

RC 22137 (99131) August 1, 2001
Computer Science/Mathematics

IBM Research Report


Improved Symbolic and Numerical Factorization Algorithms for Unsymmetric Sparse Matrices

Anshul Gupta

IBM Research Division
T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598
anshul@watson.ibm.com

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

 **Research Division**
Almaden · Austin · China · Delhi · Haifa · Tokyo · Watson · Zurich

IMPROVED SYMBOLIC AND NUMERICAL FACTORIZATION ALGORITHMS FOR UNSYMMETRIC SPARSE MATRICES

ANSHUL GUPTA*

Abstract. We present algorithms for the symbolic and numerical factorization phases in the direct solution of sparse unsymmetric systems of linear equations. We have modified a classical symbolic factorization algorithm for unsymmetric matrices to inexpensively compute minimal elimination structures. We give an efficient algorithm to compute a near-minimal data-dependency graph that is valid irrespective of the amount of dynamic pivoting performed during numerical factorization. Finally, we describe an unsymmetric-pattern multifrontal algorithm for Gaussian elimination with partial pivoting that uses the task- and data-dependency graphs computed during the symbolic phase. These algorithms have been implemented in WSMP—an industrial strength sparse solver package—and have enabled WSMP to significantly outperform other similar solvers. We present experimental results to demonstrate the merits of the new algorithms.

Key words. sparse matrix algorithms, Gaussian elimination, LU factorization, multifrontal methods

AMS subject classifications. 05C20,05C75,65F05,65F50,65Y15

1. Introduction. The Watson Sparse Matrix Package (WSMP) [18] is a high-performance and robust software package for solving general sparse systems of linear equations using direct factorization of the coefficient matrix A into triangular factors L and U . As is typical for such solvers, WSMP has four distinct phases, namely, *Analysis* comprising ordering for fill-in reduction and symbolic factorization, *Numerical Factorization* using Gaussian elimination with partial pivoting, *Forward and Backward Elimination* to solve the system using the triangular factors, and *Iterative Refinement*. In this paper, we describe some key algorithms that WSMP uses in its unsymmetric symbolic and numerical factorization phases. These algorithms are crucial to WSMP’s performance, which is significantly better than other similar solvers [16].

In the case of symmetric sparse matrices, an elimination tree is the task- and data-dependency graph for the factorization process. However, for unsymmetric matrices, the task- and data-dependency graph are directed acyclic graphs (DAGs). Moreover, the edge-set of the minimal data-dependency graph or data-DAG for unsymmetric sparse factorization can be a superset of the edge-set of a task-dependency DAG or task-DAG. In [15], Gilbert and Liu describe elimination structures for unsymmetric sparse LU factors and give an algorithm for sparse unsymmetric symbolic factorization. These elimination structures are two DAGs that are transitive reductions of the graphs of the factor matrices L and U , respectively, and can be used to derive a task-DAG for sparse LU factorization. Some researchers have argued that computing an exact transitive reduction can be too expensive [8, 14] and have proposed using subminimal DAGs with more edges than necessary. Traversing unnecessary DAG edges during numerical factorization can be a source of overhead. Moreover, in a parallel implementation, extra DAG edges can be potential sources of unnecessary synchronization or communication.

In this paper, we show how an easily implementable modification to Gilbert and Liu’s symbolic factorization algorithm enables an efficient computation of the minimal elimination DAGs. We also define a set of edges that must be added to the task-DAG

*Mathematical Sciences Department, IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (anshul@watson.ibm.com).

in order to generate a minimal data-DAG that is valid as long as partial pivoting with dynamic row and column exchanges is not performed during factorization. Finally, we describe how supplementing this data-DAG further with a small set of extra edges can yield a near-minimal data-DAG that is sufficient to handle an arbitrary number of pivot failures and the resulting row and column exchanges during numerical factorization. By means of experiments on a suite of unsymmetric sparse matrices from real applications, we show that computing the final data-DAG is extremely fast. Furthermore, for the matrices in our test suite, this data-DAG has only a slightly higher number of edges than the task-DAG constructed using complete transitive reduction.

The multifrontal method [13, 23] for solving sparse systems of linear equations usually offers a significant performance advantage over more conventional factorization schemes by permitting efficient utilization of parallelism and memory hierarchy. In [13], Duff and Reid describe a symmetric-pattern multifrontal algorithm for unsymmetric matrices that generates an elimination tree based on the structure of $A + A^T$ to guide the numerical factorization, which works on symmetric frontal matrices. This conventional multifrontal algorithm can incur a substantial overhead for very unsymmetric matrices due to unnecessary data dependencies in the elimination tree and due to extra zeros in the artificially symmetrized frontal matrices. Davis and Duff [8] and Hadfield [21] introduced an unsymmetric-pattern multifrontal algorithm that overcomes the deficiencies of a symmetric-pattern algorithm. Our powerful symbolic phase enables us to use a much simplified and computationally lean version of the unsymmetric-pattern multifrontal algorithm with partial pivoting. We describe the unsymmetric-pattern multifrontal algorithm that is used in WSMP and experimentally compare it with other state-of-the-art sparse unsymmetric factorization codes.

In Table 1.1, we introduce the suite of test matrices that we will use in experiments throughout this paper. All matrices in our test suite arise in real-life problems and are publicly available. Table 1.1 shows the size of each matrix, the number of nonzeros in it, and the application area of the origin of the matrix. All experiments reported in this paper were conducted on an IBM RS6000 WH-2 with a 375 MHz Power3 CPU, 2 Gbytes of RAM, 8 Mbytes of level-2 cache, and 64 Kbytes of level-1 cache.

The organization of this paper is as follows. We introduce the terms, conventions, and notations used in the paper in §2. A symbolic factorization algorithm that computes the structure of the triangular factors and minimal elimination structures is described in §3. In §4, we give fast algorithms to compute near-minimal data-DAGs for unsymmetric multifrontal factorization. The numerical factorization algorithm is described in detail in §5. We finish with concluding remarks in §6. The last subsection of each of the major sections contains experimental results pertaining to the algorithms in that section.

2. Terminology and conventions. We assume that the original $n \times n$ sparse unsymmetric coefficient matrix is irreducible and cannot be permuted into a block-triangular form. This is not a serious restriction, because a general matrix can first be reduced to a block-triangular form and then only the irreducible diagonal blocks need to be factored [11]. We assume that the coefficient matrix A is factored into a lower triangular matrix L and an upper triangular matrix U . Multiple row and column permutations may be applied to A during various stages of the solution process. However, for the sake of clarity, we will always denote the coefficient matrix by A and the factors by L and U . The state of permutation of A , L , and U will usually be clear from the context.

We denote the directed graph corresponding to an $n \times n$ matrix M by

| <i>Number</i> | <i>Matrix</i> | <i>N</i> | <i>NNZ</i> | <i>Application</i> |
|---------------|---------------|----------|------------|-------------------------|
| 1 | af23560 | 23560 | 484256 | Fluid dynamics |
| 2 | av41092 | 41092 | 1683902 | Finite element analysis |
| 3 | bayer01 | 57735 | 277774 | Chemistry |
| 4 | bbmat | 38744 | 1771722 | Fluid dynamics |
| 5 | comp2c | 16783 | 578665 | Linear programming |
| 6 | e40r0000 | 17281 | 553956 | Fluid dynamics |
| 7 | e40r5000 | 17281 | 553956 | Fluid dynamics |
| 8 | ecl32 | 51993 | 380415 | Circuit simulation |
| 9 | epb3 | 84617 | 463625 | Thermodynamics |
| 10 | fidap011 | 16614 | 1091362 | Fluid dynamics |
| 11 | fidapm11 | 22294 | 623554 | Fluid dynamics |
| 12 | invextr1 | 30412 | 1793881 | Fluid dynamics |
| 13 | mil053 | 530238 | 3715330 | Structural engineering |
| 14 | mixtank | 29957 | 1995041 | Fluid dynamics |
| 15 | nasasrb | 54870 | 2677324 | Structural engineering |
| 16 | onetone1 | 36057 | 341088 | Circuit simulation |
| 17 | onetone2 | 36057 | 227628 | Circuit simulation |
| 18 | pre2 | 659033 | 5959282 | Circuit simulation |
| 19 | raefsky3 | 21200 | 1488768 | Fluid dynamics |
| 20 | raefsky4 | 19779 | 1316789 | Fluid dynamics |
| 21 | rma10 | 46835 | 2374001 | Fluid dynamics |
| 22 | tib | 18510 | 145149 | Circuit simulation |
| 23 | twotone | 120750 | 1224224 | Circuit simulation |
| 24 | wang3 | 26064 | 177168 | Circuit simulation |
| 25 | wang4 | 26068 | 177196 | Circuit simulation |

TABLE 1.1

Test matrices with their order (N), number of nonzeros (NNZ), and the application area of origin.

