i \mid i \in \mathcal{L}''_k\}$) and columns ($\{j \mid j \in \mathcal{U}''_k\}$) suffices. The lists are kept short via edge removal. The time taken for this scan is linear in the sum of the sizes of the L-child/U-child lists of the affected rows and columns.

However, at the cost of an additional scan of either the affected L-child or U-child lists, more accurate degree bounds can be computed, based on the *external* row and column degrees of previous frontal matrices with respect to the current frontal matrix, E_k. The time taken is asymptotically unchanged.

The column pattern $\mathcal{U}^{[k]}_m$ of an unassembled contribution block $E^{[k]}_m$ divides into two disjoint subsets with respect to $\mathcal{U}_k$: *internal* entries

$$\hat{\mathcal{U}}_{mk} = \mathcal{U}^{[k]}_m \cup \mathcal{U}_k$$

and *external* entries

$$\check{\mathcal{U}}_{mk} = \mathcal{U}_m^{[k]} \backslash \mathcal{U}_k$$

and thus, by definition,

$$\mathcal{U}_m^{[k]} = \hat{\mathcal{U}}_{mk} \cup \check{\mathcal{U}}_{mk}.$$

An external entry $j \in \check{\mathcal{U}}_{mk}$ is an unassembled column of E_m that is not present in the column pattern of the current frontal matrix E_k. Let the internal and external row degree of E_m with respect to E_k be denoted as

$$\hat{r}_{mk} = |\hat{\mathcal{U}}_{mk}|$$

and

$$\check{r}_{mk} = |\check{\mathcal{U}}_{mk}|$$

respectively. Thus,

$$\check{r}_{mk} = |\mathcal{U}_m^{[k]}| - \hat{r}_{mk}. \tag{18}$$

The external and internal column degrees are defined similarly, based on $\mathcal{L}_m$ and $\mathcal{L}_k$. Note that the external row and column degrees are not affected by the assembly of contributions from E_m into the current frontal matrix E_k. Assembly can decrease the internal degrees.

Each U-child list for columns $j \in \hat{\mathcal{U}}_{mk}$ contains a tuple (m, f) (where f is the offset of where column j appears in E_m). The internal degree, $\hat{r}_{mk}$, can be computed by simply counting how many times node m appears in the U-child lists for columns $j \in \mathcal{U}_k$. This count can be done for all frontal matrices affecting E_k in a single scan of all U-child lists for columns $j \in \mathcal{U}_k$. The external row degree of E_m can then be computed with Equation 18.

The new lower bound degree of row i ($lower(r_i^{[k+g_k]})$), then, is the *largest* of the following:

1. $r_i^{[k+g_k]} \geq lower(r_i^{[k]}) - g_k$, since the degree cannot drop by more than the number of pivots applied to row i,
2. $r_i^{[k+g_k]} \geq |Struct\ (M_{i*}^{[k+g_k]})|$, which is the number of unassembled entries in row i of the active part of A (see Equation 4), and
3. the number of columns in the contribution block of E_k plus the maximum external row degree of the frontal matrices in the L-child list for row i. That is,

$$r_i^{[k+g_k]} \geq |\mathcal{U}_k''| + \max_{\langle m,\ \text{row } i\rangle \in \mathcal{F}_L^{[k+g_k]}} \check{r}_{mk}.$$

   This lower bound is tight if the column patterns of all the external entries of frontal matrices affecting row i of E_k,

$$\check{\mathcal{U}}_{mk}\ \forall\{m \mid \langle m, \text{row } i\rangle \in \mathcal{F}_L^{[k+g_k]}\},$$

   are all subsets of the largest such column pattern.

The new upper bound degree of row i ($upper(r_i^{[k+g_k]})$) can be computed in a similar way. It is the *smallest* of the following:

1. $r_i^{[k+g_k]} \leq upper(r_i^{[k]}) + |\mathcal{U}_k''|$, since the row degree cannot increase by more than the upper bound on the fill-in in row i,

2. $r_i^{[k+g_k]} \leq n - k - g_k + 1$, which is the size of the active submatrix after E_k is factorized, and

3. the number of columns in the contribution block of E_k plus the sum of the external row degrees of the frontal matrices in the L-child list for row i. That is,

$$r_i^{[k+g_k]} \leq |\mathcal{U}_k''| + \sum_{\langle m,\ \text{row } i\rangle \in \mathcal{F}_L^{[k+g_k]}} \check{r}_{mk}.$$

   This upper bound is tight if the column patterns of all the external entries affecting row i are disjoint.

An approximate degree update (and edge reduction) based on external row and column degrees requires one extra scan of either the U-child lists of the affected columns $j \in \mathcal{U}_k$ or the L-child lists of the affected rows $i \in \mathcal{L}_k$, as follows:

1. Scan the U-child lists for all columns $j \in \mathcal{U}_k$, and compute the external row degree for each frontal matrix E_m appearing in the lists. Perform edge reduction if E_m is a true U-child or LU-child.

2. Scan the L-child lists for all rows $i \in \mathcal{L}_k$. Compute both the external column degree for each frontal matrix E_m appearing in the lists, and the approximate degree bounds for each row i (using the external row degrees computed in the first scan). Perform edge reduction if E_m is a true or *effective* L-child or LU-child (see below).

3. Re-scan the U-child lists for all $j \in \mathcal{U}_k$, and compute the approximate degree bounds for each column i (using the external column degrees computed in the second scan). Perform edge reduction if E_m is an *effective* U-child or LU-child.

If $\check{r}_{mk} = 0$, then node m is an *effective* L-child of node k. Similarly, node m is an effective U-child of node k if $\check{c}_{mk} = 0$. When the affected L-child or U-child lists are scanned, any edge from a true or effective child of node k is removed, and the corresponding numerical contribution is assembled. Node m is an effective LU-child if $\check{r}_{mk} = \check{c}_{mk} = 0$, in which case the entire $E_m^{[k]}$ is assembled into E_k.

Edge reductions from effective children remove any edge that would have been removed by transitive reduction. Furthermore, these edge reductions take advantage of coincidental overlap between previous frontal matrices and the current one, and thus remove *even more*

edges than is possible by transitive reduction. Consider Equation 10 in Section 4.2. If $L_{ks} = 0$, then node s is not a true L-child of the node k, the frontal matrix currently being factorized. Equation 11 may still hold, by coincidence, even though E_s does not cause fill-in in the g_k pivot rows of E_k. In this case, node k is an effective L-father of node s. If both E_s and E_k contribute to a common row i, then the contribution from E_s is assembled into E_k, and the active L-edge $\langle s, \text{row } i \rangle$ is removed. If transitive reduction were applied, this edge might not be removed.

Edge reductions from true children are still useful (as described in Section 4.2), since the edges from these nodes can be removed in the first scan of the affected L-child and U-child lists, rather than in the second scan of either set of lists.

Our experiments into the approximate degree update show a moderate, acceptable increase in fill-in in exchange for a significant reduction in time when compared with computing the true degrees (see performance results in Section 8).

7.3 The AFup algorithm

The first approach (the *AFup* algorithm) selects a single pivot using the global Markowitz-like strategy described above, allowing a pivot to be selected from anywhere in the active submatrix. The new node in $\mathcal{G}$ is augmented by additional pivot entries that lie within the current element. The new node in $\mathcal{G}$ "grows upward" by considering nodes in $\mathcal{A}$ connected via active edges to the new node.

