Unsymmetric-pattern multifrontal methods for parallel sparse LU factorization

T. A. Davis*
Computer and Information Sciences Department
University of Florida,
Gainesville, Florida, USA

I. S. Duff†
European Center for Research and Advanced Training
in Scientific Computation (CERFACS)
Toulouse, France

September 1991

Technical Report TR-91-23
Computer and Information Sciences Department
University of Florida
Gainesville, FL, 32611 USA

*phone: (904) 392-1481, email: davis@cis.ufl.edu. Supported by a post-doctoral grant from CERFACS, September 1989 to December 1990. Continuing research supported by the National Science Foundation (ASC-9111263, 8/91). This report is available in postscript form via anonymous ftp (in binary mode) to cis.ufl.edu as the compressed file /cis/tech-reports/tr91/tr91-023.ps.Z.

†Also Rutherford Appleton Laboratory, Chilton, Didcot, Oxon. 0X11 0QX England.

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 method 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 elimination 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, potentially giving it high performance on parallel-vector supercomputers. Performance of a sequential version is compared with the classical multifrontal method and a standard unsymmetric solver on an Alliant FX/80 computer.

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 is designed with regular memory access behavior in the innermost loops [14]. 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 father in the elimination tree. The elimination tree controls parallelism across multiple frontal matrices, while dense matrix operations [6] provide parallelism and vectorization within each frontal matrix. These concepts are discussed by Duff and Reid in [14].

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 task graph replaces the elimination tree, and frontal matrices are no longer assembled by a single father.

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

1.1 Previous work

The multifrontal method of Duff and Reid (MA37) [3, 9, 14] will be referred to as the *classical* multifrontal method. It will be described in Section 2 to contrast it with the unsymmetric-pattern multifrontal method proposed here. Other parallel algorithms for unsymmetric sparse matrices include the D2 algorithm [5], partial pivoting methods [20, 21, 22, 24], the PSolve algorithm [4], and others [2, 27]. The D2 algorithm is based on a nondeterministic 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). Of all these algorithms for unsymmetric sparse matrices, the classical multifrontal method takes most advantage of dense matrix kernels, but is unsuitable when the pattern of the matrix is very unsymmetric.

Most recently, Gilbert and Liu [23] and Eisenstat and Liu [16] 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 reduced data flow and control flow graphs presented in this report. Moreover, we also indicate how our graphs can be applied to the case where the pivot ordering is not known a priori.

1.2 Outline of report

The following sections discuss issues in developing an unsymmetric-pattern multifrontal method and in designing algorithms and software to implement it. Section 2 describes LU factorization in terms of general frontal matrices. Section 3 shows that the elimination tree is insufficient for the unsymmetric-pattern multifrontal method. Section 4 presents a reduced data flow graph and a reduced control flow graph, and discusses their amalgamation and representation. These two directed acyclic graphs are to the unsymmetric-pattern multifrontal method as the elimination 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 the reduced data flow and control flow graphs, 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 the analysis-factor algorithm 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 ($1 \leq k \leq n, A^{[0]} = A, L^{[n]} = L, U^{[n]} = U$). The *active submatrix*, A_k, is the lower-right $(n-k)$-by-$(n-k)$ submatrix of $A^{[k]}$, and is the portion of the matrix $A^{[k]}$ that has still to be reduced after step k has finished. We can write

$$L^{[k]}A^{[k]}U^{[k]} = \begin{bmatrix} L_1 & 0 \\ L_2 & I_{n-k} \end{bmatrix} \cdot \begin{bmatrix} I_k & 0 \\ 0 & A_k \end{bmatrix} \cdot \begin{bmatrix} U_1 & U_2 \\ 0 & I_{n-k} \end{bmatrix},$$

where matrices L_1 and U_1 are k-by-k lower and upper triangular matrices, respectively, I_k is the k-by-k identity matrix, and I_{n-k} is the $(n-k)$-by-$(n-k)$ 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, permutations will be ignored without loss of generality.

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.

Step k selects a single pivot entry $a_{ij}^{[k-1]}$ from the submatrix A_{k-1}, and interchanges row i and column j with the leading row and column of A_{k-1}, respectively (equivalently, the k-th row and column of $A^{[k-1]}$). Row i and column j are the k-th pivot row and column, respectively. Step k then computes $L^{[k]}$, $A^{[k]}$, and $U^{[k]}$ from $L^{[k-1]}$, $A^{[k-1]}$, and $U^{[k-1]}$.

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

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

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

The *row degree* $r_i^{[k]}$ $(i > k)$ is the number of entries in row i of $A^{[k]}$, and the *column degree* $c_j^{[k]}$ $(j > k)$ 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 ($c_j^{[k-1]}$-by-$r_i^{[k-1]}$) submatrix that corresponds to the pivot $(a_{ij}^{[k-1]})$ 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,

$$\mathcal{U}_k = Struct(A_{i*}^{[k-1]})$$
$$\mathcal{L}_k = Struct(A_{*j}^{[k-1]}).$$

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.

One or more consecutive pivots $a^{[k]}$, ..., $a^{[k+g_k-2]}$ (for some $g_k > 1$) may have the same pivot row and column structure (excluding earlier pivots) as the first pivot $a^{[k-1]}$ in the frontal matrix. In this case, their pivot rows and columns are also in the element E_k. The q-th row and column in E_k is the $(k+q-1)$-th pivot row and column $(1 \leq q \leq g_k)$. 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 $A^{[k-1]}$ to obtain the reduced matrix $A^{[k+g_k-1]}$. 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 structure are referred to as a *super-row*, and columns with identical column structure 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 subsets:

$$\mathcal{L}_k = \mathcal{L}_{1k} \cup \mathcal{L}_{2k}$$

(where $\mathcal{L}_{1k}$ is the set of g_k pivot rows in E_k) and

$$\mathcal{U}_k = \mathcal{U}_{1k} \cup \mathcal{U}_{2k},$$

(where $\mathcal{U}_{1k}$ is the set of g_k pivot columns in E_k). The sets $\mathcal{L}_{1k}$, $\mathcal{L}_{2k}$, $\mathcal{U}_{1k}$, and $\mathcal{U}_{2k}$ divide the element E_k into four submatrices F_k, B_k, C_k and D_k,

$$E_k = \begin{array}{c c} & \begin{array}{cc} \mathcal{U}_{1k} & \mathcal{U}_{2k} \end{array} \\ \begin{array}{c} \mathcal{L}_{1k} \\ \mathcal{L}_{2k} \end{array} & \left[\begin{array}{cc} F_k & B_k \\ C_k & D_k \end{array} \right] \end{array},$$

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, all contributions from previous elements are added into F_k, B_k and C_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 ($F_k = L_{1k}U_{1k}$), computes the block column L_{2k} of L and the block row U_{2k} of U, and updates the contribution block with the Schur complement

$$D_k \leftarrow D'_k = D_k - L_{2k}U_{2k},$$

overwriting D_k with D'_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_{1k} and L_{2k} define columns k through $k + g_k - 1$ of L, and U_{1k} and U_{2k} define rows k through $k + g_k - 1$ of U.

The lower-right $(n-k)$-by-$(n-k)$ submatrix of A is referred to as the *active part* of A, and is denoted by $A_{k+1..n,k+1..n}$. The active submatrix A_k is represented as a sum of elements created during factorization, plus the active part of A,

$$A_k = A_{k+1..n,k+1..n} + \sum_{t=1}^{k} E_t^{[k]}$$

(where elements amalgamated into other elements are not included in the summation). The active part of an element E_t just after step k (denoted as $E_t^{[k]}$) is a submatrix of E_t formed by rows and columns that are non-pivotal after step k (where $1 \leq t \leq k$). The row and column pattern of $E_t^{[k]}$ are denoted as $\mathcal{L}_t^{[k]}$ and $\mathcal{U}_t^{[k]}$, respectively.