$G_M(V_M, E_M)$, where $V_M = \{1, 2, \dots, n\}$. Although a graph may not always be associated with an explicitly defined matrix, but when it is, then an edge $\vec{ij} \in E_M$ if and only if m_{ij} is a structural nonzero entry in the sparse matrix M . The transpose of a matrix M is represented by M' . If $\vec{ij} \in E_M$, then $\vec{ji} \in E_{M'}$, and vice-versa.

$\text{Struct}(M_{i,*})$ is the set of columns in M that have a structural nonzero entry in row i . This is also the set of all vertices to which i has an outbound edge in G_M . Similarly, $\text{Struct}(M_{*,i})$ is the set of rows in M that have a structural nonzero entry in column i and is also the set of all vertices from which i has an inbound edge in G_M . A directed path from node i to node j in the directed graph G_M is denoted by \vec{ij} . The graph G_{M^O} is a transitive reduction of the graph G_M if G_{M^O} has a directed path \vec{ij} if and only if G_M has a directed path \vec{ij} and there is no other graph with fewer edges than G_{M^O} that satisfies this condition. Since we are primarily dealing with the nonzero structure of matrices rather than the actual values, we may also loosely refer to M^O as the transitive reduction of M if G_{M^O} is a transitive reduction of G_M . The leading $i \times i$ submatrix of M is denoted by M_i and the corresponding graph and its transitive reduction by G_{M_i} and $G_{M_i^O}$, respectively.

The edges and paths in the some of the graphs used in this paper are labeled. An edge can have one of the three labels—L, U, or LU. Depending on its label, an edge can be an L-edge, a U-edge, or an LU-edge. L-, U-, and LU-edges from vertex i to j are denoted by $\overset{L \rightarrow}{ij}$, $\overset{U \rightarrow}{ij}$, and $\overset{LU \rightarrow}{ij}$, respectively. An L-path from i to j , denoted

for $i = 1$ to n **do**

1. Compute $\text{Struct}(L_{i,*})$ from $\text{Struct}(A_{i,*})$ by traversing $G_{U_{i-1}^O}$ and using the fact that $\forall j < i, j \in \text{Struct}(L_{i,*})$ if and only if $\exists k \leq j$ such that $k \in \text{Struct}(A_{i,*})$ and $\exists \tilde{k} j \in E_{U_{i-1}^O}$.
2. Transitively reduce $\text{Struct}(L_{i,*})$ using $G_{L_{i-1}^O}$ and extend it to $G_{L_i^O}$.
3. Compute $\text{Struct}(U_{i,*}) = ((\bigcup_{j:ji \in E_{L_i^O}} \text{Struct}(U_{j,*})) \cup \text{Struct}(A_{i,*})) - \{1, 2, \dots, i-1\}$.
4. Transitively reduce $\text{Struct}(U_{*,i})$ using $G_{U_{i-1}^O}$ and extend it to $G_{U_i^O}$.

end for

FIG. 3.1. Gilbert and Liu's unsymmetric symbolic factorization algorithm [15].

by $\overset{L \rightsquigarrow}{ij}$, is a directed path containing only L- and LU-edges. Similarly, a U-path from i to j , denoted by $\overset{U \rightsquigarrow}{ij}$, is a directed path containing only U- and LU-edges. If an L-edge $\overset{L \rightarrow}{ij}$ exists in the graph, then $j = \text{L-parent}(i)$. Similarly, if $\overset{U \rightarrow}{ij}$ exists, then $j = \text{U-parent}(i)$ and if $\overset{LU \rightarrow}{ij}$ exists, then $j = \text{LU-parent}(i)$.

We define¹ a supernode $[q : r]$ as a maximal set of consecutive indices $\{q, q + 1, \dots, r\}$ such that $\forall i \in [q : r], \text{Struct}(L_{*,i}) = \text{Struct}(L_{*,q}) - \{q, q + 1, \dots, i - 1\}$ and $\text{Struct}(U_{i,*}) = \text{Struct}(U_{q,*}) - \{q, q + 1, \dots, i - 1\}$. For $n \times n$ matrices L and U , we define $m \times m$ supernodal matrices \mathbf{L} and \mathbf{U} such that each supernode $[q : r]$ in L and U is represented by a single row and column $g = \mathcal{S}([q : r])$ in \mathbf{L} and \mathbf{U} . Here $m \leq n$ is the total number of supernodes. Furthermore, if $g = \mathcal{S}([q : r]), h = \mathcal{S}([s : t])$, and $r < s$, then $g < h$; i.e., the column and row indices in \mathbf{L} and \mathbf{U} maintain the relative order of supernodes in L and U .

3. Computing a task-DAG and the structures of L and U . In [15], Gilbert and Liu present an unsymmetric symbolic factorization algorithm to compute the structures of the factors L and U and their transitive reductions L^O and U^O . Fig. 3.1 summarizes Gilbert and Liu's algorithm. The algorithm computes the structure of L , U , and L^O row by row and computes the structure of U^O by columns.

The total time that the algorithm shown in Fig. 3.1 spends in Step 1 is bounded by $\text{flops}(LU^O)$ [15], which is the number of operations required to multiply the sparse matrices L and U^O . Similarly, the time spent in Step 3 is bounded by $\text{flops}(UL^O)$. The total computational cost of Steps 2 and 4 is $O(n(|E_{L^O}| + |E_{U^O}|))$. This is because transitive reduction is performed on n rows of U and columns of L , and the i -th step could potentially traverse all edges in $G_{L_i^O}$ and $G_{U_i^O}$. Steps 2 and 4 of Gilbert and Liu's algorithm are much costlier than steps 1 and 3. The cost of these steps has prompted researchers to seek alternatives, such as computing fast but incomplete transitive reduction [8, 14]. Such use of alternatives to G_{L^O} and G_{U^O} with more edges than G_{L^O} and G_{U^O} , respectively, can increase the cost of Steps 1 and 3, as well as that of numerical factorization.

¹Other definitions of supernodes in the context of unsymmetric sparse factorization have been used in the literature [10].

for $i = 1$ to n **do**
 1. Transitively reduce $\text{Struct}(L_{i,*})$ using $G_{L_{i-1}^O}$ and extend it to $G_{L_i^O}$.
 2. Compute $\text{Struct}(U_{i,*}) = ((\bigcup_{j:j^i \in E_{L_i^O}} \text{Struct}(U_{j,*}) \cup \text{Struct}(A_{i,*})) - \{1, 2, \dots, i-1\})$.
 3. Transitively reduce $\text{Struct}(U_{*,i})$ using $G_{U_{i-1}^O}$ and extend it to $G_{U_i^O}$.
 4. Compute $\text{Struct}(L_{*,i}) = ((\bigcup_{j:j^i \in E_{U_i^O}} \text{Struct}(L_{*,j}) \cup \text{Struct}(A_{*,i})) - \{1, 2, \dots, i-1\})$.
end for

FIG. 3.2. A modified symbolic factorization algorithm.

3.1. A modification to Gilbert and Liu’s algorithm. We now describe a relatively simple modification of the algorithm shown in Fig. 3.1. We start by splitting the original coefficient matrix into a lower triangular part stored by columns and an upper triangular part stored by rows. In our modified symbolic factorization algorithm, we compute the structure of L by the columns (i.e., L' by rows) and that of U by the rows. This is achieved by simply reformulating the algorithm shown in Fig. 3.1 to perform only Steps 2 and 3, but twice for each i on two sets of identical data structures—one corresponding to L' and the other corresponding to U . The modified algorithm is shown in Fig. 3.2.

Note that Steps 3 and 4 of the algorithm shown in Fig. 3.2 are identical to Steps 1 and 2, respectively. The first two steps compute the i -th rows of L^O and U and the last two steps compute the i -th columns of U^O and L . An actual code of this algorithm can use the same pair of routines with different arguments to implement all four steps. The reduction in the size of the code by half, however, is a secondary benefit of the modified algorithm. The primary advantage of this scheme is that it allows immediate detection of supernodes during symbolic factorization. This, as we shall explain in §3.2, allows us to avoid computing and storing G_{L^O} and G_{U^O} explicitly. Instead, we can work only with their supernodal counterparts G_{L^O} and G_{U^O} , respectively.