The AFup algorithm can be outlined as follows. Initially, $k = 1$.

1. The pivot search finds the pivot $a_{ij}^{[k]}$ in $A_{k..n,k..n}^{[k]}$ and interchanges rows i and k, and columns j and k. The pivot row and column define $\mathcal{U}_k$ and $\mathcal{L}_k$, respectively. The L-children, U-children, and LU-children of node k are now located in the L-child list k and U-child list k.

2. Perform degree update, and try to extend frontal matrix to include near super-rows and super-columns (rows and columns whose patterns can be considered identical with the first pivot row and column). The search is limited to pivot entries lying in the current element. It "looks upward" because it considers the amalgamation of a single child k with one or more of its (potential) parents. Perform edge reduction, and find the effective children of node k.

3. Allocate space for the numerical values of E_k, performing garbage collection if necessary.

4. Assemble the children of node k into E_k. Deallocate any elements whose contributions have been completely assembled. Let n_{row} and n_{col} be the number of rows and columns in the super-row i and super-column j, and let $g_k = \min(n_{row}, n_{col})$.

5. Perform up to g_k steps of numerical factorization within the front. Set g_k to the number of pivots actually found.

6. Update the L-child and U-child lists. Increment k by g_k and repeat until the matrix is factorized.

7.4 The AFdown algorithm

The upward-looking algorithm can sometimes lead to excessive fill-in due to its limited pivot search. The *downward-looking* approach (AFdown) can decrease fill-in by replacing the local search with a global search that looks at the entire active submatrix. The method constructs a set $\mathcal{S}$ of unfactorized frontal matrices, each of which is a candidate for amalgamation. These frontal matrices have been symbolically factorized, but not numerically factorized. The method "looks downward" because it considers the amalgamation of a single node f (corresponding to the latest pivot entry) with one or more of its unfactorized children (in $\mathcal{S}$). The pivot search finds a single pivot entry, corresponding to a new *proposed* node f in the partially constructed graph. This node is amalgamated with its unfactorized children and placed in $\mathcal{S}$, unless doing so would cause excessive fill-in, in which case the children that caused amalgamation to fail are factorized and removed from $\mathcal{S}$.

It is interesting to note that the pivot blocks of the nodes in $\mathcal{S}$ form a block diagonal matrix. Thus, the frontal matrices in $\mathcal{S}$ can be factorized in parallel.

If a proposed node f has one or more unfactorized children, the numerical test is only an estimate, since the numerical values of the unfactorized children are not available. The numerical criterion is also checked during the numerical factorization, and unacceptable pivots are rejected.

7.5 The AFstack algorithm

Both the AFup and AFdown algorithms make only a estimate of the numerical acceptability of additional pivots (2 through g_k) in a frontal matrix E_k. These pivots may then be delayed, since the numerical factorization of the frontal matrix performs the accurate numerical test *after* all g_k pivots in E_k have been selected. Delayed pivots cause additional fill-in and waste the work performed in combining those pivots into E_k. The third approach (the *AFstack* algorithm) is more suitable for matrices that experience excessive delayed pivoting in the AFup and AFdown algorithms. It is essentially the same as the AFup algorithm, except that the frontal matrix E_k for the new node k is allocated on a stack, so that it can "grow" while pivots are being included into node k. These additional pivot rows and columns are factorized as they are included, so that an accurate numerical check can be combined with the symbolic (fill-reducing) pivoting test. No pivots are delayed.

The AFstack algorithm typically out-performs the other two (see Section 8), and is thus presented in more detail in the following pseudo-code. Comments are enclosed in curly-brackets. Let $m \equiv |\mathcal{L}_k|$ and $n \equiv |\mathcal{U}_k|$, for the current m-by-n frontal matrix E_k. The number of factorized pivots contained in E_k is g_k (enumerated 1 through g_k). Garbage collection is performed as necessary. For simplicity, assume all variables are global.

procedure AFstack
$k \leftarrow 1$
$n_b \leftarrow$ block size for tradeoff between Level-2 and Level-3 BLAS
while ($k \leq$ size of sparse matrix A) **do**
{ create and factorize frontal matrix E_k }
call find_first_pivot_in_frontal_matrix
call extend_frontal_matrix
call factor_frontal_matrix
$k \leftarrow k + g_k$
endwhile { factorizing }
end procedure { AFstack }

procedure find_first_pivot_in_frontal_matrix
global pivot search { find $a_{ij}^{[k]}$, the k-th pivot in the factorization of A }
$\mathcal{L}_k \leftarrow$ *Struct* $(A_{*j}^{[k]})$ { row pattern of E_k }
$\mathcal{U}_k \leftarrow$ *Struct* $(A_{i*}^{[k]})$ { column pattern of E_k }
$m \leftarrow |\mathcal{L}_k|$ { E_k is m-by-n }
$n \leftarrow |\mathcal{U}_k|$
$\mathcal{L}same \leftarrow$ **false** { $\mathcal{L}same$ is true if the latest pivot does not change $\mathcal{L}_k$ }
$\mathcal{U}same \leftarrow$ **false** { $\mathcal{U}same$ is true if the latest pivot does not change $\mathcal{U}_k$ }
allocate frontal matrix E_k on a stack at the top of allocatable memory
$E_k \leftarrow 0$
{ An initial frontal matrix of size m-by-n is allowed to grow as large as }
{ (Gm)-by-(Gn), where G is a user-settable parameter controlling the tradeoff }
{ between fill-in and amalgamation (typically 2 to 3). }
$k_0 \leftarrow 0$ { D_k has been updated with the first k_0 pivots in E_k }
$g_k \leftarrow 1$ { number of pivots in E_k }
end procedure { find_first_pivot_in_frontal_matrix }

procedure extend_frontal_matrix

{ Local pivot search to extend the frontal matrix, which currently has just one pivot }

while (extending the current frontal matrix, E_k) **do**

do approximate degree update, and assemble from previous frontal matrices

delete edges in $\mathcal{F}$ that correspond to assembled contributions

{ some portions of update/assembly are skipped, depending on $\mathcal{L}same$ and $\mathcal{U}same$ }

if (**not** $\mathcal{L}same$) **then**

attempt to find a candidate pivot column, $j \in \mathcal{U}_k''$,

for the $(g_k + 1)$-st pivot of E_k

{ based on column degree only. No scan of the column is done. }

endif

$\mathcal{L}same \leftarrow$ **true** { assembly and degree update completed for current size of E_k }

$\mathcal{U}same \leftarrow$ **true**

while ($\mathcal{L}same$ **and** $\mathcal{U}same$) **do**

scale current pivot column using Level-1 BLAS { column g_k of E_k }

{ At this point, g_k pivots in E_k have been successfully factorized }

if ($g_k \bmod n_b = 0$) **then**

update D_k with pivots $k_0 + 1$ to g_k in E_k

{ Level-3 BLAS matrix multiply, rank-n_b update }

$k_0 \leftarrow g_k$

endif

$g \leftarrow g_k + 1$ { g is the leftmost column of E_k to be updated in factor_frontal_matrix }

if (no acceptable candidate pivot column, j, for $(g_k + 1)$-st pivot of E_k) **then**

return { amalgamation of node k has proceeded as far as possible }

endif

swap candidate column j_c with column $g_k{+}1$ of E_k

{ where column j_c of E_k holds the contribution of E_k to column j of A }

update pivot column $g_k + 1$ with previous pivots $k_0 + 1$ to g_k {Level-2 BLAS }