2.1 Example matrix

Consider the matrix A

$$A = \left[\begin{array}{ccccccc}
p_1 & \cdot & \cdot & \times & \times & \cdot & \cdot \\
\times & p_2 & \times & \cdot & \times & \cdot & \times \\
\times & \times & p_3 & \cdot & \cdot & \cdot & \times \\
\times & \cdot & \cdot & p_4 & \times & \cdot & \cdot \\
\cdot & \times & \times & \cdot & \times & \times & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \times & \times \\
\times & \times & \cdot & \cdot & \times & \cdot & \times
\end{array}\right]$$

with a partially factorized matrix $L^{[3]}A^{[3]}U^{[3]}$

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

after three steps 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 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 after step 3 is

$$A_3 = A_{4..7,4..7} + E_1^{[3]} + E_2^{[3]},$$

or,

$$A_3 = \begin{array}{c|cccc} & 4 & 5 & 6 & 7 \\ \hline 4 & \times & \times & \cdot & \cdot \\ 5 & \cdot & \times & \times & \cdot \\ 6 & \cdot & \cdot & \times & \times \\ 7 & \cdot & \times & \cdot & \times \end{array} + \begin{array}{c|cc} & 4 & 5 \\ \hline 4 & \times & \times \\ 7 & \times & \times \end{array} + \begin{array}{c|ccc} & 4 & 5 & 7 \\ \hline 5 & \times & \times & \times \\ 7 & \times & \times & \times \end{array}.$$

Note that E_3 is not generated, since it lies entirely within E_2. If the entry $a_{4,4}^{[3]}$ (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.

3 Elimination tree and elimination directed acyclic graphs

The *elimination tree* forms the basis of many sparse matrix algorithms, describing either the data flow graph or the control flow graph, or both [25]. A more general pair of graphs is needed for the unsymmetric-pattern multifrontal method.

A *data flow graph* is a directed acyclic graph $G_{data} = (\mathcal{V}, \mathcal{E}_{data})$ that consists of a set of nodes $\mathcal{V} \subseteq \{1, \cdots, n\}$, and a set of directed edges $\mathcal{E}_{data}$. Without amalgamation, $\mathcal{V} = \{1, \cdots, n\}$. With amalgamation, $\mathcal{V}$ is the set of all explicitly represented frontal matrices ($\mathcal{V} = (1, 2, 4, \cdots)$ for the example in Section 2). The corresponding *control flow graph* is $G_{control} = (\mathcal{V}, \mathcal{E}_{control})$.

For multifrontal methods, a node $s \in \mathcal{V}$ represents the assembly and factorization of element E_s. The directed edge $(s,t) \in \mathcal{E}_{data}$, $s < t$, if node t receives data directly from node s. Similarly, $(s,t) \in \mathcal{E}_{control}$ if the execution of node t must pause at some point and wait for node s to reach a certain point in its execution (for multifrontal methods, t usually waits until s completes). The data flow graph is sufficient to describe the control flow dependences, but some of the edges might be unnecessary. Thus, $\mathcal{E}_{control} \subseteq \mathcal{E}_{data}$.

A *complete* directed acyclic graph $G_{complete} = (\mathcal{V}, \mathcal{E}_{complete})$ is sufficient for describing both the data flow and control flow for the unsymmetric-pattern multifrontal method. The complete data flow graph, $G_{complete\_data}$, and the complete control flow graph, $G_{complete\_control}$ are simply equal to $G_{complete}$. $\mathcal{E}_{complete}$ has directed edges (s,t) of two types: $\mathcal{E}^L_{complete}$ or $\mathcal{E}^U_{complete}$, where

$$\mathcal{E}^L_{complete} = \{(s,t) \mid s < t, l_{ts} \neq 0\} = \{(s,t) \mid t \in \mathcal{L}_{2s}\}$$

$$\mathcal{E}^U_{complete} = \{(s,t) \mid s < t, u_{st} \neq 0\} = \{(s,t) \mid t \in \mathcal{U}_{2s}\}$$

$$\mathcal{E}_{complete} = \mathcal{E}^L_{complete} \cup \mathcal{E}^U_{complete}$$

$$\mathcal{E}^L_{complete\_control} \equiv \mathcal{E}^L_{complete\_data} \equiv \mathcal{E}^L_{complete}$$

$$\mathcal{E}^U_{complete\_control} \equiv \mathcal{E}^U_{complete\_data} \equiv \mathcal{E}^U_{complete}.$$

Similar directed acyclic graphs have been used for symbolic LU factorization of unsymmetric sparse matrices [16, 23].

Most parallel sparse matrix factorization algorithms are based on the elimination tree, $\mathcal{T}$,

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

$$\mathcal{E}_{tree} = \{(s,t) \mid t = \mathit{father}\,(s)\}$$

$$\mathit{father}\,(s) = \min\{t \mid s < t, l_{ts} \neq 0\}.$$

If a node a appears in the unique path from node d to the root, then node a is an *ancestor* of its *descendant* node d. The elimination tree is typically defined only for symmetric-patterned LU factors, with the exception of partial-pivoting methods [21, 22].

The classical multifrontal method [13] is one method based on the elimination 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 [19], determines the elimination tree, and finds the patterns of L and U. Each node s in the elimination tree represents the work associated with element E_s. The elimination tree describes the large-grain parallelism between the nodes. For both the classical and unsymmetric-pattern multifrontal methods, the work at a node s will be referred to as task s, although additional medium-grain parallelism can occur within each task, using the Level-3 BLAS [6] for supernodes or Level-2 BLAS [7] for simple nodes.

The column pattern $\mathcal{U}_t$ of the frontal matrix E_t is given by the union of the column pattern $\mathcal{U}_s^{[t-1]}$ of each son s of node t and the structure of row t of the upper triangular part of A. A node can be amalgamated with one of its sons if the row pattern of the father is identical to the son (excluding the index of the son's pivot column). That is, if $\mathcal{U}_t = \mathcal{U}_{2s}$ for a single son s of node t. 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 elimination tree to compute the LU factorization. The tree acts as both a data flow graph by guiding the assembly process and the construction of new elements, and as a control flow graph by describing the precedence between nodes. Task t assembles the frontal matrices $E_s^{[t-1]}$ of each son node s into E_t. Because of the symmetric-pattern assumption the pattern $\mathcal{U}_t$ is a superset of the pattern $\mathcal{U}_s^{[t-1]}$ of the contribution block of each son s. The entire contribution block D_s of a son can always be assembled into its father, since all the rows and columns that are affected by D_s are present in E_t. The method takes advantage of the dense matrix kernels [6, 7] to factorize E_t: the Level-2 BLAS (outer-product) if $g_t = 1$ or the Level-3 BLAS if $g_t > 1$.

The classical multifrontal method is not the only method based on the elimination tree. In the sparse column-Cholesky factorization of George et al. [17], 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. The data flow graph is not a tree, but it is at least spanned by the control flow graph, which is simply the elimination tree. This is in contrast to the classical multifrontal method, in which data is assembled only from the sons of a node.

However, the elimination 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_r, E_s, and E_t $(r < s < t)$ described the following 3-by-3 submatrix of $L \backslash U$ formed from the pivot rows and columns r, s, and t:

$$\begin{bmatrix} p_r & u_{rs} & u_{rt} \\ l_{sr} & p_s & u_{st} \\ l_{tr} & l_{ts} & p_t \end{bmatrix}.$$

Assume that there are other nonzeros in these row and columns of U and L, respectively. If, for example, $u_{rs} = l_{ts} = 0$, the inter-relationships become

$$\begin{bmatrix} p_r & \cdot & u_{rt} \\ l_{sr} & p_s & u_{st} \\ l_{tr} & \cdot & p_t \end{bmatrix}. \tag{1}$$

The example given earlier in Section 2.1 fits this case, where $r = 1$, $s = 2$, and $t = 4$ (except that element E_2 is a supernode with $g_2 = 2$).

Task r computes update terms for both rows s and t. The row pattern $\mathcal{L}_s$ of E_s, however, does not contain the row index t, so the update terms in row t of E_r cannot be assembled into E_s. Figure 1 shows the corresponding fragment of the data flow graph (only nodes r, s, and t). The data flow edges are labeled with rows and columns of the updates computed by the source and sent to the sink. The data flow edge (r, t) is present because row t of E_r must be assembled directly into E_t. It cannot be assembled into E_s for later assembly into E_t. The data flow graph shown in Figure 1 is not a tree, although it would be possible to use an elimination tree as a control flow graph in this case. The edges of a sufficient control flow graph fragment would be (r, s) and (s, t). This 3-node elimination tree spans the data flow graph in Figure 1, as is the case in sparse column-Cholesky.