3.2. Use of supernodes to speed up transitive reduction. Most modern sparse factorization codes rely heavily on supernodes to efficiently utilize memory hierarchies and parallelism in the hardware. Supernodes are so crucial to high performance in sparse matrix factorization that the criterion for the inclusion of rows and columns in the same supernode is often relaxed [5] to increase the size of the supernodes. Consecutive rows and columns with nearly same but not identical structures are often included in the same supernode and artificial nonzero entries with a numerical value of 0 are added to maintain identical row and column structures for all members of a supernode. The rationale is that the slight increase in the number of nonzeros and floating-point operations involved in factorization is more than compensated for by higher factorization speed.

WSMP’s LU factorization algorithm also works on the relaxed supernodes generated by its symbolic factorization. In the symbolic factorization algorithm, as soon as $\text{Struct}(L_{*,i})$ and $\text{Struct}(U_{i,*})$ are computed in the i -th iteration of the outer loop, they can be compared with $\text{Struct}(L_{*,i-1})$ and $\text{Struct}(U_{i-1,*})$ to determine if they belong to the current supernode. A new row-column pair is added to the current supernode

if its structure is either identical or nearly identical to the previous row-column pair. If the i -th row-column pair fails to meet the criterion for membership into the current supernode, then a new supernode is started at i .

The use of supernodes allows us to significantly reduce the cost of computing the transitive reductions. In Step 1 of the algorithm shown in Fig. 3.2, instead of transitively reducing the entire $\text{Struct}(L_{i,*})$, we only reduce the set $\{h : h = \mathcal{S}([q:r]), \text{ where } [q:r] \subseteq \text{Struct}(L_{i,*})\}$. Step 3 is treated similarly. As a result of working only with supernodes, the upper bound on the cost of computing the transitive reduction decreases from $O(n(|E_{L^O}| + |E_{U^O}|))$ to $O(n(|E_{L^O}| + |E_{U^O}|))$. This is because only the supernodal DAGs G_{L^O} and G_{U^O} are searched during each of the n transitive reduction steps. Strict supernodal graphs G_{L^O} and G_{U^O} would have at least $n - m$ fewer edges than G_{L^O} and G_{U^O} , where m is the number of supernodes. This is because U^O and L^O do not contain any edges \vec{ij} , where $j = i + 1$, $q \leq i < r$, and $[q:r]$ is a supernode. The use of relaxed supernodes reduces the number of edges even further. As a part of the process of supernode-relaxation, a node $i + 1$ can be forced into the same supernode that contains node i , even if row and column $i + 1$ contain indices that are not present in row and column i . In order to accommodate node $i + 1$, these extra indices are added to the supernode that is absorbing node $i + 1$. As a result, some potential edges of the form \vec{ij} , where $j > i + 1$, may be eliminated from the task-DAG.

3.3. Task-DAGs for LU factorization. We first define a task-DAG $Tdag^C$ in terms of the conventional structures L^O and U^O . The transpose matrix L' is used to indicate that $\forall \vec{ij} \in E_{Tdag^C}, j > i$.

THEOREM 3.1. *$Tdag^C$ is a task-DAG for LU factorization if its vertex set $V_{Tdag^C} = \{1, 2, \dots, n\}$ and its edge-set $E_{Tdag^C} = E_{U^O} \cup E_{L'^O}$.*

Proof. To prove that $Tdag^C$ is a task-DAG, we show that E_{Tdag^C} is sufficient to represent a proper ordering of the n elimination tasks denoted by V_{Tdag^C} . $\text{Struct}(L_{*,i})$ can contribute to $\text{Struct}(L_{*,j})$ only if $i \in \text{Struct}(U_{*,j})$, and if this is the case, then the symbolic factorization algorithm of Fig. 3.2 ensures that U^O contains either \vec{ij} or $\vec{\tilde{ij}}$. The same is true for $\text{Struct}(U_{i,*})$, $\text{Struct}(U_{j,*})$, and L'^O . Therefore, every row-column pair i that updates row-column j , must be eliminated before j . \square

Theorem 3.1 can be easily extended to the supernodal case. A supernodal task-DAG $Tdag^S$ is defined by a vertex set $V_{Tdag^S} = \{1, 2, \dots, m\}$ and an edge set $E_{Tdag^S} = E_{U^O} \cup E_{L'^O}$, where m is the number of supernodes.

Although, in a practical implementation, we always work with supernodal DAGs, we will often use conventional task- and data-DAGs in the remainder of the paper to keep the exposition simple. All results and descriptions presented in terms of the conventional DAGs map naturally to the supernodal case.

3.4. Experimental results. In Table 3.1, we compare Gilbert and Liu's symbolic factorization algorithm [15] with the supernodal symbolic factorization algorithm described in §3.2. We report their CPU times T^C and T^S , respectively and the number edges in task DAGs $Tdag^C$ and $Tdag^S$ generated by them.

The last column of Table 3.1 shows the factor by which the supernodal symbolic factorization is faster than the conventional algorithm. The table also shows average supernode size (n/m) and the ratio of edges in $Tdag^C$ and $Tdag^S$ for each matrix. These two ratios are closely related. The ratio of T^C and T^S bears some correlation to the ratio of edges in $Tdag^C$ and $Tdag^S$, but the actual ratio is matrix dependent.

| Matrix | V (n) | N_{sup} (m) | Conventional Symbolic | | Supernodal Symbolic | | $\frac{n}{m}$ | $\frac{ E_{Tdag^C} }{ E_{Tdag^S} }$ | $\frac{T^C}{T^S}$ |
|----------|-----------|------------------|--------------------------|----------------|------------------------|----------------|---------------|-------------------------------------|-------------------|
| | | | T^C | $ E_{Tdag^C} $ | T^S | $ E_{Tdag^S} $ | | | |
| af23560 | 23560 | 4744 | 2.6 | 23644 | .47 | 4793 | 5.0 | 4.9 | 5.5 |
| av41092 | 41086 | 19302 | 3.1 | 62197 | .83 | 34708 | 2.1 | 1.8 | 3.7 |
| bayer01 | 48803 | 33891 | .45 | 113948 | .28 | 87028 | 1.4 | 1.3 | 1.6 |
| bbmat | 38744 | 4877 | 10. | 41260 | 1.7 | 6077 | 7.9 | 6.8 | 5.9 |
| comp2c | 6756 | 996 | .66 | 8005 | .22 | 1736 | 6.8 | 4.6 | 3.0 |
| e40r0000 | 17281 | 3049 | .47 | 19262 | .14 | 3225 | 5.7 | 6.0 | 3.4 |
| e40r5000 | 17281 | 2755 | .60 | 19891 | .16 | 3182 | 6.3 | 6.3 | 3.8 |
| ecl32 | 42341 | 12087 | 7.9 | 48779 | 1.2 | 15239 | 3.5 | 3.2 | 6.6 |
| epb3 | 84617 | 30009 | 1.2 | 106137 | .50 | 38088 | 2.8 | 2.8 | 2.4 |
| fidap011 | 16614 | 1262 | 2.3 | 16613 | .42 | 1261 | 13. | 13. | 5.5 |
| fidapm11 | 22294 | 2327 | 3.7 | 23144 | .65 | 2651 | 9.6 | 8.7 | 5.7 |
| invextr1 | 30412 | 5295 | 4.5 | 37685 | .93 | 10108 | 5.7 | 3.8 | 4.8 |
| mil053 | 530238 | 166155 | 15. | 530237 | 4.5 | 166154 | 3.2 | 3.2 | 3.3 |
| mixtank | 29957 | 2984 | 7.8 | 30949 | 1.2 | 3203 | 10. | 9.7 | 6.5 |
| nasasrb | 54870 | 3808 | 4.9 | 54869 | .97 | 3807 | 14. | 14. | 5.1 |
| onetone1 | 32211 | 14215 | 1.1 | 45227 | .31 | 23585 | 2.3 | 1.9 | 3.5 |
| onetone2 | 32211 | 14843 | .44 | 45073 | .18 | 23999 | 2.2 | 1.9 | 2.4 |
| pre2 | 629628 | 243693 | 30. | 765210 | 6.4 | 317216 | 2.6 | 2.4 | 4.7 |
| raefsky3 | 21200 | 1282 | 2.1 | 21199 | .41 | 1281 | 17. | 17. | 5.1 |
| raefsky4 | 19779 | 1359 | 2.9 | 19778 | .50 | 1358 | 15. | 15. | 5.8 |
| rma10 | 46835 | 3855 | 2.1 | 47152 | .56 | 3911 | 12. | 12. | 3.8 |
| tib | 17583 | 7823 | .11 | 22904 | .07 | 10060 | 2.2 | 2.3 | 1.6 |
| twotone | 105740 | 34304 | 2.6 | 126656 | .91 | 44856 | 3.1 | 2.8 | 2.9 |
| wang3 | 26064 | 8451 | 3.1 | 26063 | .54 | 8450 | 3.1 | 3.1 | 5.7 |
| wang4 | 26068 | 8254 | 3.0 | 26067 | .53 | 8253 | 3.2 | 3.2 | 5.7 |

TABLE 3.1

Comparison of conventional symbolic factorization (due to Gilbert and Liu [15]) with supernodal symbolic factorization. $|V|$ is the size of the largest diagonal block in the matrix on which symbolic factorization is performed, N_{sup} is the number of supernodes; T^C and T^S are the times in seconds of the two symbolic factorization algorithms; and, $|E_{Tdag^C}|$ and $|E_{Tdag^S}|$ are the number of edges in the task-DAGs produced by the two algorithms.