$g \leftarrow g_k + 2$ { g is the leftmost column of E_k to be updated in factor_frontal_matrix }

attempt to find a pivot row, $i \in \mathcal{L}_k''$, such that $a_{ij}^{[k]}$ is an acceptable pivot

{ pivot row search is both symbolic (Equation 17) and numerical (Equation 16) }

if (no acceptable candidate pivot row, i, for $(g_k + 1)$-st pivot of E_k) **then**

return { amalgamation of node k has proceeded as far as possible }

endif

swap pivot row i_c with row $g_k{+}1$ of E_k

{ where row i_c of E_k holds the contribution of E_k to row i of A }

{ entry $a_{ij}^{[k+g_k]}$ is the $(g_k + 1)$-st pivot in E_k, }

{ and the $(k + g_k)$-th pivot in the factorization of A }

update pivot row $g_k + 1$ with previous pivots $k_0 + 1$ to g_k { Level-2 BLAS }

{ the pivot row update excludes the pivot itself }

$\mathcal{L}same \leftarrow (\mathcal{L}_k = \mathcal{L}_k \cup \mathit{Struct}\ (A_{*j}^{[k+g_k]}))$

if $(\mathcal{L}same)$ **then**
{ new pivot column adds no new rows to $\mathcal{L}_k$, do cheap degree update }
decrement upper and lower bounds of degrees of each column $j \in \mathcal{U}''_k$
find a good candidate pivot column $j \in \mathcal{U}''_k$ for next iteration, $g_k + 2$
{ based on column degree only. No scan of the column is done. }
endif
$\mathcal{U}same \leftarrow (\mathcal{U}_k = \mathcal{U}_k \cup Struct\ (A_{i*}^{[k+g_k]}))$
if $(\mathcal{U}same)$ **then**
{ new pivot row adds no new columns to $\mathcal{U}_k$, do cheap degree update }
decrement upper and lower bounds of degrees of each row in $i \in \mathcal{L}''_k$
endif
$\mathcal{L}_k \leftarrow \mathcal{L}_k \cup Struct\ (A_{*j}^{[k+g_k]})$
$\mathcal{U}_k \leftarrow \mathcal{U}_k \cup Struct\ (A_{i*}^{[k+g_k]})$
$g_k \leftarrow g_k + 1$
endwhile { finding pivots without extending frontal matrix }
zero the newly extended portions of E_k, on the stack
endwhile { finding pivots in local pivot search }
end procedure { extend_frontal_matrix }

procedure factor_frontal_matrix
{ The current m-by-n frontal matrix E_k has been extended to include g_k pivots.

$$E_k = \begin{bmatrix} L'_k \backslash U'_k & U''_k \\ L''_k & D_k \end{bmatrix}$$

where $L'_k \backslash U'_k$ is g_k-by-g_k, U''_k is g_k-by-$(n - g_k)$, L''_k is $(m - g_k)$-by-g_k, and D_k is $(m - g_k)$-by-$(n - g_k)$. The contribution block D_k has been updated for pivots 1 through k_0. It is not updated for pivots $k_0 + 1$ to g_k, except perhaps for the first column of D_k (depending on g). }
if $(g_k \neq k_0)$ **then**
update D_k with pivots $k_0 + 1$ to g_k { starting at column g of E_k }
{ Level-3 BLAS matrix multiply, rank-b update, where $1 \leq b < n_b$ }
endif
save the pivot rows and columns in the LU arrowhead { compress on the stack }
copy D_k to a new location allocated from the tail end of memory
{ D_k will be deallocated when fully assembled into subsequent frontal matrices }
make an explicit list of the children of node k, and place in LU data structure
{ These are inactive edges in $\mathcal{E}$, which may be used in a factor-only algorithm. }
add active L-edges $\{\langle k, \text{row } i\rangle \mid i \in \mathcal{L}''_k\}$ to $\mathcal{F}_L$
add active U-edges $\{\langle k, \text{column } j\rangle \mid j \in \mathcal{U}''_k\}$ to $\mathcal{F}_U$
end procedure { factor_frontal_matrix }

The AFstack algorithm has available to it the numerical values of the current frontal matrix, E_k, when it performs degree update and edge reduction. The degree update, edge reductions, and numerical assembly can all be done in a single phase (procedure extend_frontal_matrix). The AFup and AFdown algorithms split this work into two separate symbolic and numerical phases (for each frontal matrix), resulting in a duplication of work. Note that in AFstack, the assembly phase is skipped and degree update phase is simplified if the new pivot does not extend the pattern of the frontal matrix.

The local pivot search attempts to find a candidate pivot column j based purely on symbolic (sparsity-preserving) considerations. Once it is located, the AFstack algorithm updates the candidate column with previous pivots (a Level-2 BLAS matrix-vector multiply), and then searches the column j for an acceptable pivot, finding the row i with lowest upper bound degree ($upper(r_i)$) such that a_{ij} is numerically acceptable. No delayed pivots occur because the algorithm checks the numerical acceptability of candidate pivots as it selects them. Only one candidate column is updated and searched, since this could be an expensive operation. Examining all entries in E_k, for instance, would require an immediate Level-2 BLAS update of the entire contribution block. This would defeat the use of the Level-3 BLAS in the extend_frontal_matrix procedure (although the Level-3 BLAS would still be used in the factor_frontal_matrix procedure).

If a pivot is found, the candidate row is updated with a matrix-vector multiply. As soon as sufficient pivots have accumulated (n_b), the entire contribution block is updated with a matrix-multiply operation (a rank-n_b update). Thus, the Level-2 BLAS updates of individual candidate rows and columns require no more than a rank-n_b matrix-vector multiply. The block-size parameter n_b can be adjusted depending on the relative speed of the Level-3 BLAS versus the Level-2 BLAS on a given computer. On the Cray YMP, with $n_b = 16$, most of the floating-point operations are performed using Level-3 BLAS subroutines (between 50% and 96%, depending on the sparsity of the matrix).

7.6 Summary of the analysis-factor algorithm

Each version of the analysis-factor algorithm is based on the assembly graph, $\mathcal{D}$, that guides the pivot search, the construction of new elements, the assembly process, the detection of super-rows and super-columns, and the degree update. Edge reductions and dynamic amalgamation keep this graph pruned. Dynamic amalgamation allows the algorithms to take advantage of the Level-3 BLAS. The analysis-factor algorithms do not suffer from the disruptions in the graphs or in the patterns of L and U caused by numerical pivoting in the factor-only algorithm. Parallelism is not yet addressed; this and other issues are considered in Section 9, which presents open problems and future work. The following section compares the performance of the three sequential versions of the analysis-factor algorithm with that of the classical multifrontal method (Mups), the D2 algorithm, and the MA28 algorithm.

8 Performance results

We have developed three prototype sequential versions of the unsymmetric-pattern multifrontal method, and have tested them extensively on a Cray-YMP. In this section, we compare their performance with the MA28 algorithm [14], sequential versions of the classical multifrontal method (Mups) [2], and the D2 algorithm [5].