In general, however, it is not possible to use an elimination tree to describe the control flow graph. Consider the case in Equation 1 if $u_{rt} = u_{st} = 0$. The 3-by-3 submatrix of $L \backslash U$ becomes

$$\begin{bmatrix} p_r & \cdot & \cdot \\ l_{sr} & p_s & \cdot \\ l_{tr} & \cdot & p_t \end{bmatrix}. \tag{2}$$

In this case, tasks s and t both assemble a contribution from E_r (assuming that there are other nonzeros in row r), but they may proceed in parallel

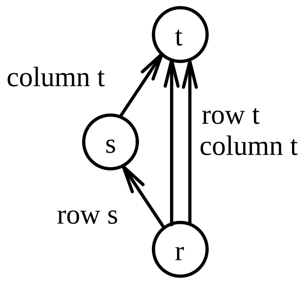

Figure 1: Data flow graph fragment for Equation 1

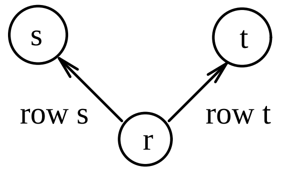

Figure 2: Data flow graph fragment for Equation 2

after task r completes. The corresponding fragment of the data flow graph is shown in Figure 2. The control flow graph is not a tree since node r has more than one father; the parallelism between s and t cannot be described by a tree rooted at node n.

Thus the elimination tree can be used neither as a data flow graph nor as a control flow graph for a general frontal matrix method. This point is further illustrated by the following example. The matrix (3) shows the leading 5-by-5 submatrix of the $L\backslash U$ factors of a larger matrix,

$$\begin{bmatrix} p_1 & \cdot & \cdot & \times & \cdot \\ \times & p_2 & \cdot & \times & \cdot \\ \times & \cdot & p_3 & \times & \cdot \\ \cdot & \cdot & \cdot & p_4 & \cdot \\ \times & \cdot & \cdot & \times & p_5 \end{bmatrix}. \tag{3}$$

Figure 3 shows the corresponding complete graph. The data dependences in the figure are labeled with the entries in L and U that cause them,

$$\mathcal{E}^L_{complete} = \{(1,2),(1,3),(1,5),(4,5)\}$$

$$\mathcal{E}^U_{complete} = \{(1,4),(2,4),(3,4)\}.$$

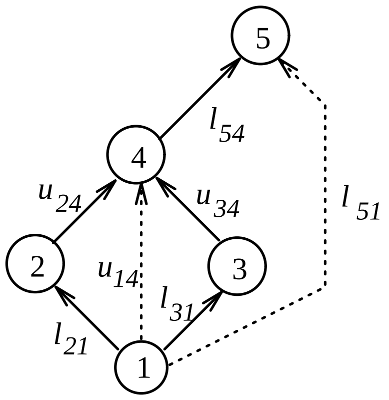

Figure 3: Data flow and control flow graph for Equation 3

With the seven data dependences shown, the control flow graph must include at least the edges drawn with solid lines (edges (1,2), (1,3), (2,4), (3,4), and (4,5)). It is not a tree but a directed acyclic graph. If a control flow graph was chosen that satisfied the data dependences (and yet was still a tree), it would limit the parallelism. The parallelism between nodes 2 and 3 cannot be described by a tree. If matrix (3) and the given pivot ordering were presented to the classical multifrontal method, the entire 5-by-5 submatrix would be dense and no parallelism would exist between nodes 2 and 3.

4 Reduced data flow and control flow graphs

A frontal matrix will persist until its contribution block is completely assembled into other frontal matrices. Using the complete graph as the data flow graph, the contribution blocks are held for the longest time possible before being completely assembled and discarded, because each edge represents the assembly of a small amount of data (a single row or column). A simpler data flow graph will improve the memory requirements of the method. Similarly, although the complete graph can be used as the control flow graph, removing redundant control flow edges will simplify amalgamation. Common simplifications to both the data flow graph and the control flow graph are discussed first, followed by simplifications applicable to the control flow graph only.

Relationships between adjacent nodes in the control flow graph and data flow graph are defined as follows. Node s is an *Lson* of its *Lfather* node

t if $(s,t) \in \mathcal{E}^L$ and $(s,t) \notin \mathcal{E}^U$. Node s is a *Uson* of its *Ufather* node t if $(s,t) \notin \mathcal{E}^L$ and $(s,t) \in \mathcal{E}^U$. Finally, node s is an *LUson* of its *LUfather* node t if $(s,t) \in \mathcal{E}^L$ and $(s,t) \in \mathcal{E}^U$. The term *father* refers to any type of father, and the term *son* refers to any type of son.

Data flow edges are redundant if the data flow they represent can be accounted for by other edges in the graph. Redundant edge removal in the data flow graph is divided into two cases: removal of edges in $\mathcal{E}^L$ (case 1), and removal of edges in $\mathcal{E}^U$ (case 2). Case 2 is simply the transpose of case 1, so we only discuss case 1. For case 1 edge removal, consider the lower triangular part of the submatrix formed from rows and columns r, s, and t of $L \backslash U$ $(r < s < t)$,

$$\begin{bmatrix} p_r & \cdot & \cdot \\ l_{sr} & p_s & \cdot \\ l_{tr} & l_{ts} & p_t \end{bmatrix}.$$

For any given entry l_{sr} in L element E_r makes a contribution to row s, which later is assembled into the pivot row s of the element E_s (assuming r is not a singleton). Fill-in from pivot row r causes the column pattern of $E_r^{[s-1]}$ (just before step s) to be a subset of the column pattern of E_s,

$$\mathcal{U}_r^{[s-1]} \subseteq \mathcal{U}_s \text{ if } l_{sr} \neq 0. \qquad (4)$$

If there are two other entries l_{tr} and l_{ts} then both elements E_r and E_s compute a contribution to the pivot row t. Because of Equation 4, the contribution that E_r makes to row t can be assembled into the frontal matrix of E_s without changing the pattern of E_s. If this assembly is made, the data flow edge $(r,t) \in \mathcal{E}^L_{complete\_data}$ is removed. This is shown in the graph in Figure 4. The new graph, after redundant edge removal via cases 1 and 2, will be referred to as the *reduced data flow graph*, and denoted as $G_{rdata} = (\mathcal{V}, \mathcal{E}_{rdata})$.

The reduced data flow graph G_{rdata} can be partially characterized as follows. $\mathcal{E}_{rdata}$ is a subset of all edges

$$\{(r,s) \mid (r,s) \in \mathcal{E}_{complete\_data}, s \leq \mathit{firstpair}\,(r)\}$$

where *firstpair* (r) is the first off-diagonal pair in row r of U and column r of L,

$$\mathit{firstpair}\ \ (\mathrm{r}) = min\{t \mid l_{tr} u_{rt} \neq 0\}.$$

Some edges can satisfy cases 1 or 2 and precede the first off-diagonal pair (edge (r,t) in Figure 4, for example). Any given node in the reduced data

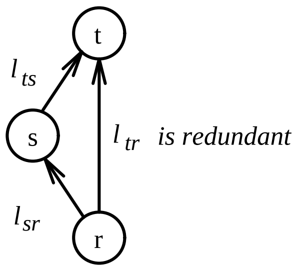

Figure 4: Case 1 edge removal in data flow and control flow graphs

flow graph has at most one LUfather (corresponding to the first off-diagonal pair), in addition to Lfathers and Ufathers corresponding to edges preceding the first off-diagonal pair.