Note that only the time of transitive reduction steps 1 and 3 of the algorithm in Fig. 3.2 is reduced by the use of supernodes; the time of computing the structures of L and U in steps 2 and 4 remains mostly unchanged (other than some reduction in the number of structures merged due to supernode relaxation). Therefore, the actual reduction achieved in symbolic factorization time depends on the relative amounts of time spent in transitive reduction and computing L and U structures. Moreover, Table 3.1 reports the number of edges in the task-DAGs, not the number of edges in the actual lower and upper triangular transitively reduced graphs that are traversed during symbolic factorization. Recall that the edge-set of a task-DAG is the union of the edge-sets of the corresponding lower and upper triangular transitively reduced graphs. The amount of structural symmetry in the matrix affects the number of common edges between the upper and lower transitively reduced graphs, which in turn determines the actual number of edges in the task-DAG.

In [14], Eisenstat and Liu present an alternative to complete transitive reduction to reduce the cost of this step in sparse unsymmetric symbolic factorization. They propose exploiting structural symmetry in the matrix to compute partial tran-

sitive reductions. Although they present experimental results on a different set of much smaller matrices, it appears that the use of supernodes, as proposed in §3.2 can achieve much higher speedups in symbolic factorization while computing exact transitive reductions than the partial transitive reduction scheme proposed in [14].

4. Data-DAGs for unsymmetric multifrontal LU factorization. The multifrontal method [13, 23] for sparse matrix factorization usually offers a significant performance advantage over more conventional factorization schemes by permitting efficient utilization of parallelism and memory hierarchy. The original multifrontal algorithm was described in the context of a symmetric-pattern coefficient matrix and has been applied to matrices with unsymmetric patterns by introducing zero-valued entries at appropriate locations to convert the original matrix into one with the pattern of $A + A^T$ [13, 1, 2]. This can cause a substantial overhead for very unsymmetric matrices due to the extra computation performed on the introduced entries and the resulting fill-in. Davis and Duff [8] and Hadfield [21] introduced an unsymmetric-pattern multifrontal algorithm to overcome this shortcoming of the symmetric-pattern multifrontal algorithm. In this section, we develop near-minimal data-DAGs for the unsymmetric multifrontal algorithm—an aspect of unsymmetric multifrontal factorization that has not been well investigated in previous works. As we shall show in §5, the availability of a near minimal data-DAG aids an efficient implementation of the numerical factorization phase. It would also help minimize the synchronization and communication overheads in a parallel implementation of the unsymmetric-pattern multifrontal algorithm.

4.1. Outline of the symmetric multifrontal algorithm. The symmetric-pattern multifrontal algorithm is guided by an assembly or elimination tree [22, 23, 19], which serves as both the task- and the data-dependency graph for the factorization process. The data associated with each supernode of the elimination tree is a square frontal matrix. A frontal matrix F^g associated with a supernode $g = \mathcal{S}([q : r])$ is a dense matrix whose dimensions are equal to $|\text{Struct}(L_{*,q})|$ or $|\text{Struct}(U_{q,*})|$. The contiguous local row and column indices in the dense frontal matrix correspond to noncontiguous global indices of the matrix $L + U$. Each entry in a frontal matrix corresponds to a structural nonzero entry in the global matrix. After a frontal matrix F^g is fully assembled or populated, the leading $r - q + 1$ rows and columns corresponding to the supernode are factored and become parts of the factors U and L , respectively. The remaining trailing part of the square matrix is now called the update or the contribution matrix, denoted by C^g . The contribution matrix corresponding to a supernode is assembled completely into the frontal matrix of its only parent supernode, and is never assessed again. This is because if $h = \mathcal{S}([s : t])$ is the parent of supernode $g = \mathcal{S}([q : r])$ in the elimination tree, then $\text{Struct}(L_{*,r}) - \{r\} \subset \text{Struct}(L_{*,s})$. The same is true for columns of U due to symmetry. The task corresponding to the multifrontal elimination at a supernode first recursively completes identical subtasks for each of its children in the elimination tree, then assembles their contribution matrices into its frontal matrix, and finally performs the partial factorization on the frontal matrix. Calling a recursive procedure to perform the task described above on the root supernode of the elimination tree completes the factorization of a sparse matrix with a symmetric structure.

4.2. Outline of the unsymmetric multifrontal algorithm. The overall structure of an unsymmetric-pattern multifrontal algorithm is similar to its symmetric counterpart and can be expressed in the form of a recursive procedure starting at

the root (the supernode with no outgoing edges) of the task-DAG. However, there are two major differences. The first difference is in the control-flow. In the unsymmetric multifrontal algorithm, before starting a subtask for a child, the task corresponding to the parent supernode must check to see if the child supernode has already been processed by another parent. Only the first parent to reach a child actually performs the recursive computation starting at that child. The second difference is in the data-flow, or the way contribution matrices are assembled into frontal matrices. This is explained below in greater detail.

Recall that the edge-set E_{Tdag^C} of the task-DAG $Tdag^C$ is the union of the edge-set $E_{L' \circ}$ of the transitive reduction of L' and the edge-set $E_{U \circ}$ of the transitive reduction of U . We now assign labels to the edges in $Tdag^C$. The edges contributed to E_{Tdag^C} solely by $E_{L' \circ}$ are labeled as L-edges. Similarly, edges contributed to E_{Tdag^C} solely by $E_{U \circ}$ are labeled as U-edges. The third type of label, the LU-label, is assigned to the edges that belong to both $E_{L' \circ}$ and $E_{U \circ}$. Finally, an L-edge $\overset{L \rightarrow}{ij}$ is converted to an LU-edge $\overset{LU \rightarrow}{ij}$ if there is a U-path $\overset{U \rightsquigarrow}{ij}$ in $Tdag^C$ and a U-edge $\overset{U \rightarrow}{ij}$ is converted to $\overset{LU \rightarrow}{ij}$ if there is an L-path $\overset{L \rightsquigarrow}{ij}$. The edges of the supernodal task-DAG $Tdag^S$ can be defined similarly.

Unlike the symmetric multifrontal algorithm, the frontal and contribution matrices in the unsymmetric multifrontal algorithm are, in general, rectangular rather than square. Furthermore, a contribution matrix in the unsymmetric multifrontal algorithm can potentially be assembled into more than one frontal matrices because a supernode in the data-DAG can have more than one parent. As described in [21], the assembly of contribution matrices into the parent frontal matrices in the unsymmetric multifrontal algorithm proceeds as follows.

Let $\overset{L \rightarrow}{gh}$ be an L-edge in the data-DAG, where $g = \mathcal{S}([q:r])$ and $h = \mathcal{S}([s:t])$. If $\text{Struct}(L_{*,q})$ and $\text{Struct}(L_{*,s})$ have an index i in common, then all elements of row i of U in C^g can potentially be assembled into F^h . Similarly, if $\overset{U \rightarrow}{gh}$ is a U-edge and $\text{Struct}(U_{q,*})$ and $\text{Struct}(L_{s,*})$ have an index i in common, then all elements of column i of L in C^g can potentially be assembled into F^h . Finally, if $\overset{LU \rightarrow}{gh}$ is a LU-edge, then the entire trailing submatrix of C^g with global row and column indices greater than or equal to s can be assembled into F^h .

Certain entries of C^g may have potential destinations in the frontal matrices of more than one parent of g even if the data-DAG contains no unnecessary edges. This is because C^g can have common rows (columns) with the frontal matrices of more than one among g 's LU- and L-parents (U-parents). The unsymmetric multifrontal algorithm must ensure that any entry of a contribution matrix is not used to update more than one frontal matrices. Additionally, a correct data-DAG must have sufficient outgoing edges from all supernodes so that each entry of a contribution matrix has a potential destination in at least one frontal matrix.