Table 1 summarizes our results [3] for eighty-six matrices from the Harwell/Boeing collection [12, 13] and other sources. Twenty-seven of these are symmetric positive-definite. Only matrices of order 500 or larger were considered. The Z matrices are chemical engineering problems from S. Zitney and others [29] (Z/m2 is from a PDE). The Hm matrices are circuit simulation matrices from S. Hamm (Motorola). Table 1 lists the results for these matrices obtained on a Cray YMP-8/128, sorted by asymmetry (and by order if tied). Each line lists the matrix number, name, order, number of nonzeros, and asymmetry. The asymmetry, s, is the number of unmatched off-diagonal pairs over the total number of off-diagonal entries (0 is a symmetric pattern, 1 is completely asymmetric). The run time includes both the analysis and factorize time, in seconds, and is listed for the AFstack, AFdown, AFup, Mups, D2, and MA28 algorithms. The fastest run time is shown in bold.

Of the three versions of the unsymmetric-pattern multifrontal method, the AFstack algorithm typically out-performs the other two. It is faster than both AFdown and AFup for 63 out of 86 matrices. When AFstack is slower than the other two algorithms, its run time is no more than 1.3 times the run time of the faster of AFdown and AFup. The sum of the run times of AFstack, AFdown, and AFup for all 86 matrices is 96.8, 154.0 and 161.3 seconds, respectively, whereas the sum of the *best* run time for each matrix (for these three algorithms) is 94.0 seconds. The AFstack algorithm is clearly superior to AFdown and AFup.

Figures 7 through 9 show the normalized run time of AFstack, Mups, D2, and MA28, for all 86 matrices. The normalized run time is the run time divided by the fastest run time found for that particular matrix. Thus the fastest method would have a normalized run time of 1.0.

The new method (AFstack) is faster than Mups, D2 and MA28 for only 13 out of 86 matrices. However, these matrices include nearly all of the large unsymmetric matrices which are not extremely sparse. The present release of the Harwell/Boeing collection is very weak in this class [12, 13]. Also, the method demonstrates a consistent performance for the entire range of matrices, as can be observed in Figures 7 through 9. It usually takes no more than twice the time as Mups for symmetric-patterned matrices, and is even occasionally faster (for bcsstk08 and the two Hm/add matrices). This is because it takes advantage of dense matrix kernels, as Mups does. It is faster than Mups for most matrices with asymmetry greater than 0.5 because it does not make the symmetric pattern assumption (the gemat matrices are notable exceptions; they become nearly symmetric-patterned when permuted to obtain a zero-free diagonal, as is done in Mups). The new method also avoids the worst-case behavior of D2 and MA28 for symmetric matrices and for very large matrices with substantial dense

substructure (such as the large chemical engineering problems).

Table 2 presents detailed results for each method on nine representative matrices taken from Table 1. The first section duplicates the matrix statistics from Table 1 (matrix names are abbreviated), and also states whether or not the matrix is symmetric positive-definite. The table then lists (1) the run time relative to the fastest method for each matrix, (2) the number of nonzeros in the LU factors, (3) the number of floating point operations performed during factorization, (4) the number of frontal matrices, (5) the number of pivots that were delayed due to an unacceptable numerical value (there are no delayed pivots in AFstack), (6) the number of edges in the assembly tree for Mups ($\mathcal{T}$) and in the assembly dag ($\mathcal{G}$) for AFdown, AFup, and AFstack, and (7) the number of times an active edge in $\mathcal{F}$ is scanned during the pivot search, assembly, and degree update phases in AFdown, AFup, and AFstack.

The sherman5 and mahindasb matrices contain 1674 and 669 singletons, respectively, which are 1-by-1 diagonal blocks arising from a permutation to upper-block-triangular form [10]. These singletons are included in the count of frontal matrices for the AF algorithms. The dags are unconnected for these matrices.

The number of delayed pivots in AFdown and AFup, although acceptable for many matrices, can be quite high. A delayed pivot is selected on symbolic grounds and then rejected on numerical grounds. The symbolic factorization then "retreats" by one step, possibly selecting another pivot that will later be rejected. Thus, the number of delayed pivots can actually exceed the size of the matrix, n, as is seen in the Z/m2 matrix for the AFdown algorithm. The problem of delayed pivoting seems to be more acute for the larger matrices in our test set.

The dag, $\mathcal{G}$, found by the unsymmetric-patterned multifrontal method does not have many more edges than the assembly tree for Mups (the number of edges in the tree is simply the number of frontal matrices minus one). The number of times an active edge in $\mathcal{F}$ is traversed is more critical to the performance of the new method than the number of edges in $\mathcal{G}$. Although this number is not guaranteed to be less than the number floating-point operations in the LU factorization, in practice it is much less than that (by three orders of magnitude for the larger matrices). It is usually less than the number of nonzeros in the LU factors (the exceptions are for very sparse matrices). This result demonstrates the effectiveness of our edge reductions.

The Z/rdist1 matrix is a distillation column with 19 components and 100 stages [30]. Reactions occur in stages 35 to 70. Both Mups and AFstack can take advantage of its unsymmetric block structure, which occurs within each stage. For example, of the 44.5 million floating-point operations in AFstack, only 80,255 are done in frontal matrices with 1 or 2 pivots (428 fronts). The remaining 44.4 million operations are performed in only 116 frontal matrices, (a typical one of which is 69-by-65 with 31 pivots). The largest frontal matrix constructed by AFstack is 216-by-281, with 171 pivots. For this matrix, Mups is directed to first find a maximum transversal [8], otherwise excessive fill-in is obtained. MA28 and D2 both find a poor pivot ordering for rdist1, when compared with Mups and the AF

algorithms. MA28 and D2 use only a global pivot search, and do not consider the overlap of the fill-in of a previous pivot with the current pivot. Using only a global pivot search destroys the block structure of the matrix and leads to excessive fill-in.

Table 3 compares the performance of a true degree update and an approximate degree update (described in Section 7.2) in the AFdown algorithm, for five matrices selected from Table 2. These results are on an Alliant FX/80, and are based on more appropriate values of user-settable parameters (for controlling amalgamation, pivoting, etc.) than those used on the Cray YMP. Thus, the amount of fill-in for AFdown reported in Table 2 differs from that reported in Table 3. Both versions allow some controlled fill-in due to amalgamation. The approximate degree update typically results in slightly higher fill-in, in exchange for a drastic reduction in run time, when compared with a true degree update. In two of these matrices (sherman5 and gre_1107), the use of approximate degrees instead of true degrees actually results in *less* fill-in and floating-point work.

Overall, these results show that the sequential prototypes of the unsymmetric-pattern multifrontal method are competitive algorithms when compared with the both classical multifrontal approach (Mups) and algorithms based on more conventional sparse matrix data structures (D2 and MA28).

9 Final remarks

The factor-only algorithm is "fragile" with respect to disruptions in the graphs and patterns of L and U caused by numerical pivoting. This problem is addressed by the analysis-factor algorithm. The disruptions are avoided by combining the numerical and symbolic phases so that pivots can be selected on both sparsity preserving and numerical criteria. However, the factor-only algorithm still forms an important part of a complete unsymmetric-pattern multifrontal method. If multiple problems are to be solved that have similar pattern and numerical characteristics (in solving nonlinear systems, for example), the pivot ordering of the first matrix is often suitable for successive matrices. The analysis-factor algorithm would factorize the first matrix and provide the pivot ordering to the factor-only algorithm, which factorizes subsequent matrices. Few numerical problems would be expected in the factor-only algorithm. However, a better handling of the disruptions caused by numerical pivoting is the most important open problem facing the development of a practical factor-only algorithm.