The control flow graph can be simplified using the same cases 1 and 2. Many other edges, however, are redundant for describing the task precedence. Additional edge removal in the control flow graph falls into two cases: fill-in in L (case 3), and fill-in in U (case 4). Case 4 is the transpose of case 3, and so we only discuss case 3. Consider the following submatrix,

$$\begin{bmatrix} p_r & u_{rs} & \cdot \\ \cdot & p_s & \cdot \\ l_{tr} & l_{ts} & p_t \end{bmatrix}.$$

Two entries u_{rs} and l_{tr} force l_{ts} to be an entry in L. If l_{ts} was zero, it becomes a fill-in entry. If it was already an entry, it still is an entry since we ignore numerical cancellation. The entry l_{ts} creates a dependence from node s to node t in the control flow graph fragment shown in Figure 5, where $r < s < t$. Edge (r, t) is redundant to describe the control flow dependences between tasks r, s, and t and can be removed. The edge (r, t) would also be removed if transitive reduction [1] is applied to this fragment. The new graph, after edge removal via all four cases, will be referred to as the *reduced control flow graph*, and denoted as $G_{rcontrol} = (\mathcal{V}, \mathcal{E}_{rcontrol})$. In general, the reduced control flow graph is not sufficient to describe the data flow dependences.

The reduced control flow graph $G_{rcontrol}$ can be described as follows. Let $jmin$ be the column of the first off-diagonal entry in row r of U. Let $imin$ be the row of the first off-diagonal entry in column r of L. If $jmin < imin$

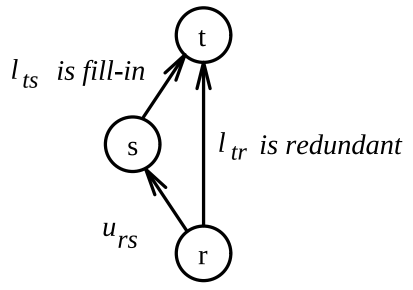

Figure 5: Case 3 edge removal in control flow graph only

then node r has no Lfathers, as shown in the following example submatrix,

$$\begin{bmatrix} p_r & u_{rs} & \cdot & u_{rv} \\ \cdot & p_s & \cdot & \cdot \\ l_{tr} & l_{ts} & p_t & u_{tv} \\ \cdot & \cdot & \cdot & p_v \end{bmatrix}, \tag{5}$$

where $imin = t$ and $jmin = s$.

All edges in $\mathcal{E}^L_{complete\_control}$ starting at r are removed via case 3 edge removal because of fill-in from the single entry $u_{r,jmin}$. Some, but not all, of the edges in $\mathcal{E}^U_{complete\_control}$ are also removed via case 4 edge removal. The edge $(r,v) \in \mathcal{E}^U_{complete\_control}$ is redundant because of the fill-in entry u_{tv}, and can be removed via case 4 edge removal for nodes r, t, and v. Thus, any edge $\{(r,v)|(r,v) \in \mathcal{E}^U_{complete\_control},\ v > imin\}$ is redundant. The node r will have Ufathers described by the set $\{s|u_{rs} \neq 0,\ s < imin\}$. Node r will have no LUfathers, since all edges in $\mathcal{E}^L_{complete\_control}$ starting at r are removed. Similarly, if $jmin > imin$ then node r has Lfathers in the set $\{s|l_{sr} \neq 0,\ s < jmin\}$. If $jmin = imin$, all but the first off-diagonal pair are removed via cases 1 and 2, and node r only has a single LUfather, $jmin$. Any node has either a single LUfather, or one or more Ufathers, or one or more Lfathers, but never a combination of these three types.

These two graphs are not necessarily minimal. For example, the redundant edge $(1,4)$ in the graph shown in Figure 6 will be present in the reduced graphs (where all edges are in $\mathcal{E}^L$). Transitive reduction [1] could be applied to $G_{complete\_control}$ to obtain a minimal $G_{rcontrol}$. Transitive reduction cannot be applied to $G_{complete\_data}$. For example, edge (r,t) is required in G_{rdata} in Figure 5, but is removed from $G_{complete\_data}$ by transitive reduction. Transitive reduction could be applied separately to the graphs $(\mathcal{V}, \mathcal{E}^L_{complete\_data})$

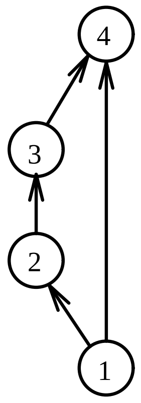

Figure 6: Reduced data flow graph with redundant edge (1,4)

and $(\mathcal{V}, \mathcal{E}^{U}_{complete\_data})$, and the results combined to give G_{rdata}. However, performing transitive reduction would introduce a significant computational overhead (specifically, $\mathcal{O}(n^{\log_2 7})$, which is much more than the time to factorize a sparse matrix). Although these graphs are not minimal, the four cases of edge removal yield a reduced graph with many fewer edges than the complete graph, without the cost of transitive reduction. Experimental results showing the level of reduction achieved are given in Table 2 and the accompanying discussion in Section 8. Eisenstat and Liu [16] describe similar reductions, which they refer to as *partial* transitive reductions. These reductions improve the assembly process, decrease the amount of memory needed to represent the graphs, simplify amalgamation, and improve the memory requirements for unassembled contributions. Both the reduced control flow graph and the reduced data flow graph are identical to the elimination tree if the pattern of $L \backslash U$ is symmetric.

4.1 Amalgamation

Both the factor-only algorithm (Section 6) and the analysis-factor algorithm (Section 7) require a representation of the reduced data flow and reduced control flow graphs. They also require amalgamation to be performed on these graphs. These operations can either be done by the analysis-only algorithm (Section 5) or by the analysis-factor algorithm.

A father node f and son node s can be amalgamated into a single super-

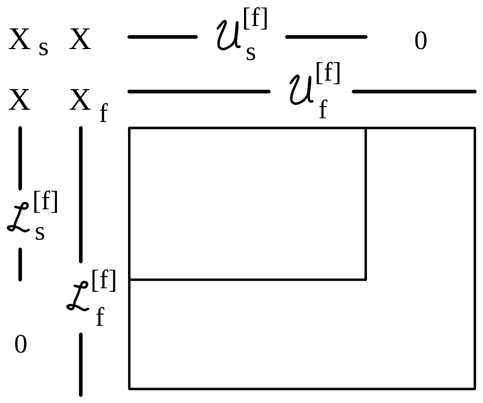

Figure 7: Fill-in due to amalgamation between LUson and LUfather

node s'. The amalgamated element $E_{s'}$ has row pattern $\mathcal{L}_s \cup \mathcal{L}_f$ and column pattern $\mathcal{U}_s \cup \mathcal{U}_f$. Extra fill-in due to amalgamation will occur if the pattern of $E_{s'}$ is not identical to the pattern of E_s.

If f is an LUfather of s, then extra fill-in can occur in row and column s, but not row or column f, nor in the contribution block, because

$$\mathcal{U}_s^{[f]} \subseteq \mathcal{U}_f^{[f]} \tag{6}$$

$$\mathcal{L}_s^{[f]} \subseteq \mathcal{L}_f^{[f]} \tag{7}$$

as shown in Figure 7. Fill-in does occur in columns $\mathcal{U}_f^{[f]} \backslash \mathcal{U}_s^{[f]}$ in row s, and in rows $\mathcal{L}_f^{[f]} \backslash \mathcal{L}_s^{[f]}$ in column s.

If f is only an Lfather of s, then the two pivot columns f and s are unrelated (Equation 6 holds, but not Equation 7), as shown in Figure 8. Fill-in can occur in columns f and s, and in row s, but not in row f. Fill-in can also occur in the contribution block of $E_{s'}$, specifically, the submatrix defined by columns $\mathcal{U}_f^{[f]} \backslash \mathcal{U}_s^{[f]}$ and rows $\mathcal{L}_s^{[f]} \backslash (\mathcal{L}_f^{[f]} \cap \mathcal{L}_s^{[f]})$ (outlined with a dashed box in Figure 8). Amalgamation between s and a Ufather f is the transpose of the Lfather case.