4.3. Inadequacy of task-DAG for unsymmetric multifrontal algorithm.

By means of a small example in Fig. 4.1, we show that if the task-DAG defined in §3.3 is used as a data-DAG, then all contribution matrices may not get fully absorbed into their parent frontal matrices. The figure shows a sparse matrix with factorization fill-in, the transitively reduced DAGs L^O and U^O , and the task-DAG with its edges labeled as described in §4.2. For the sake of simplicity, each supernode is of size 1. The figure shows all frontal and contribution (shaded portions) matrices and the flow

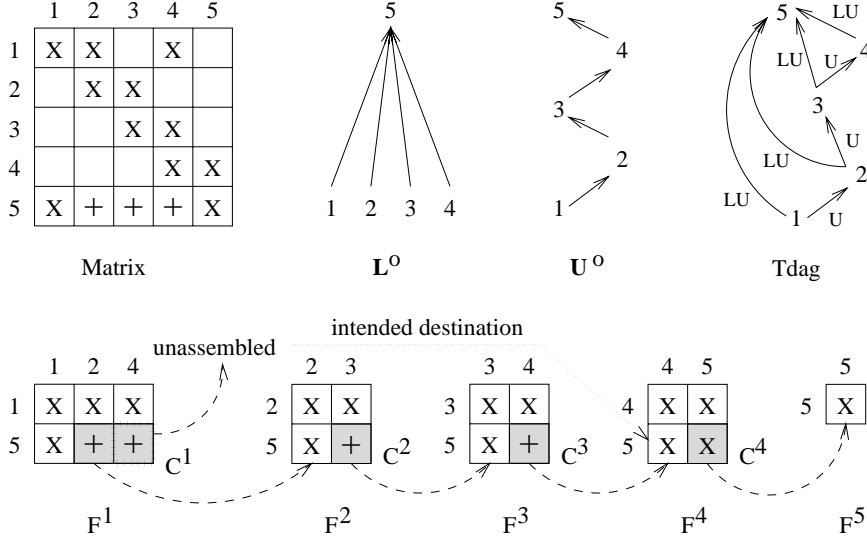


FIG. 4.1. An example to show the inability of a task-DAG to guide complete assembly of all contribution matrices in the unsymmetric multifrontal algorithm. An 'X' denotes a nonzero in the coefficient matrix and a '+' denotes a nonzero created due to fill-in.

of data from the contribution to frontal matrices along the edges of the task-DAG. Note that all edges may not lead to a data transfer; e.g., $1 \xrightarrow{LU} 5$. It is easily seen that the U-edge $1 \xrightarrow{U} 4$ that is absent from the task-DAG (because it is removed while transitively reducing \mathbf{U} to \mathbf{U}^O) is necessary for the complete assembly of C^1 .

4.4. A data-DAG for a predefined pivot sequence. Having shown that the task-DAG cannot serve as the data-DAG for unsymmetric multifrontal factorization, we now define a data-DAG that is sufficient for the proper assembly of all contribution matrices as long as rows and columns are not exchanged among different supernodes for pivoting. We will use $Ddag^N$ to denote such a DAG, where the superscript N stands for 'no pivoting'. A data-DAG $Ddag^P$ that can accommodate pivoting will be described in §4.5.

THEOREM 4.1. *If (1) $j \in Struct(U_{i,*})$, (2) the LU-parent of i , if it exists, is greater than j , (3) none of i 's U-parents are in $Struct(U_{*,j})$, and (4) $\exists k \in Struct(L_{*,i})$ such that $k > j$, then a U-edge ij is necessary for C^i to be completely assembled into its parents' frontal matrices. The transpose of this theorem can be stated similarly.*

Proof. The contribution matrix C^i has a column that contributes to $L_{*,j}$, because, at the least, there is an element corresponding to $L_{k,j}$ in C^i . At the same time, none of i 's U-parents' frontal matrices have column j , so they cannot absorb $L_{*,j}$ from C^i . Since the LU-parent of i is greater than j , it too cannot absorb $L_{*,j}$ from C^i . The addition of ij makes it possible to contribute $L_{*,j}$ from C^i to F^j . The transpose case can be proven similarly. \square

Theorem 4.1 captures the situation illustrated in Fig. 4.1 and prescribes the addition of $1 \xrightarrow{U} 4$ to ensure complete assembly of C^1 .

THEOREM 4.2. *If $Ddag^N$ is a DAG formed by adding all possible edges according to Theorem 4.1 to $Tdag^C$ if these edges don't already exist, then $Ddag^N$ is a data-*

dependency DAG for unsymmetric multifrontal algorithm without pivoting.

Proof. To show that $Ddag^N$ is a data-DAG, we must show that its edge-set is sufficient for the complete absorption of all contribution matrices into their parent frontal matrices. We prove this by contradiction.

Without loss of generality, assume that an element corresponding to $L_{k,j}$ in C^i is not assembled. Note that $i < j < k$. If $L_{k,j}$ is in C^i , then $j \in \text{Struct}(U_{i,*})$ and $k \in \text{Struct}(L_{*,i})$. Since $j \in \text{Struct}(U_{i,*})$, either $i \overset{U \rightarrow}{j} \in E_{Tdag^C}$ or there is a U-path $i \overset{U \rightsquigarrow}{j}$ in $Tdag^C$. If $i \overset{U \rightarrow}{j} \in E_{Tdag^C}$, then all entries with row indices greater than or equal to j in column j of C^i will be absorbed by F^j , and these entries include the one corresponding to $L_{k,j}$. If $i \overset{U \rightarrow}{j} \notin E_{Tdag^C}$, then a U-path $i \overset{U \rightsquigarrow}{j}$ exists in $Tdag^C$ and there are two possibilities: either $\text{LU-parent}(i) \leq j$ or $\text{LU-parent}(i) > j$. Let $l = \text{LU-parent}(i)$. If $l \leq j$, then the entire trailing submatrix of C^i with row and column indices greater than l , including $L_{k,j}$, will be assembled into F^l . If $l > j$, then consider two further possibilities: either one of i 's U-parents is in $\text{Struct}(U_{*,j})$ or not. If one is, then its frontal matrix will absorb column j from C^i . If none of i 's U-parents is in $\text{Struct}(U_{*,j})$, then all conditions for the applicability of Theorem 4.1 are satisfied.

Therefore, $i \overset{U \rightarrow}{j}$ would have been added to $Ddag^N$ and would have caused the entry corresponding to $L_{k,j}$ in C^i to be absorbed into F^j . Thus, it is not possible for the entry corresponding to $L_{k,j}$ to be left unassembled in any C^i . Similarly, it can be shown that the entry corresponding to any $U_{j,k}$ cannot be left unassembled in any C^i . \square

Having shown that the edge-set of $Ddag^N$ is sufficient for unsymmetric multifrontal factorization without pivoting, we now show that all edges that $Ddag^N$ inherits from $Tdag^C$ may not be necessary if pivoting is not performed during factorization.

THEOREM 4.3. *For LU factorization without pivoting, an edge $i \overset{U \rightarrow}{j}$ ($i \overset{L \rightarrow}{j}$) or $i \overset{LU \rightarrow}{j}$ in $Tdag^C$ is redundant if the maximum index in $\text{Struct}(L_{*,i})$ ($\text{Struct}(U_{i,*})$) is smaller than j .*

Proof. Recall that $\text{Struct}(L_{*,j}) = ((\cup_{i:ij \in E_{U^O}} \text{Struct}(L_{*,i})) \cup \text{Struct}(A_{*,j})) - \{1, 2, \dots, j-1\}$. If the maximum index in $\text{Struct}(L_{*,i})$ is smaller than j , then $\text{Struct}(L_{*,i}) \subseteq \{1, 2, \dots, j-1\}$ and does not contribute to $\text{Struct}(L_{*,j})$. The proof for L^O and $\text{Struct}(U_{i,*})$ is similar. \square

Note that Theorem 4.3 is valid only if row and column exchanges are not performed during LU factorization. Otherwise, additional fill-in caused by pivoting could create an index greater than or equal to j in $\text{Struct}(L_{*,i})$ or $\text{Struct}(U_{i,*})$, even if it is not predicted by the symbolic factorization on the original permutation of the matrix. Therefore, all edges in $Tdag^C$ could potentially be used.

Supernodal versions of Theorems 4.1–4.3 for $Tdag^S$ can be proven similarly. To summarize the results of this subsection, we have shown how to construct a data-DAG for unsymmetric multifrontal factorization without pivoting from a task-DAG and we have shown that although the task-DAG is derived from the strict transitive reductions of L' and U (or \mathbf{L}' and \mathbf{U}), it may still pass on edges to the data-DAG that are redundant if pivoting is not performed during factorization. Therefore, the data-DAG is not minimal. However, if pivoting is performed, then potentially all the edges could get used.