Parallelism is not yet addressed in the sequential versions of the analysis-factor algorithm. Some parallel work can take place within the dense matrix kernels, and while this is important, it will not provide enough parallelism in general. A truly parallel version must take advantage of parallelism across multiple frontal matrices. Parallelism can be incorporated in one of several ways. The parallel pivot search of the D2 algorithm can be adapted to this algorithm. The pivot search first creates an independent set of pivots (m, say). Each factorization task takes a single pivot and extends it into a block of pivots via

Table 1: Matrix statistics and run time

	Matrix Statistics				Run Time (in seconds)					
	name	n	nz	s	AFstack	AFdown	AFup	Mups	D2	MA28
1	662_bus	662	2474	0	0.087	0.139	0.103	0.072	**0.044**	0.066
2	nos6	675	3255	0	0.114	0.172	0.153	**0.102**	0.145	0.171
3	685_bus	685	3249	0	0.100	0.158	0.111	0.084	**0.082**	0.094
4	nos7	729	4617	0	0.218	0.281	0.386	**0.186**	0.341	0.997
5	bcsstk19	817	6853	0	0.155	0.216	0.175	**0.113**	0.364	0.210
6	orsirr_2	886	5970	0	0.244	0.327	0.424	**0.224**	0.379	0.741
7	gr_30_30	900	7744	0	0.231	0.287	0.275	**0.154**	0.502	0.552
8	nos2	957	4137	0	0.078	0.140	0.086	**0.071**	0.134	0.080
9	nos3	960	15844	0	0.311	0.411	0.397	**0.194**	1.201	1.187
10	pde_9511	961	4681	0	0.183	0.361	0.246	**0.152**	0.233	0.319
11	sherman1	1000	3750	0	0.162	0.270	0.204	**0.148**	0.174	0.303
12	orsirr_1	1030	6858	0	0.285	0.403	0.537	**0.266**	0.492	1.019
13	bcsstk08	1074	12960	0	**0.474**	0.845	0.636	0.894	0.775	1.604
14	bcsstk09	1083	18437	0	0.501	0.542	0.736	**0.417**	1.826	3.335
15	bcsstk10	1086	22070	0	0.268	0.374	0.329	**0.207**	1.254	0.833
16	bcsstm10	1086	22092	0	0.259	0.349	0.302	**0.208**	1.363	0.956
17	sherman4	1104	3786	0	**0.143**	0.244	0.187	0.172	0.165	0.214
18	1138_bus	1138	4054	0	0.129	0.208	0.140	0.113	**0.062**	0.087
19	bcsstk27	1224	56126	0	0.518	0.514	0.555	**0.422**	4.010	3.135
20	bcsstm27	1224	56126	0	0.537	0.528	0.572	**0.422**	5.793	3.136
21	bcsstk11	1473	34241	0	0.563	0.655	0.659	**0.328**	3.467	2.039
22	bcsstk12	1473	34241	0	0.563	0.657	0.659	**0.328**	3.459	2.039
23	bcsstm12	1473	19659	0	0.330	0.489	0.470	**0.251**	1.541	0.961
24	bcsstk14	1806	63454	0	1.441	1.944	1.114	**0.637**	9.968	6.516
25	bcsstk26	1922	30336	0	0.597	0.696	0.736	**0.357**	1.934	2.291
26	orsreg_1	2205	14133	0	0.754	0.911	2.099	**0.705**	1.889	4.041
27	Hm/add20	2395	13151	0	0.293	0.602	0.339	0.390	**0.179**	0.264
28	bcsstk23	3134	45178	0	5.319	8.195	7.562	**2.279**	32.519	205.430
29	bcsstk24	3562	159910	0	2.510	2.208	2.955	**1.420**	36.899	66.472
30	saylr4	3564	22316	0	1.322	1.618	3.270	**1.085**	13.138	9.835
31	bcsstk21	3600	26600	0	1.324	1.740	2.114	**1.131**	13.682	5.375
32	bcsstk15	3948	117816	0	12.114	13.907	10.874	**3.447**	96.093	108.980
33	bcsstk28	4410	219024	0	2.770	2.670	4.014	**1.829**	48.904	26.865
34	bcsstk16	4884	290378	0	7.298	6.886	13.163	**3.404**	153.726	76.674
35	Hm/add32	4960	19848	0	0.500	0.917	0.499	0.651	**0.266**	0.378
36	sherman3	5005	20033	0	**1.237**	2.782	2.361	1.372	5.508	6.157
37	bcsstk18	11948	149090	0	16.943	30.794	60.287	**4.870**	168.935	393.710

Table 1 continued. Matrix statistics and run time.

	Matrix Statistics				Run Time (in seconds)					
	name	n	nz	s	AFstack	AFdown	AFup	Mups	D2	MA28
38	hwatt_1	1856	11360	0.013	0.595	1.578	1.308	**0.487**	2.579	3.210
39	hwatt_2	1856	11550	0.020	0.581	1.370	1.302	**0.534**	2.623	3.306
40	Hm/mem+	17758	99147	0.021	2.898	6.787	3.510	5.612	**1.305**	2.563
41	jpwh_991	991	6027	0.064	0.294	0.500	0.438	**0.242**	0.425	1.372
42	lns_3937	3937	25407	0.150	3.073	4.788	3.358	**1.062**	6.075	19.277
43	lns_3937d	3937	25407	0.150	3.480	5.224	3.311	**1.053**	7.527	17.483
44	nnc666	666	4032	0.179	0.170	0.256	0.182	**0.103**	0.152	0.377
45	lns_511	511	2796	0.201	0.124	0.173	0.156	0.091	**0.081**	0.189
46	lns_511c	511	2796	0.201	0.132	0.177	0.163	0.090	**0.072**	0.206
47	pores_3	532	3474	0.258	0.106	0.163	0.117	**0.067**	0.124	0.116
48	sherman5	3312	20793	0.261	0.957	1.314	1.085	**0.947**	2.199	5.434
49	mc_fe	765	24382	0.301	**0.338**	0.560	0.387	0.440	0.723	1.685
50	fs_541_4	541	4273	0.317	0.134	0.180	0.170	0.220	**0.123**	0.228
51	fs_541_1	541	4282	0.317	0.132	0.188	0.158	0.220	**0.130**	0.196
52	fs_541_2	541	4282	0.317	**0.128**	0.181	0.172	0.220	0.148	0.228
53	fs_541_3	541	4282	0.317	**0.134**	0.188	0.173	0.220	0.158	0.227
54	sherman2	1080	23094	0.329	0.805	0.918	0.697	**0.449**	1.397	8.262
55	fs_760_1	760	5739	0.354	0.329	0.336	0.370	**0.176**	0.218	1.036
56	fs_760_3	760	5739	0.354	0.327	0.336	0.370	**0.176**	0.218	1.036
57	pores_2	1224	9613	0.388	**0.354**	0.525	0.536	0.402	0.375	1.107
58	fs_680_3	680	2471	0.439	0.058	0.081	0.065	0.085	**0.049**	0.059
59	fs_680_2	680	2424	0.448	0.058	0.079	0.065	0.085	**0.053**	0.063
60	steam2	600	5660	0.451	0.175	0.279	0.266	**0.139**	0.243	0.565

Table 1 continued. Matrix statistics and run time.