Extra fill-in due to amalgamation has different effects on the symbolic factorization computed before amalgamation is performed. Fill-in in the pivot rows and columns f and s has only a local effect on the graph. Edges between f and s are removed, edges that start at f or s now start at s', and edges that terminate at f or s now terminate at s'. If there is no fill-in in

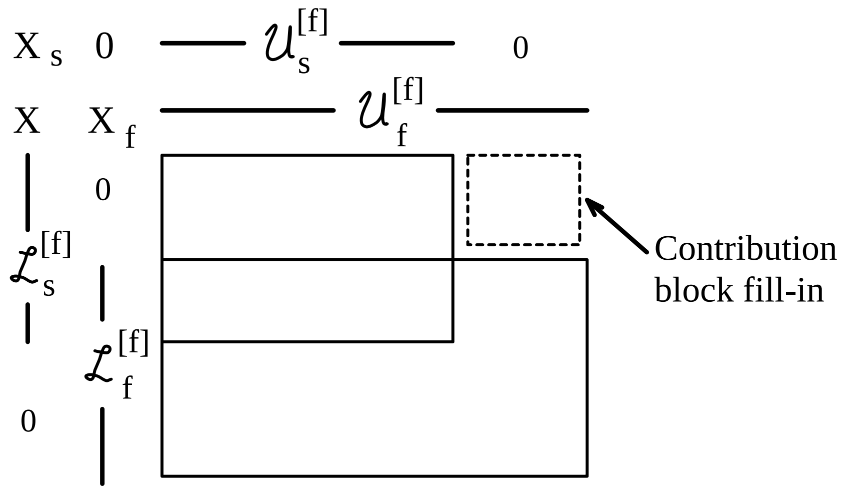

Figure 8: Fill-in due to amalgamation between Lson and Lfather

the contribution block, the patterns of the remaining factors L and U (in rows and columns $f+1$ through n) are unaffected. However, fill-in in the contribution block can cause a ripple effect in the remaining factors and their graphs, requiring a re-application of the symbolic factorization. Fill-in in the contribution blocks can be avoided by limiting amalgamation so that it never occurs. An alternative is to include amalgamation as an integral part of the symbolic factorization so that the extra fill-in due to amalgamation can be computed during symbolic factorization.

4.2 Representation of the data flow and control flow graphs

The pattern of L is stored in a column-oriented form as the sets $\{\mathcal{L}_t \,|\, t \in \mathcal{V}\}$, while the pattern of U is stored in a row-oriented form as the sets $\{\mathcal{U}_t | t \in \mathcal{V}\}$. For the factor-only algorithm, each set is sorted in pivot order. Together, the patterns of L and U represent the complete graph. The fathers of a node s in the reduced data flow and reduced control flow graphs are readily found from the patterns of L and U using the rules given above. Redundant edges in the complete data flow graph can be flagged by the symbolic phase, although only entries up to and including the first off-diagonal pair need to be scanned. However, the numerical phase also needs the sons of a node in the data flow graph, requiring access to the pattern of L by rows and to the pattern of U by columns. Scanning the column-oriented form of the pattern of L and the row-oriented form of the pattern of U in this order is inefficient. One alternative is to list in a separate, static, data structure the sons of each node in the reduced data flow graph, and their type (Lson, Uson, or LUson). The additional storage could be a significant overhead, as much as $O(\text{entries in } L + \text{entries in } U)$. This alternative is not suitable for the analysis-factor algorithm, since the pivot order and the patterns of L and U are not known in advance.

An alternative, dynamic, representation requires less storage than the static representation. Associated with each pivot entry is a pair of link lists, one for Lsons, and the other for Usons. The lists are empty at the start of the numerical phase. When element E_r is created, the Lson and Uson lists of its pivot entries contain the Lsons and Usons of node r in G_{rdata}. When E_r is factorized, a tuple is placed in the Lson lists for each row s in the pattern $\mathcal{L}_{2r}$ of the contribution block. The tuple placed in the Lson list s contains (r, q_s), where s is the q_s-th entry in $\mathcal{L}_{2r}$. The contribution that E_r makes to the s-th pivot row is located in the q_s-th row of D_r. Only tuples up to and including the first off-diagonal pair need to be placed. Operations for the Uson lists are similar.

Consider the following example LU factorization of a 5-by-5 matrix,

$$\begin{bmatrix} p_1 & \times & \cdot & \times & \times \\ \cdot & p_2 & \cdot & \cdot & \times \\ \times & \times & p_3 & \times & \times \\ \times & \times & \times & p_4 & \times \\ \cdot & \times & \cdot & \times & p_5 \end{bmatrix}$$

After the first element, E_1, is factorized, tuples are placed in the Lson and Uson lists as follows:

Pivot	Lson list	Uson list
2		$(1,1)$
3	$(1,1)$	
4	$(1,2)$	$(1,2)$
5		

List 5 is empty since the first off-diagonal pair for the first pivot is u_{14} and l_{41}. When element E_2 is factorized, the tuple $(1,1)$ is found in Uson list 2.

Thus, E_2 has one Uson E_1 whose contributions to pivot column 2 are found in column 1 of D_1. Tuples for E_2 are placed in the lists,

Pivot	Lson list	Uson list
3	$(1,1)(2,1)$	
4	$(1,2)(2,2)$	$(1,2)$
5	$(2,3)$	$(2,1)$

Case 1 edge removal can be done dynamically using the Lson lists. A edge is removed by marking its associated tuple. With the following submatrix of L,

$$\left[\begin{array}{lll} p_r & & \\ l_{sr} & p_s & \\ l_{tr} & l_{ts} & p_t \end{array}\right],$$

task r places the tuple (r, q_s) in Lson list s, and the tuple (r, q_t) in Lson list t. Task s assembles contributions from the unmarked tuples in its Lson list, and scans the Lson list of each node $t \in \mathcal{L}_s$. Task s finds a reference to its Lson node r in list t, marks the tuple, and assembles the corresponding row t of $E_r^{[s-1]}$ into E_s. If a reference to r is found in both the Lson and Uson list of node s, then s is an LUfather of r. Task s assembles the entire LUson $E_r^{[s-1]}$ into E_s, and removes all tuples placed by task r in the Lson and Uson lists. In either case, the edge (r,t) has been deleted from the graph (case 1 edge removal). The element E_r is deallocated if it no longer contains unassembled contributions. This is an example of how edge removal can decrease the amount of working storage required for the frontal matrices. Similarly, case 2 edge removal can be done using the Uson lists. Once assembly is complete, the lists for node s are discarded.

In the previous example LU factorization of a 5-by-5 matrix, case 1 edge removal occurs when element E_3 is factorized, with $r = 1$, $s = 3$, and $t = 4$. Task 3 finds a tuple from one of its Lsons (namely, tuple $(1,2)$ from node 1) in Lson list 4. This tuple is marked and the second row of D_1 (a contribution to row 4 of the matrix being factorized) is assembled into E_3. Task 3 can also mark the tuple $(2,2)$ in Lson list 4, and assemble the second row from D_2 into E_3 (case 1 edge removal, with $r = 2$, $s = 3$, and $t = 4$).

The unmarked tuples in all the lists at step k form a dynamic representation of the unassembled edges in the reduced data flow graph,

$$\{(s,t) \mid (s,t) \in \mathcal{E}_{rdata}, s < k < t\}.$$

This set of edges is smaller than the set of all edges in $\mathcal{E}_{rdata}$. With this second alternative, the symbolic phase described in the next section does not need to precompute the reduced data flow graph, and less memory is required to represent it.

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 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 elsewhere in this report. However, we can apply recent work of Duff and Reid [15] based on algorithms developed by Duff, Gould, Reid, Scott, and Turner [10] to obtain a suitable analysis. We discuss the use of their algorithms in this section. Methods based on the elimination dag are presented in [16, 23].

Duff et al. [10] 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

$$\begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix} \tag{8}$$

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. [10] is used to choose pivots from the coefficient matrix of Equation 8, 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. [10] to obtain a symbolic analysis of an unsymmetric matrix. The code of Duff and Reid [15] will also produce the equivalent of our data dependency directed acyclic graph 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 and graphs 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.

Task r does the following (for tasks $r \in \mathcal{V}$):

1. Wait until all sons of node r in the control flow graph have finished.

2. Create E_r and assemble the contributions represented by the edges in the data flow graph that terminate at node r into E_r. These are the Lsons, Usons, and LUsons of node r in the dynamic representation of the reduced data flow graph. Deallocate elements of sons that no longer have unassembled contributions. The row and column pattern of E_r is $\mathcal{L}_r$ and $\mathcal{U}_r$, 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_r into $L_{1r}\backslash U_{1r}$, and compute the block row U_{2r} of U and block column of L_{2r}, overwriting them in E_r. Then store them in a separate, statically-allocated data structure for L and U. This step tries to find g_r pivots within F_r, but might only be able to find g'_r pivots (where $0 \leq g'_r \leq g_r$). Replace g_r by the number of pivots actually found ($g_r \leftarrow g'_r$).