4.5. Supplementing the data-DAG for dynamic pivoting. We will now show that the edge-set of data-DAG $Ddag^N$ constructed in §4.4 is not sufficient if piv-

oting is performed during factorization and will describe how to supplement E_{Ddag^N} to generate a data-DAG $Ddag^P$ whose edge-set is sufficient to handle any amount of pivoting. We start with an overview of the pivoting methodology in the unsymmetric multifrontal algorithm, which has been described in detail in [21].

If a diagonal element $A_{i,i}$ ($q \leq i \leq r$) in a supernode $[q : r]$ fails to meet the pivoting criterion, then first an attempt is made to exchange row and column i with a row j and a column k such that $i < j \leq r$, $i < k \leq r$ and $A_{j,k}$ satisfies the pivoting criterion. Such intra-supernode pivoting has no effect on the structure of the factors and factorization can continue as usual. However, it may not always be possible to find a suitable row-column pair within a supernode's pivot block to satisfy the pivoting criterion. In such a situation, inter-supernode pivoting is necessary. If $h = \mathcal{S}([s : t])$ is the LU-parent of $g = \mathcal{S}([q : r])$ in the data-DAG and a suitable i -th pivot cannot be found within the pivot block of F^g , then all row-column pairs from i to r are symmetrically permuted to new locations from $s - (r - i + 1)$ to $s - 1$. Thus, effectively, supernode $[q : r]$ shrinks to $[q : i - 1]$ and the supernode $[s : t]$ expands to $[s - (r - i + 1) : t]$. As a side effect of this pivoting, there is additional fill-in in all the ancestors of g in the data-DAG that are smaller than h . In particular, the columns of L of all of g 's U-ancestors smaller than h get extra row indices $[i : r]$ and the rows of U of all of g 's L-ancestors smaller than h get extra column indices $[i : r]$.

In $Ddag^N$, whose construction is described in §4.4, all supernodes may not have an LU-parent to support the symmetric pivoting method described above. Therefore, as the first step towards deriving $Ddag^P$ from $Ddag^N$, we alter the edge-set of the latter as follows. For each g from 1 to m (where m is the total number of supernodes), the smallest supernode h to which both $\overset{L \rightsquigarrow}{gh}$ and $\overset{U \rightsquigarrow}{gh}$ exist is designated as the LU-parent of g ; i.e., if an edge $\overset{\rightarrow}{gh}$ does not exist, then an LU-edge $\overset{LU \rightarrow}{gh}$ is added to the data-DAG, or if an L- or a U-edge $\overset{\rightarrow}{gh}$ exists, then it is converted to an LU-edge. Then, all edges gk such that $k > h$ are deleted. If the original matrix is not reducible to a block-triangular form, then after this modification, each supernode other than the root supernode has an LU-parent [21] to accommodate row-column pairs that fail to satisfy the pivoting criterion.

Fig. 4.2 shows how the failure of pivot row and column 1 is attempted in the unsymmetric multifrontal factorization of a small 5×5 example matrix. Row-column 1 is symmetrically permuted to a new location adjacent to 1's LU-parent 4 in the data-DAG. This results in an addition to row index 1 to 1's U-parent 2 and an addition of column index 1 to 1's L-parent 3. Additionally, after moving to their new location, row 1 in U and column 1 in L get fill-in in column and row positions where row 4 in U and column 4 in L have nonzeros; i.e., $U_{1,5}$, $L_{4,1}$, and $L_{5,1}$. Fig. 4.2 also shows that after pivoting, the new row 1 of C^2 cannot be fully assembled in the absence of an L-edge $\overset{L \rightarrow}{24}$.

Clearly, apart from adding LU-edges as described above, $Ddag^N$ requires further modifications in order to serve as a data-DAG for unsymmetric multifrontal algorithm with dynamic pivoting. First, we present a modified version of Theorem 4.1 for supplementing the edge-set of $Tdag^C$ to yield $Ddag^N$.

THEOREM 4.4. *If (1) $j \in \text{Struct}(U_{i,*})$, (2) the LU-parent of i , if exists, is greater than j , (3) none of i 's U-parents are in $\text{Struct}(U_{*,j})$, and (4) there exists a k such that there is a U-path $\overset{U \rightsquigarrow}{ki}$ in $Tdag^C$ and LU-parent(k) $>$ j , then a U-edge $\overset{U \rightarrow}{ij}$ is necessary for C^i to be completely assembled into its parents' frontal matrices in the*

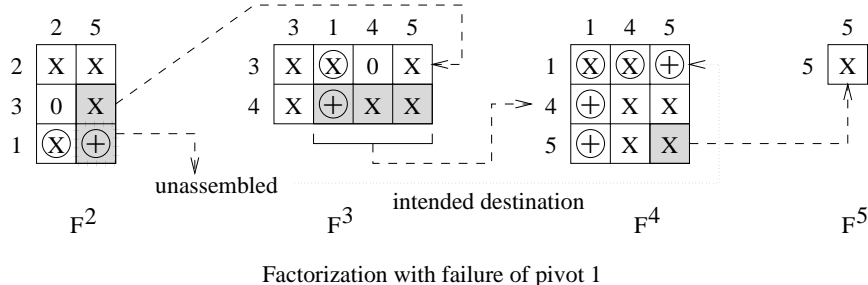
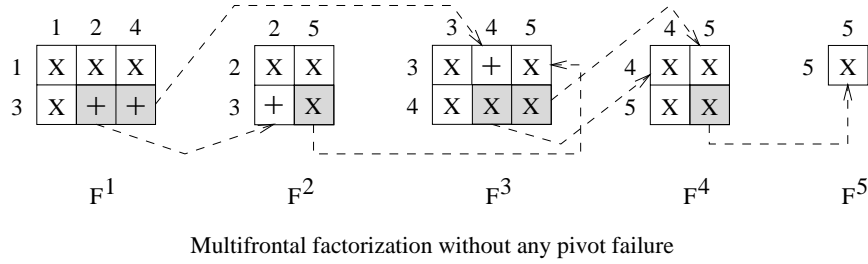
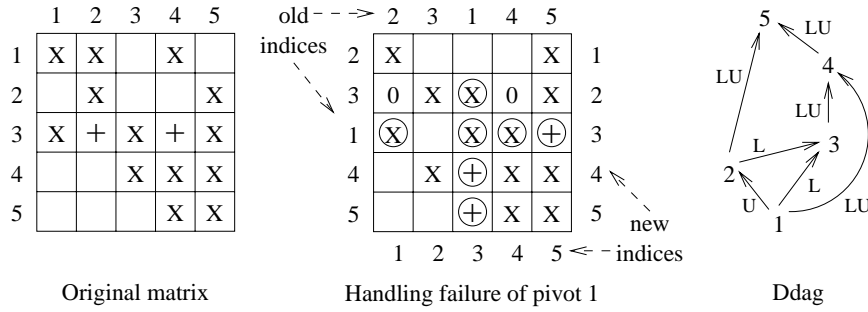


FIG. 4.2. An example factorization to show how the failure of pivot 1 is handled by a symmetric permutation of row and column 1 to merge them with their LU-parent supernode, 4. An 'X' denotes a nonzero in the coefficient matrix and a '+' denotes a fill-in. The circled 'X' and '+' are created due to pivoting. A '0' denotes a fill-in predicted by the original symbolic factorization that has a value of zero due to pivoting. The figure also shows that the absence of $2 \xrightarrow{L} 4$ leaves the entry $U_{1,5}$ unassembled from F^2 .

event of failure of pivot k . The transpose of this theorem can be stated similarly.

Proof. Note that Theorem 4.4 is very similar to Theorem 4.1. The only difference is condition (4). If pivot k fails, then it will add a row in $\text{Struct}(L_{*,i})$ that corresponds to $\text{LU-parent}(k) - 1$, which is the new location of k and is greater than $j - 1$, the new index for j . Thus, the failure of pivot k transforms condition (4) of Theorem 4.4 into condition (4) for the applicability of Theorem 4.1, which has already been proved. \square

Theorem 4.4 states that even if $\text{Struct}(L_{*,i})$ does not have any index greater than j but all other conditions for the applicability of Theorem 4.1 are satisfied and $i \xrightarrow{U} j$ is not present in the DAG, then pivoting may result in incomplete assembly unless this edge is added. In the light of Theorem 4.4, before proceeding further, we redefine

$Ddag^N$. First, instead of using Theorem 4.1 strictly to derive $Ddag^N$ from $Tdag^C$, we omit checking for condition (4); i.e., $\exists k \in \text{Struct}(L_{*,i})$ such that $k > j$. Then, as describe earlier, we add outgoing LU-edges to all nodes and remove the edges rendered redundant as a result. In the remainder of this section, $Ddag^N$ will denote the data-DAG with these two modifications over the data-DAG defined in §4.4.