	Matrix Statistics				Run Time (in seconds)					
	name	n	nz	s	AFstack	AFdown	AFup	Mups	D2	MA28
61	Z/m2	8641	102449	0.508	**13.596**	31.595	14.975	33.691	93.621	362.766
62	Z/rdist3a	2398	61896	0.860	**0.648**	1.970	1.404	1.606	9.864	66.584
63	Z/rdist1	4134	94408	0.941	**1.625**	3.089	1.811	2.506	33.013	267.308
64	Z/radfr1	1048	13299	0.946	**0.221**	0.373	0.240	0.252	0.787	2.668
65	Z/rdist2	3198	56834	0.954	0.886	1.388	**0.861**	1.252	7.504	117.395
66	west0989	989	3518	0.982	0.152	0.174	0.130	0.142	**0.064**	0.075
67	mahindas	1258	7682	0.983	0.223	0.381	0.218	0.825	**0.096**	0.156
68	bp_1600	822	4841	0.989	0.105	0.123	0.104	0.269	0.088	**0.082**
69	bp_1400	822	4790	0.990	0.116	0.136	0.119	0.275	0.089	**0.087**
70	bp_1000	822	4661	0.991	0.097	0.117	0.095	0.283	0.079	**0.073**
71	bp_1200	822	4726	0.991	0.099	0.119	0.100	0.278	0.088	**0.080**
72	bp_0	822	3276	0.991	0.024	**0.020**	0.020	0.157	0.031	0.025
73	bp_200	822	3802	0.994	0.046	0.048	0.042	0.204	0.047	**0.039**
74	bp_600	822	4172	0.994	0.076	0.087	0.074	0.244	0.069	**0.059**
75	west0655	655	2808	0.994	0.108	0.151	0.114	0.131	**0.058**	0.082
76	bp_400	822	4028	0.995	0.069	0.074	0.063	0.222	0.058	**0.050**
77	bp_800	822	4534	0.995	0.091	0.106	0.091	0.261	0.074	**0.065**
78	west2021	2021	7310	0.997	0.324	0.379	0.275	0.320	0.181	**0.162**
79	gemat12	4929	33044	0.999	0.606	1.016	0.771	**0.549**	1.245	0.831
80	gemat11	4929	33108	0.999	0.597	0.919	0.716	**0.520**	0.961	0.756
81	west1505	1505	5414	0.999	0.234	0.277	0.204	0.220	**0.117**	0.120
82	gre_512	512	1976	1.000	0.140	0.209	0.182	0.128	**0.064**	0.263
83	shl_0	663	1687	1.000	0.016	**0.013**	0.013	0.118	0.019	0.016
84	shl_200	663	1726	1.000	0.017	**0.014**	0.014	0.126	0.019	0.016
85	shl_400	663	1712	1.000	0.017	**0.014**	0.014	0.128	0.020	0.016
86	gre_1107	1107	5664	1.000	0.417	0.628	0.573	0.518	**0.294**	1.146

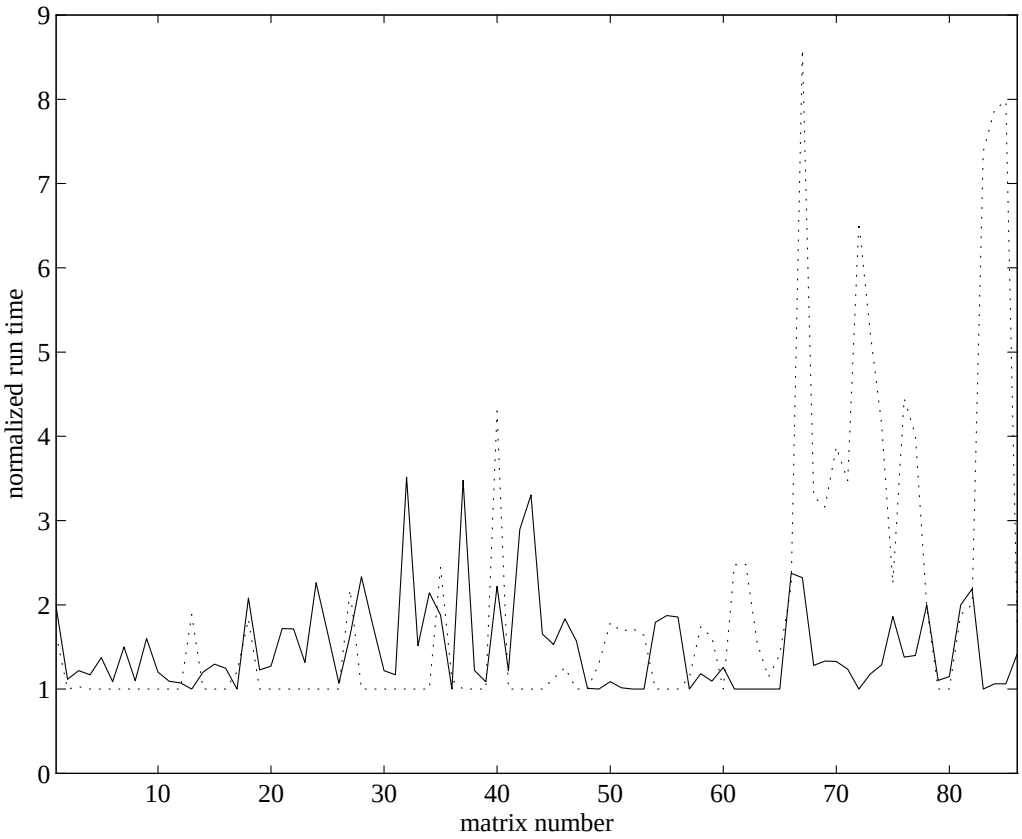

Figure 7: Relative performance: AFstack (solid) and Mups (dotted)