4. Compute the update terms with a rank-g_r update to the contribution block D_r, using the Level-2 BLAS if $g_r = 1$, or Level-3 if $g_r > 1$. Parallelism can occur within these kernels, as well as between independent tasks.

5. If node r is the last son to complete for a father t (in the control flow graph), then enable task t.

Numerical pivoting considerations might not allow the expected number of pivots to be chosen from a pivot block F_r. The work associated with the failed pivots must be performed later. This can be regarded as a forced amalgamation of the the failed pivots with one or more fathers of r in the data flow and control flow graphs, with any fill-in constraints removed.

In the classical multifrontal method, the failed pivots are amalgamated with the single father node s of r. Task s then attempts to perform both its own g_s steps of LU factorization and that of any failed pivots of its sons. Numerical pivoting causes only local changes in the elimination tree and in the patterns of L and U, although these changes can ripple up if the pivots also fail in the father node. The changes are limited to the failed pivot rows and columns. This case also occurs in the unsymmetric-pattern multifrontal method if node r has a single LUfather and no Lfathers or Ufathers. Otherwise, larger disruptions can occur.

If a node r has either Lfathers or Ufathers in the data flow graph, and is found to have numerically unacceptable pivots, the effects are not limited to a single pair of nodes. In the following example, node s is the Lfather of a simple node r with a single failed pivot,

$$\begin{bmatrix} p_r & \cdot & u_{rt} \\ l_{sr} & p_s & u_{st} \\ l_{tr} & \cdot & p_t \end{bmatrix}.$$

One option is to amalgamate nodes r and s, as was done for numerical pivoting failures in the classical multifrontal method. However, this causes fill-in in the contribution block of E_s, since the pivot rows are not likely to be identical. This fill-in causes far-reaching effects in the data flow and control flow graphs between node r and the root node. Catastrophic fill-in and loss of parallelism in the control flow graph can result.

The second option for recovering from numerical pivoting failures is to amalgamate node r with its single LUfather t, assuming it exists. This has the effect of reordering the matrix so the pivot p_r follows p_t,

$$\begin{bmatrix} p_s & u_{st} & u_{sr} \\ \cdot & p_t & u_{tr} \\ \cdot & l_{rt} & p_r \end{bmatrix}.$$

Limited fill-in occurs in the intermediate nodes between node r and node t that are Lfathers and Ufathers of node r. In this example, the column pattern $\mathcal{U}_s$ of element E_s is augmented by including the failed pivot column r (because of entry u_{sr}). The fill-in in column r from E_s (assuming p_s is not a column singleton) is

$$\mathcal{L}_r \leftarrow \mathcal{L}_r \cup \mathcal{L}_s.$$

If node s was instead a Ufather of node r, then the row pattern $\mathcal{L}_s$ of element E_s would be augmented by the single failed pivot row r. The fill-in in row r from E_s would be

$$\mathcal{U}_r \leftarrow \mathcal{U}_r \cup \mathcal{U}_s.$$

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

In general, node r is shifted past each intervening Lfather and Ufather node, augmenting each Lfather by column r and each Ufather by row r. Each Lfather s causes fill-in with pattern $\mathcal{L}_s$ in column r, and each Ufather s causes fill-in with pattern $\mathcal{U}_s$ in row r. Finally, node r is amalgamated with its single LUfather, node t. Amalgamation with the LUfather t does not cause fill-in in the contribution block of E_t because $\mathcal{L}_t$ already contains the pattern $\mathcal{L}_s$ of any Lfather node s of r, even without amalgamation. This can be seen in the example above (prior to reordering). If s is an Lfather of r, and t is an LUfather of r, then u_{st} is nonzero (because of fill-in) and $\mathcal{L}_s \subseteq \mathcal{L}_t$. Similarly, $\mathcal{U}_t$ is a superset of the pattern $\mathcal{U}_s$ of any Ufather node s of r.

The data flow and control flow graphs are only locally modified, but this option breaks down if node r does not have an LUfather t. In this case, node r is delayed until the end of the factorization. Fill-in is caused in row and column r by every ancestor of node r. 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 elimination 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 precomputed data flow and control flow graphs to guide the construction of new elements, the assembly process, and the exploitation of parallelism. Instead, the algorithm must compute the graphs and the patterns of L and U during factorization. With a parallel pivot search, these symbolic computations must also be done in parallel.

7.1 Data structures

The key to this algorithm is an extension to the dynamic representation of the reduced data flow graph presented in Section 4.2. At step k in the factorization, only pivots 1 through k are defined. The pair of linked lists associated with each pivot t cannot be defined for pivots $t > k$. Instead, a linked list is associated with each non-pivotal row i (an Lson list) and with each non-pivotal column j (a Uson list).

When task r creates and factorizes E_r, it places a tuple (r, q) in the Lson list of each row $i \in \mathcal{L}_{2r}$, where i is the q-th entry in $\mathcal{L}_{2r}$. It also places a tuple (r, q) in the Uson list of each column $j \in \mathcal{U}_{2r}$. When task s starts $(r < s)$, its Lsons are defined by the tuples in the Lson lists of the g_s pivot rows of E_s (including both marked and unmarked tuples). Dynamic edge removal via cases 1 and 2 can still be performed. Task s scans the Lson list of each row $i \in \mathcal{L}_s$. If it finds a reference to an Lson r of node s, then the tuple (r, q) is marked, and the contribution to row i in E_r (found in the q-th row of D_r) is assembled into E_s. This is an example of case 1 edge removal. If a tuple referring to node r is found in both the Lson and Uson lists of s then s is an LUfather of r. Task s assembles the entire $E_r^{[s-1]}$ into E_s.

Consider the following 5-by-5 matrix after two steps of LU factorization, where the third pivot entry is shown as p_3, but is not yet permuted to the third row and column.

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

The Lson and Uson lists before the factorization of E_3 are:

Row/Column	Lson list of row	Uson list of column
3	$(2,1)$	$(1,2)$
4	$(1,1)(2,2)$	
5	$(1,2)(2,3)$	$(1,3)(2,2)$

Task 3 finds the pivot, p_3, in row 5 and column 4, thus its Lson list is $((1,2),(2,3))$, while its Uson list is empty. During assembly, it finds these two tuples in the Lson list of row 4 corresponding to its Lson nodes 1 and 2. These tuples are marked, and the contributions of E_1 and E_2 to row 4 are assembled into E_3.

All data structures are allocated out of a single pair of one-dimensional real and integer arrays. The original matrix A is stored in both row and column form at the beginning of the two arrays, followed by various workspaces and data structures of size n. The end of the two arrays holds a stack containing the pattern and numerical values of the LU factors. These only grow in size during factorization. The space between the original matrix and fixed arrays and the LU factors holds the frontal matrices and Lson and Uson lists. The frontal matrices and the Lson and Uson lists are allocated when elements are created and deallocated when their corresponding contribution block has been completely assembled. These form the main dynamic data structures in the algorithm.

7.2 Algorithm

The analysis-factor algorithm first initializes the data structures for the allocatable memory, the original matrix A, and the pivot search procedure. It then factorizes the matrix A into the product $PAQ = LU$, where P and Q are the row and column permutations due to the pivot ordering. The lower and upper triangular systems are then solved to arrive at the solution x to the original problem $Ax = b$. The factorization forms the major part of the algorithm, and can be outlined as follows. Initially, $k = 1$.

1. The pivot search finds the pivot $a_{ij}^{[k-1]}$ in A_{k-1} 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 Lsons, Usons, and LUsons of node k are located in the Lson list i and Uson list j.

2. If the newly created element is about to exhaust the remaining unallocated memory, perform garbage collection. Allocate the c_k-by-r_k frontal matrix E_k and store the pivot row and column in it.

3. Completely assemble LUsons of node k into E_k. Assemble columns from Usons and rows from Lsons. Look for super-columns and super-rows. Permute the n_{row} super-rows and n_{col} super-columns to the upper-left of the frontal matrix, and change $\mathcal{L}_k$ and $\mathcal{U}_k$ to reflect this. Let $g_k = min(n_{row}, n_{col})$. Deallocate any elements whose contributions have been completely assembled.