Now, by means of Theorem 4.5, we will show that the data-DAG, even after the modifications described above, is not sufficient to ensure complete assembly of all contribution matrices in the event of inter-supernode pivoting. Theorem 4.5 alludes to the exact same condition that is illustrated in Fig. 4.2. Finally, Theorem 4.6 will show that supplementing the data-DAG with additional edges prescribed by Theorem 4.5 makes it sufficient to handle all contribution matrices in the face of inter-supernode pivoting. As we did earlier in this paper, for the sake of clarity and simplicity, we will state and prove Theorems 4.5 and 4.6 in the context of conventional DAGs with single-node supernodes. The results naturally extend to supernodal DAGs.

THEOREM 4.5. *If h is the LU-parent of j and (1) there exists an L-path $\overset{L \rightsquigarrow}{ji}$ (a U-path $\overset{U \rightsquigarrow}{ji}$) such that $i < h$ and LU-parent(i) $> h$, (2) none of i 's U-parents (L-parents) are in $\text{Struct}(L_{*,j})$ ($\text{Struct}(U_{j,*})$), and (3) either $\exists k \in \text{Struct}(L_{*,i})$ ($\text{Struct}(U_{i,*})$) such that $k > h$, or there is a U-path $\overset{U \rightsquigarrow}{ki}$ (an L-path $\overset{L \rightsquigarrow}{ki}$) and LU-parent(k) $> h$, then a U-edge $\overset{U \rightarrow}{ij}$ (an L-edge $\overset{L \rightarrow}{ih}$) is necessary for C^i to be completely assembled into its parents' frontal matrices in the event that j fails to meet the pivot criterion in its original location.*

Proof. If pivot j fails, then, along with other failed LU-children of h , it occupies a new position just before h . Since there is an L-path $\overset{L \rightsquigarrow}{ji}$, column j is added to C^i after the failure of pivot j ; i.e., in the new matrix after pivoting, $j \in \text{Struct}(U_{i,*})$. We know that the LU-parent of i is greater than the new j , because LU-parent(i) $> h$. Since none of i 's U-parents were in the old $\text{Struct}(L_{*,j})$, they are not in the new $\text{Struct}(U_{*,j})$ either. Thus the first three conditions for the applicability of Theorems 4.1 and 4.4 are satisfied. Condition (3) of Theorem 4.5 is equivalent to condition (4) of Theorems 4.1 or 4.4. Therefore, a U-edge $\overset{U \rightarrow}{ij}$ is needed for proper multifrontal factorization of the new matrix after permuting j to its new location. Since, in its new location, j is merged with h into a common supernode, a U-edge $\overset{U \rightarrow}{ih}$ in the original matrix would have sufficed. The transpose case can be proven similarly. \square

THEOREM 4.6. *If $Ddag^P$ is a DAG formed by adding all possible edges according to Theorem 4.5 to $Ddag^N$, then $Ddag^P$ is an adequate data-DAG for unsymmetric multifrontal factorization with potentially unlimited inter-supernode pivoting.*

Proof. We prove this by showing that with $Ddag^P$, it is not possible for any element of a contribution matrix C^i to remain unassembled. Without loss of generality, consider an element corresponding to $L_{k,j}$ in C^i . If $L_{k,j}$ is in C^i , then either $k \in \text{Struct}(L_{*,i})$ and $j \in \text{Struct}(U_{i,*})$ in the original L and U predicted by symbolic factorization, or row k or column j or both were added to C^i due to pivoting. If row k and column j are parts of the original structure of C^i , then Theorem 4.2 has already shown that the edge-set of $Ddag^N$, which is a subset of the edge-set of $Ddag^P$, is sufficient to assemble $L_{k,j}$. We now show that $L_{k,j}$ will be absorbed from C^i by one of i 's parents in $Ddag^P$ when column j was added to C^i due to pivoting, irrespective of whether row k belonged to the original $\text{Struct}(L_{*,i})$ or if it too was added to C^i due to pivoting.

Let $g = \text{LU-parent}(i)$ and $h = \text{LU-parent}(j)$. We consider two cases: (1) $g \leq h$ and (2) $g > h$. If $g \leq h$, F^g will have both row k and column j and will absorb the element corresponding to $L_{k,j}$ from C^i . If $g > h$, then the first condition for the applicability of Theorem 4.5 has been satisfied. Now we consider two further scenarios: (2a) at least one of i 's U-parents is in the original $\text{Struct}(L_{*,j})$ or (2b) none of i 's U-parents is in the original $\text{Struct}(L_{*,j})$. In case of (2a), after pivoting, at least one of i 's U-parents is in the new $\text{Struct}(U_{*,j})$ and the frontal matrix of this U-parent will absorb column j from C^i , including the entry corresponding to $L_{k,j}$. In case of (2b), the second condition for the applicability of Theorem 4.5 has also been satisfied. Finally, whether row k was in the original $\text{Struct}(L_{*,i})$ or was added to C^i due to the failure of a U-descendent k , in its final location, k must be greater than h . This is because if $j \leq k \leq h$ (i.e., k 's new location is in the extended supernode h), then h will have to be an LU-ancestor of i because $j \leq k \leq h$ implies that there are both ih and ih in the data-DAG. But that is not possible because we are already working under the assumption that the LU-parent g of i is greater than h . Therefore, $k > h$ and the third condition of Theorem 4.5 has also been satisfied and Theorem 4.5 would have ensured that a U-edge ih is present in $Ddag^P$ to assemble column j from C^i into F^h .

Similarly, we can prove that no entry corresponding any $U_{j,k}$ will be left unassembled in C^i . \square

4.6. Experimental results. In §4.4 and §4.5, we showed how to supplement the edge-set of the task-DAG to construct a data-DAG for the unsymmetric multifrontal algorithm. Table 4.1 shows experimental results of WSMP's implementation of the procedures to generate the various DAGs. Three DAGs are considered in Table 4.1: the supernodal task-DAG $Tdag^S$, the supernodal data-DAG $Ddag^N$ for unsymmetric multifrontal factorization without pivoting, and the supernodal data-DAG $Ddag^P$ for unsymmetric multifrontal factorization with pivoting. The table shows the time to compute each of the DAGs and the number of edges in them for the 25 matrices in our test suite.

$Tdag^S$ is computed by the basic symbolic factorization described in §3; therefore, T^S is the basic symbolic factorization time. We refer to the process of computing $Ddag^N$ from $Tdag^S$ as *Supplement-1*. Supplement-1 checks for the first three conditions of Theorem 4.1 or 4.4 to find the edges to be added to E_{Tdag^S} and then adds outgoing LU-edges from supernodes without LU-parents to yield E_{Ddag^N} . *Supplement-2* is the process that adds edges based on the first two conditions of Theorem 4.5 to E_{Ddag^N} to yield E_{Ddag^P} . T^1 and T^2 refer to the execution time of Supplement-1 and Supplement-2, respectively. Recall that all the edges in $Ddag^N$ and $Ddag^P$ are not necessary. Furthermore, for the sake of computational speed, Supplement-1 and Supplement-2 do not check for all the conditions of Theorems 4.1 and 4.5 while adding edges. Condition (4) of Theorems 4.1 and 4.4 and condition (3) of Theorem 4.5 are skipped. Therefore, the resulting $Ddag^N$ and $Ddag^P$ are not minimal DAGs. However, as Table 4.1 shows, these DAGs do not have many more edges than $Tdag^S$ for most real-life matrices. The average excess edges in $Ddag^P$ over $Tdag^S$ is only about 4% for our test suite. The time required to construct $Ddag^N$ and $Ddag^P$ is also fairly small compared to the basic symbolic factorization time. Thus, the methodology described in this section for the construction of data-DAGs for unsymmetric multifrontal factorization is efficient in both time and the number of DAG edges. A comparison of the $T^S + T^1 + T^2$ column of Table 4.1 with the default WSMP column