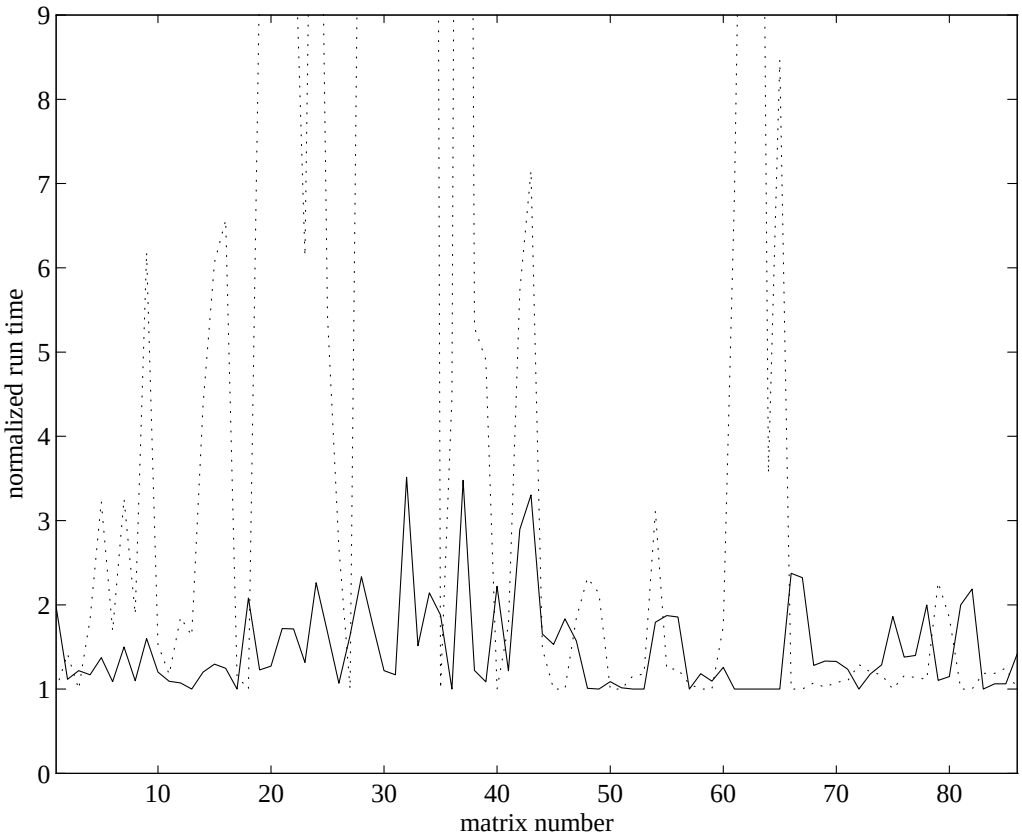

Figure 8: Relative performance: AFstack (solid) and D2 (dotted)

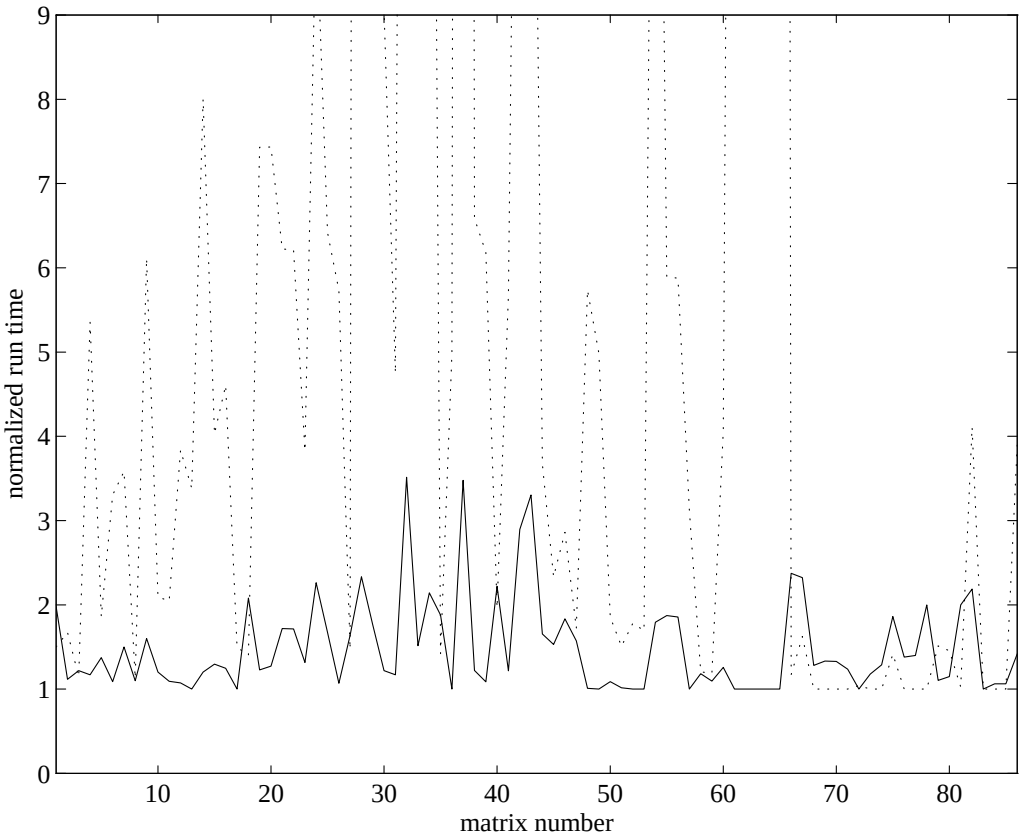

Figure 9: Relative performance: AFstack (solid) and MA28 (dotted)

Table 2: Detailed results for nine representative matrices

Matrix									
matrix number	29	37	42	48	61	63	67	80	86
name	b24	b18	lns	sher5	m2	rdist1	mah	ge11	gre
sym. pos. def.?	yes	yes	no	no	no	no	no	no	no
order	3562	11948	3937	3312	8641	4134	1258	4929	1107
nonzeros	159910	149090	25407	20793	102449	94408	7682	33108	5664
asymmetry	0	0	0.150	0.261	0.508	0.941	0.983	0.999	1.000
Rel. run time									
MA28	46.8	80.8	18.1	5.7	26.7	164.5	1.6	1.5	3.9
D2	26.0	34.7	5.7	2.3	6.9	20.3	1	1.8	1
Mups	1	1	1	1	2.5	1.5	8.6	1	1.8
AFdown	1.6	6.3	4.5	1.4	2.3	1.9	4.0	1.8	2.1
AFup	2.1	12.4	3.1	1.1	1.1	1.1	2.3	1.3	1.9
AFstack	1.8	3.5	2.9	1.0	1	1	2.3	1.1	1.4
Nz in LU (10^3)									
MA28	990.0	2910.7	427.1	158.0	3045.0	2280.2	10.1	50.1	47.0
D2	1699.4	7389.7	847.3	316.3	4966.7	2431.0	11.2	72.3	71.0
Mups	592.4	1540.3	332.7	187.0	2805.6	279.4	50.3	79.4	187.6
AFdown	805.1	5123.3	1060.8	269.1	5928.2	624.0	16.1	92.9	132.8
AFup	1253.8	9087.6	937.7	260.6	3864.9	564.5	19.3	103.6	133.0
AFstack	1053.0	4733.0	543.1	285.3	3994.6	532.5	13.2	99.4	94.4
Flops (10^6)									
Mups	72.2	347.5	27.2	16.8	938.7	10.2	2.80	0.83	25.0
AFdown	141.4	4725.4	362.9	37.5	5301.9	49.8	0.39	1.18	21.7
AFup	325.6	9412.1	248.3	34.4	2380.2	40.7	0.38	1.65	15.1
AFstack	181.5	2335.4	100.9	43.6	1636.1	24.5	0.16	1.08	5.7
Frontal mat.									
Mups	285	3209	1521	2226	1860	579	508	983	342
AFdown	219	2130	1207	2057	1782	302	1027	641	425
AFup	224	2684	1248	2243	2894	449	822	1041	243
AFstack	189	3897	2213	2151	2881	544	1016	1144	334
Delayed pivots									
Mups	0	0	1204	0	0	14	4	0	270
AFdown	51	3678	1339	177	11078	865	0	6	8
AFup	755	1225	453	190	1027	309	3	42	50
Edges in $\mathcal{T}$, $\mathcal{G}$									
Mups	284	3208	1520	2225	1859	578	507	982	341
AFdown	244	3398	3341	841	4284	541	616	692	1093
AFup	356	4479	3120	1286	5687	498	311	799	760
AFstack	254	4965	6300	886	5968	705	818	892	929
$\mathcal{F}$ **scans** (10^3)									
AFdown	168.2	2686.8	965.4	232.5	2146.8	280.7	186.5	86.5	159.7
AFup	183.6	1681.9	559.2	194.3	1440.8	132.2	39.7	50.8	111.0
AFstack	131.7	2219.4	865.4	140.8	2364.0	212.1	78.1	39.1	102.2