4. Perform up to g_k steps of numerical factorization within the front and save the computed rows and columns of U and L. Set g_k to the number of pivots actually found.

5. Perform degree update and update the Lson and Uson lists. Increment k by g_k and repeat until the matrix is factorized.

7.2.1 Pivot search

The pivot search is based on Markowitz' strategy [26], which selects the pivot $a_{ij}^{[k-1]}$ with minimum upper bound on fill-in (or *cost*),

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

Scanning a row i of $A^{[k-1]}$ involves scanning row i of the active part of A and unmarked tuples in the Lson list i. Adding these terms gives the structure and numerical values of row i in A_{k-1}. 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. When, and if, the true degree is calculated, the two bounds are set equal to the true degree. Only the first few columns with minimum upper bound degree are searched [12, 28], 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.

The pivot $a_{kk}^{[k-1]}$ at step k must satisfy the threshold partial-pivoting criterion [9]:

$$|a_{kk}^{[k-1]}| \geq u \cdot \max_{k \leq i \leq n} |a_{ik}^{[k-1]}|, \ \ 0 < u \leq 1. \qquad (9)$$

The candidate pivot is the numerically acceptable entry $a_{ij}^{[k-1]}$ with lowest approximate Markowitz cost using the true column degree and the upper bound row degree.

7.2.2 Assembly and dynamic amalgamation

The frontal matrix $E_s^{[k-1]}$ of any LUson node s is assembled into E_k. To minimize the amount of indirect addressing necessary for the numerical additions, offsets are computed for the pivot row and column k. A single scatter operation assembles a row of $E_s^{[k-1]}$ into E_k. Without this offset, both a scatter and a gather would be needed. This approach is taken in the classical multifrontal method [3]. The row and column offset vectors R and C for E_k are initialized as

$$R(\mathcal{L}_k(q)) = q, \ q = 1 \cdots |\mathcal{L}_k|$$
$$C(\mathcal{U}_k(q)) = q, \ q = 1 \cdots |\mathcal{U}_k|,$$

then adding the d-th row of E_s into E_k is done as follows:

```
$q = R(\mathcal{L}_s(d))$
for $e = g_s + 1$ to $|\mathcal{U}_s|$
    $E_k(q, C(\mathcal{U}_s(e))) = E_k(q, C(\mathcal{U}_s(e))) + E_s(d, e)$
    endfor
```

One or more columns $j \in \mathcal{U}_k$ are assembled from each Uson, following the dynamic representation of the reduced data flow graph discussed in Section 7.1. The assembly takes advantage of the pivot column offsets computed for the LUson assembly. If all tuples in the Uson list of column j are marked, column j of the active part of A is scanned, and a count m is made of entries lying outside the row pattern, $\mathcal{L}_k$. If $m = 0$, then the structure of column j is identical to the structure of the pivot column k, and column j of the active part of A is assembled into E_k. Columns k and j form a super-column. Otherwise, column j is not a super-column of column k, but its true degree can be computed. After factorization,

$$c_j^{[k+g_k-1]} = |\mathcal{L}_{2k}| + m. \tag{10}$$

The assembly of Lsons and the search for super-rows is similar.

7.2.3 Numerical factorization within the frontal matrix

Pivots are constrained to the upper-left n_{row}-by-n_{col} submatrix, since only these rows and columns are fully assembled. They are selected with threshold partial-pivoting (Equation 9). The first step of the numerical factorization of E_k finds up to g_k pivots and updates the left c_k-by-n_{col} submatrix of E_k (thereby updating D'_{1k}). The rest of the frontal matrix is not modified by this step, except by the pivot permutations. The current version uses only the Level-2 BLAS for this step (Level-3 BLAS should be used if g_k is large enough). g_k is set to the actual number of pivots found. At this point, E_k has been partially factorized into

$$\left[\begin{array}{ccc} L_{1k}\backslash U_{1k} & U_{21k} & B_{2k} \\ L_{2k} & D'_{1k} & D_{2k} \end{array}\right].$$

The matrices $L_{1k}\backslash U_{1k}$ are the g_k-by-g_k factorization of F_k. The first $n_{col} - g_k$ columns of U_{2k} and D'_k are denoted as U_{21k} and D'_{1k}, respectively. The second step of numerical factorization updates B_{2k} and D_{2k}, using the Level-3 BLAS if $g_k > 1$. The lower-triangular system

$$L_{1k}U_{22k} = B_{2k}$$

is solved for U_{22k}, and B_{2k} is overwritten by U_{22k}. Finally, D_{2k} is overwritten with

$$D'_{2k} = D_{2k} - L_{2k}U_{22k}.$$

The g_k rows and columns of L_k and U_k are copied from E_k into the data structure for L and U. This data remains after E_k is deallocated.

7.2.4 Partial degree update

Unless the true degree can be computed using Equation 10 (or its counterpart for the row degree), the upper and lower bounds of the degrees of each row $i \in \mathcal{L}_{2k}$ and column $j \in \mathcal{U}_{2k}$ are found by scanning their Lson and Uson lists, respectively. Let $t = k + g_k - 1$. The new lower bound on the row degree $r_i^{[t]}$ is the largest of the following:

1. the previous lower bound minus g_k, since the degree cannot drop by more than the number of reductions performed on the row,
2. one, since the matrix is assumed to be non-singular,
3. $|\mathcal{U}_{2k}|$, which is the number of columns in the contribution block D_k, and also the upper bound on fill-in in row i,
4. the number of entries in row i of the active part of A, and
5. $\max(|\mathcal{U}_s^{[t]}|)$, for each element s appearing in the Lson list of row i.

If the true degree is ever computed, it is likely that the first term will be the largest. The new upper bound on the row degree can be computed in a similar way. It is the smallest of the following:

1. the previous upper bound plus $|\mathcal{U}_{2k}|$, since the row degree cannot increase by more than the upper bound on the fill-in in row i,
2. $n - t$, which is the size of the active submatrix, and
3. $|\mathcal{U}_{2k}| + \sum_s |\mathcal{U}_s^{[t]}|$, for each element s appearing in the Lson list of row i.

As with the lower bound computation, it is likely that the first term will be the smallest if the true degree is ever computed. These computations are much less than those required to find the true degrees.

7.3 Summary of the analysis-factor algorithm

The analysis-factor algorithm is based on a dynamic representation of the reduced data flow graph 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. Dynamic amalgamation is made possible by the detection of super-rows and super-columns. Thus, the algorithm can take advantage of the Level-3 BLAS. It does not suffer from disruptions in the graphs or in the patterns of L and U caused by numerical pivoting, as does the factor-only algorithm. Parallelism is not yet addressed; this and other issues are dealt with in Section 9, which presents open problems and future work. The following section compares the performance of the analysis-factor algorithm with that of the D2 algorithm and the classical multifrontal method.

8 Performance results

The analysis-factor algorithm has been implemented and tested on a parallel minisupercomputer, the Alliant FX/80. The FX/80 has eight processors, each with a peak performance of 23.6 megaflops. Since a fully parallel version is not yet developed, only one processor is used for the performance comparisons, even though both D2 and the classical multifrontal method are parallel algorithms. Amestoy and Duff's version of the classical multifrontal method is used for the comparisons [3]. Each algorithm was used to factorize a set of 39 test matrices from the Harwell/Boeing sparse matrix collection [11], and the results for five typical matrices are shown in Table 1.

The table describes the five matrices by name, discipline from which they are derived, size, number of nonzero entries, and *asymmetry*. The asymmetry of a matrix A is the number of unmatched off-diagonal entries over the total number of off-diagonal entries. An unmatched entry is an entry a_{ij} for which a_{ji} is zero. The next section of the table compares the performance of MA28 [12], the D2 algorithm (with and without the switch to dense code) [5], the classical multifrontal method (Amestoy-Duff) [3], and the analysis-factor algorithm based on the unsymmetric-pattern multifrontal method. The D2 algorithm switches to dense code when a pivot set of less than four pivots is found and when the density of A_k is greater than 20%. The Amestoy-Duff algorithm uses a form of amalgamation that allows a controlled amount of extra fill-in. The number of nonzeros in L and U, the number of floating-point operations required to factorize the matrix (in millions), and the run time in seconds are compared. The run time includes both the symbolic analysis phase and the numerical factorization phase for all three algorithms. These two phases are combined in the case of D2 and the new analysis-factor algorithm, and are separate in the Amestoy-Duff algorithm. The last section of the table shows a timing profile of the new algorithm. The total run time is divided into the run time for initializing the original matrix and pivot search data structures, the pivot search (step 1 of the algorithm outlined in Section 7.2), garbage collection and allocation of frontal matrices (step 2), assembly and dynamic amalgamation (step 3), numerical factorization within the frontal matrices (step 4), and the partial degree update (step 5).

Several conclusions can be made from these results. First, the analysis-factor algorithm finds an LU factorization with a reasonable number of nonzeros compared with MA28. The MA28 algorithm uses the true Markowitz criterion, while the analysis-factor algorithm uses a modified cost based on the upper bound of the row and column degrees. The modified method simplifies the degree update while only slightly degrading the performance in terms of fill-in. In fact, the method has less fill-in than the classical multifrontal method for all but the nearly-symmetric matrix lns3937 (another reason for this difference is the relaxed amalgamation in the classical multifrontal algorithm). It always has less fill-in than the D2 algorithm (with the switch). For this reason, it also requires fewest floating point operations to factorize the matrix, except for the lns3937 matrix.

Although the new analysis-factor method is never the fastest for these matrices, it is often faster than the slowest of the other two (D2 with switch and Amestoy-Duff). The implementation requires refinement, as discussed in the next section. The floating-point operations in the analysis-factor algorithm are divided between the assembly (step 3) and the numerical factorization (step 4). Most of the floating-point work is done in step 4, which uses dense matrix kernels. Step 4 is faster than step 3, which relies on gather/scatter operations. These steps together take up about half the total run time. The degree update and pivot search take up the bulk of the rest of the time.

Table 2 shows how effective amalgamation and edge removal are in simplifying the data flow and control flow graphs for the same five matrices. With no amalgamation, the number of edges in the complete graph ($G_{complete}$) is simply the number of off-diagonal entries in L and U. Amalgamation and case 1 and 2 edge removal is applied to obtain the reduced data flow graph, G_{rdata}. More edges can be removed from G_{rdata} using cases 3 and 4 to obtain the reduced control flow graph, $G_{rcontrol}$. As is apparent from the table, G_{rdata} and $G_{rcontrol}$ have many fewer edges than $G_{complete}$. Transitive reduction could reduce the number of edges even further, but at an unacceptable

Table 1: Performance comparisons and profile of analysis-factor algorithm

Matrix					
name	lns3937	sherman5	mahindas	gemat11	gre1107
discipline	fluid flow	geology	economics	power	computer
order	3937	3312	1258	4929	1107
nonzeros	25407	20793	7682	33108	5664
asymmetry	0.15	0.26	0.98	1.00	1.00
Nonzeros in $L\backslash U$					
MA28	417603	163468	10783	51729	44980
D2 (no switch)	433952	216644	12109	56788	43818
D2 (switch)	705240	308405	13200	61888	68305
Amestoy-Duff	292231	177818	41632	62387	181143
analysis-factor	533829	152011	11230	58457	53324
operations (10^6)					
D2 (switch)	174.7	39.7	0.1	1.0	3.6
Amestoy-Duff	21.6	16.0	2.2	0.5	24.0
analysis-factor	98.3	13.9	0.1	0.6	3.2
time (seconds)					
D2 (switch)	101.7	40.9	1.2	11.3	4.5
Amestoy-Duff	12.0	8.9	7.6	5.7	7.2
analysis-factor	129.9	20.3	4.5	10.7	10.2
analysis-factor timing profile (%):					
initializations	0.1	0.5	0.9	1.3	0.3
1. pivot search	9.8	19.6	23.5	30.7	20.5
2. allocation	12.9	5.5	2.0	3.5	4.8
3. assembly	44.9	36.9	39.2	27.2	39.5
4. numerical	16.8	17.7	14.2	20.5	12.7
5. degree update	15.5	19.8	20.2	16.8	22.2

Table 2: Reduced data flow and control flow graphs

Matrix					
name	lns3937	sherman5	mahindas	gemat11	gre1107
order	3937	3312	1258	4929	1107
b, irreducible blocks	351	1675	670	352	1
edges in $G_{complete}$	529892	148699	9972	53528	52217
edges in G_{rdata}	32842	10954	1827	6444	5707
edges in $G_{rcontrol}$	9987	3355	1279	2777	2803
lower bound, $\vert\mathcal{V}\vert - b$	3380	1390	551	2476	975

computational cost. The lower bound on the number of edges that transitive reduction could obtain for a data flow or control flow graph with $|\mathcal{V}|$ nodes is $|\mathcal{V}| - b$ (where b is the number of irreducible blocks in A [8], although the edge counts were computed without permuting the matrix A to block upper triangular form).

9 Summary

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 version of the analysis-factor algorithm. Some parallel work can take place within the dense matrix kernels, and while this will be important in the final version, 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 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. 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 ($\mathcal{L}_{2k}$ and $\mathcal{U}_{2k}$, 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 for many helpful discussions.

References

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

[2] G. Alaghband and H. F. Jordan, *Sparse Gaussian elimination with controlled fill-in on a shared memory multiprocessor*, IEEE Trans. Comput., 38 (1989), pp. 1539-1557.

[3] P. R. Amestoy and I. S. Duff, *Vectorization of a multiprocessor multifrontal code*, Internat. J. Supercomputer Appl., 3 (1989), pp. 41-59.

[4] T. A. Davis and E. S. Davidson, *Pairwise reduction for the direct, parallel solution of sparse unsymmetric sets of linear equations*, IEEE Trans. Comput., 37 (1988), pp. 1648-1654.

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

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

[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 (1988), pp. 1-17.

[8] I. S. Duff, *On permutations to block triangular form*, J. Inst. Math. Applic., 19 (1977), pp. 339-342.

[9] I. S. Duff, A. M. Erisman, and J. K. Reid, Direct Methods for Sparse Matrices, Oxford University Press, Oxford, 1986.

[10] I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner, *Factorization of sparse symmetric indefinite matrices*, IMA J. Numer. Anal., 11 (1991), pp. 181–204.

[11] I. S. Duff, R. G. Grimes, and J. K. Reid, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1-14.

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

[13] —, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Softw., 9 (1983), p. 302-325.

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

[15] —, *MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems*, To appear, 1991.

[16] S. C. Eisenstat and J. W. H. Liu, *Exploiting structural symmetry in unsymmetric sparse symbolic factorization*, Report CS-90-12, Dept. of Computer Science, York University, North York, Ontario, Canada, Nov. 1990.

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

[18] A. George and J. W. H. Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[19] —, *The evolution of the minimum degree ordering algorithm*, SIAM Review, 31 (1989), pp. 1-19.

[20] A. George and E. Ng, *An implementation of Gaussian elimination with partial pivoting for sparse systems*, SIAM J. Sci. Statist. Comput. 6 (1985), pp. 390-409.

[21] —, *Parallel sparse Gaussian elimination with partial pivoting*, Oak Ridge National Laboratory, Oak Ridge, TN, 1988.

[22] J. R. Gilbert, *An efficient parallel sparse partial pivoting algorithm*, 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*, Report CS-90-11, Dept. of Computer Science, York University, North York, Ontario, Canada, Feb. 1991.

[24] J. R. Gilbert and T. Peierls, *Sparse partial pivoting in time proportional to arithmetic operations*, SIAM J. Sci. Statist. Comput., 9 (1988) pp. 862-874.

[25] J. W. H. Liu, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134-172.

[26] H. M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Science, 3 (1957), pp. 255-269.

[27] O. Wing and J. W. Huang, *A computational model of parallel solution of linear equations*, IEEE Trans. Comput., 29 (1980), pp. 632-638.

[28] Z. Zlatev, J. Wasniewski, and K. Schaumburg, *Y12M: Solution of large and sparse systems of linear algebraic equations*, Lecture Notes in Computer Science, 121, Springer-Verlag, Berlin, 1981.