Table 3: True versus approximate degree update (AFdown on Alliant FX/80)

Matrix					
matrix number	42	48	67	80	86
name	lns_3937	sherman5	mahindas	gemat11	gre_1107
order	3937	3312	1258	4929	1107
nonzeros	25407	20793	7682	33108	5664
asymmetry	0.150	0.261	0.983	0.999	1.000
Run time (seconds)					
True	420.5	66.9	8.5	15.5	29.9
Approximate	172.5	30.3	5.7	14.6	12.1
Nz in LU (10^3)					
True	597.6	210.3	22.6	72.9	78.9
Approximate	611.7	201.8	23.9	76.6	72.4
Flops (10^6)					
True	102.1	25.8	0.31	0.71	7.4
Approximate	117.9	22.9	0.37	0.85	4.9

dynamic amalgamation. Since multiple tasks can affect a single row or column in the active submatrix, these tasks either cooperate to update the degrees, or a separate parallel degree update phase is employed. When all factorization tasks finish, a new set of independent pivots is found. A second approach would pipeline the pivot search with the numerical factorization. The pivot search task (with one processor or a set of processors) searches for pivots that are independent with the pivots of currently-executing factorization tasks. A task k is created for each pivot (multiple processors can also be used within each task). Task k creates the frontal matrix E_k, performs the assembly, factorizes it, and either cooperates with other tasks to perform the degree update or requests the pivot search task to perform the degree update. When it completes, it signals the pivot search task that the rows and columns it affected ($\mathcal{L}_k''$ and $\mathcal{U}_k''$, respectively) are now candidates for the pivot search. In both approaches, multiple factorizations of the associated frontal matrices are done in parallel.

10 Acknowledgments

We would like to thank Patrick Amestoy, Michel Daydé, and Mario Arioli at CERFACS, and Steve Zitney at Cray Research, Inc. for many helpful discussions. Steve Zitney at Cray and Steve Hamm at Motorola provided us with several large unsymmetric matrices, a class of matrices that is weak in the present version of the Harwell/Boeing collection [12, 13].

Portions of this work were supported by a post-doctoral grant from CERFACS, September 1989 to December 1990. Support for this project also provided by the National Science Foundation (ASC-9111263), and by Cray Research, Inc. and Florida State University through the allocation of supercomputer resources.

References

[1] A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1:131–137, 1972.

[2] P. R. Amestoy and I. S. Duff. Vectorization of a multiprocessor multifrontal code. *Int. J. Supercomputer Appl.*, 3(3):41–59, 1989.

[3] T. A. Davis. Performance of an unsymmetric-pattern multifrontal method for sparse LU factorization. Technical Report TR-92-014, Comp. and Info. Sci. Dept., Univ. of Florida (anonymous ftp to cis.ufl.edu:cis/tech-reports/tr92/tr92-014.ps.Z), Gainesville, FL, May 1992.

[4] T. A. Davis and I. S. Duff. Unsymmetric-pattern multifrontal methods for parallel sparse LU factorization. Technical Report TR-91-023, CIS Dept., Univ. of Florida (anonymous ftp to cis.ufl.edu:cis/tech-reports/tr91/tr91-023.ps.Z), Gainesville, FL, 1991.

[5] T. A. Davis and P. C. Yew. A nondeterministic parallel algorithm for general unsymmetric sparse LU factorization. *SIAM J. Matrix Anal. Appl.*, 11(3):383–402, 1990.

[6] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A set of level-3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, 1990.

[7] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of Fortran basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 14:1–32, 1988.

[8] I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Transactions on Mathematical Software*, 7:315–330, 1981.

[9] I. S. Duff. Parallel implementation of multifrontal schemes. *Parallel Computing*, 3:193–204, 1986.

[10] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* London: Oxford Univ. Press, 1986.

[11] I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner. The factorization of sparse symmetric indefinite matrices. *IMA Journal of Numerical Analysis*, 11:181–204, 1991.

[12] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Softw.*, 15:1–14, 1989.

[13] I. S. Duff, R. G. Grimes, and J. G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection. Technical Report TR/PA/92/86, CERFACS, Toulouse, France, Oct. 1992.

[14] I. S. Duff and J. K. Reid. Some design features of a sparse matrix code. *ACM Trans. Math. Softw.*, 5(1):18–35, 1979.

[15] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Softw.*, 9(3):302–325, 1983.

[16] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear equations. *SIAM J. Sci. Statist. Comput.*, 5(3):633–641, 1984.

[17] I. S. Duff and J. K. Reid. MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. *to appear*, 1993.

[18] S. C. Eisenstat and J. W. H. Liu. Exploiting structural symmetry in unsymmetric sparse symbolic factorization. *SIAM J. Matrix Anal. Appl.*, 13(1):202–211, 1992.

[19] A. George, M. T. Heath, J. W. H. Liu, and E. Ng. Solution of sparse positive definite systems on a shared-memory multiprocessor. *International Journal of Parallel Programming*, 15(4):309–325, 1986.

[20] A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[21] A. George and E. Ng. Parallel sparse gaussian elimination with partial pivoting. *Annals of Operation Research*, 22:219–240, 1990.

[22] J. R. Gilbert. An efficient parallel sparse partial pivoting algorithm. Technical Report 88/45052-1, Center for Computer Science, Chr. Michelsen Institute, Bergen, Norway, 1988.

[23] J. R. Gilbert and J. W. H. Liu. Elimination structures for unsymmetric sparse LU factors. Technical Report CS-90-11, Dept. of Computer Sci., York Univ., North York, Ontario, Feb. 1990.

[24] S. Hadfield and T. A. Davis. Analysis of potential parallel implementations of the unsymmetric-pattern multifrontal method for sparse LU factorization. Technical Report TR-92-017, CIS Dept., Univ. of Florida (anonymous ftp to cis.ufl.edu:cis/tech-reports/tr92/tr92-017.ps.Z), Gainesville, FL, June 1992.

[25] M. T. Heath, E. Ng, and B. W. Peyton. Parallel algorithms for sparse linear systems. *SIAM Review*, 33(3):420–460, 1991.

[26] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, 1990.

[27] H. M. Markowitz. The elimination form on the inverse and its application to linear programming. *Management Science*, 3:255–269, Apr 1957.

[28] P. Matstoms. Sparse QR factorization in MATLAB. Technical Report LiTH-MAT-R-1992-05, Dept. of Mathematics, Linkoping Univ., Linkoping, Sweden, March 1992.

[29] S. E. Zitney and M. A. Stadtherr. A frontal algorithm for equation-based chemical process flowsheeting on vector and parallel computers. In *Proc. AIChE Annual Meeting*, Washington, DC, 1988.

[30] S.E. Zitney. A frontal code for aspen plus on advanced architecture computers. In *American Inst. of Chemical Eng. Annual Meeting, Symposium on Parallel Computing*, Nov. 1990.

[31] Z. Zlatev, J. Wasniewski, and K. Schaumburg. *Y12M: Solution of Large and Sparse Systems of Linear Algebraic Equations, Lecture Notes in Computer Science 121.* Berlin: Springer-Verlag, 1981.