| Matrix | Symbolic | | Supplement-1 | | Supplement-2 | | Total Time | $\frac{ E_{Ddag^P} }{ E_{Tdag^S} }$ |
|----------|----------|----------------|--------------|----------------|--------------|----------------|-------------------|-------------------------------------|
| | T^S | $ E_{Tdag^S} $ | T^1 | $ E_{Ddag^N} $ | T^2 | $ E_{Ddag^P} $ | $T^S + T^1 + T^2$ | |
| af23560 | .47 | 4793 | .03 | 4794 | .00 | 4794 | 0.50 | 1.00 |
| av41092 | .83 | 34708 | .47 | 36346 | .02 | 37092 | 1.32 | 1.07 |
| bayer01 | .28 | 87028 | .13 | 95285 | .05 | 96818 | 0.46 | 1.11 |
| bbmat | 1.7 | 6077 | .06 | 6142 | .00 | 6181 | 1.76 | 1.02 |
| comp2c | .22 | 1736 | .03 | 1929 | .00 | 1978 | 0.25 | 1.14 |
| e40r0000 | .14 | 3225 | .01 | 3264 | .00 | 3264 | 0.15 | 1.01 |
| e40r5000 | .16 | 3182 | .01 | 3235 | .00 | 3237 | 0.17 | 1.02 |
| ec132 | 1.2 | 15239 | .07 | 15244 | .00 | 15260 | 1.27 | 1.00 |
| epb3 | .50 | 38088 | .08 | 38173 | .01 | 38300 | 0.59 | 1.01 |
| fidap011 | .42 | 1261 | .02 | 1261 | .00 | 1261 | 0.44 | 1.00 |
| fidapm11 | .65 | 2651 | .03 | 2654 | .00 | 2654 | 0.68 | 1.00 |
| invextr1 | .93 | 10108 | .11 | 10813 | .01 | 11244 | 1.05 | 1.11 |
| mil053 | 4.5 | 166154 | .51 | 166154 | .07 | 166154 | 5.08 | 1.00 |
| mixtank | 1.2 | 3203 | .05 | 3203 | .00 | 3203 | 1.25 | 1.00 |
| nasasrb | .97 | 3807 | .05 | 3807 | .00 | 3807 | 1.02 | 1.00 |
| onetone1 | .31 | 23585 | .05 | 24523 | .01 | 24691 | 0.37 | 1.05 |
| onetone2 | .18 | 23999 | .04 | 24818 | .01 | 24928 | 0.23 | 1.04 |
| pre2 | 6.4 | 317216 | .84 | 320063 | .16 | 320942 | 7.40 | 1.01 |
| raefsky3 | .41 | 1281 | .02 | 1281 | .00 | 1281 | 0.43 | 1.00 |
| raefsky4 | .50 | 1358 | .02 | 1358 | .00 | 1358 | 0.52 | 1.00 |
| rma10 | .56 | 3911 | .03 | 3911 | .00 | 3911 | 0.59 | 1.00 |
| tib | .07 | 10060 | .01 | 10517 | .00 | 10655 | 0.08 | 1.06 |
| twotone | .91 | 44856 | .12 | 45866 | .01 | 45918 | 1.04 | 1.02 |
| wang3 | .54 | 8450 | .03 | 8450 | .00 | 8450 | 0.57 | 1.00 |
| wang4 | .53 | 8253 | .03 | 8253 | .00 | 8253 | 0.56 | 1.00 |

TABLE 4.1

Time required for constructing $Tdag^S$, $Ddag^N$, and $Ddag^P$ and the number of edges in each DAG.

of Table 5.1 shows that the total symbolic time is usually significantly less than the numerical factorization time.

5. Implementation details of unsymmetric factorization. A brief outline of the unsymmetric multifrontal algorithm based on Hadfield's [21] and Davis and Duff's [8] work is contained in §4.2. We now add some details to it and present a complete algorithm that is implemented in WSMP. WSMP is geared towards multiple factorizations of matrices with the same sparsity pattern but different nonzero values. The symbolic phase is performed only once.

A fundamental data-structure in our unsymmetric multifrontal algorithm is the frontal matrix. A frontal matrix is associated with each supernode. Fig. 5.1 shows the organization of a typical frontal matrix for a supernode $g = ([q:r])$. The core of frontal matrix is a $|\text{Struct}(L_{*,q})| \times |\text{Struct}(U_{q,*})|$ portion, where $\text{Struct}(L_{*,q})$ and $\text{Struct}(U_{q,*})$ are predicted by the symbolic factorization. In the absence of pivoting, the first $r - q + 1$ rows and columns of this matrix would be factored and would be saved as parts of U and L , respectively. The remaining trailing submatrix would constitute the contribution matrix whose contents would be absorbed into the frontal matrices of the parents of g in $Ddag^P$.

In the presence of pivoting, extra pivots as well as other rows and columns may be added to the frontal matrix depending on the labels and pivot failures of the children

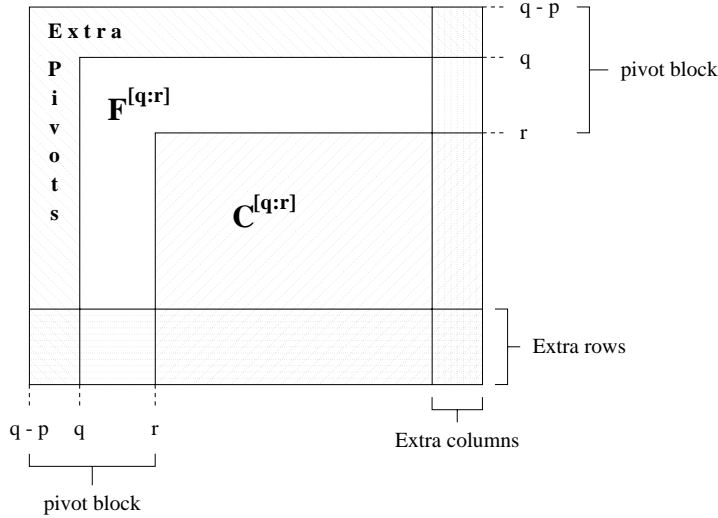


FIG. 5.1. Organization of a typical frontal matrix for a supernode $g = \mathcal{S}([q:r])$. The p failed pivots from the LU-children of the supernode are appended at the beginning of the frontal matrix and the extra rows and columns inherited from U- and L-descendents, respectively, are appended at the end.

of g in $Ddag^P$. Extra pivots (rows-column pairs with the same indices) are added to F^g if some of the pivots of g 's LU-children fail to satisfy the pivoting criterion. The LU-children themselves may have inherited some or all these failed pivots from one of their own LU-children. Therefore, failed pivots from any of the LU-descendents of g can end up in its frontal matrix. If p such pivots are added, then the size of the pivot block increases from $r - q + 1$ to $r - q + p + 1$.

The frontal matrix F^g can similarly inherit extra rows corresponding to failed pivots in its U-descendents whose LU-parents are greater than g and extra columns corresponding to failed pivots in its L-descendents whose LU-parents are greater than g . Irrespective of their new indices, these extra rows and columns are always appended at the end of the original rows and columns of F^g and a sorted list of their indices is maintained at each supernode. Eventually, these are assembled into the extra pivots of the frontal matrices of the LU-parents of the supernodes where these pivots failed. The row and column structures predicted by symbolic factorization are kept intact for future factorizations of matrices with the same nonzero pattern. The additions to these structures due to pivoting, which depend on the nonzero values in the matrix being factored, are maintained separately and are discarded before each new factorization.

The availability of a static data-DAG $Ddag^P$ that is sufficient for handling an arbitrary amount of dynamic pivoting is critical to our implementation of the unsymmetric multifrontal algorithm. Fig. 5.2 gives a high-level pseudocode of our factorization algorithm. The algorithm starts with the root supernode of task- and data-DAGs. At any supernode, first, it recursively factors all the unfactored children of that supernode. Then it looks at the failed pivots (if any) of its children to figure out the number and indices of the extra rows, columns, and pivots, if any, and accordingly allocates a frontal matrix of the appropriate size. In the next step, the contribution from the original coefficient matrix and the contribution matrices of the current supernode's children is accumulated in the appropriate locations inside the frontal matrix. Finally,

```

function uns_mf (root) {
  /* 1. Recursive calls to root's children */
  for each child k of root in TdagS do
    if not already processed k then
      Call uns_mf (k);
    end for
  /* 2. Collect pivoting info to determine size of  $F^{root}$  */
  for each child k of root in DdagP do
    if k is an L-child then
      if k has failed pivots then
        Add them to the sorted list of  $F^{root}$ 's extra columns;
      if  $C^k$  has extra columns then
        Add those whose LU-parent is greater than root to the
        sorted list of  $F^{root}$ 's extra columns while checking for duplicates;
      else if k is a U-child then
        if k has failed pivots then
          Add them to the sorted list of  $F^{root}$ 's extra rows;
        if  $C^k$  has extra rows then
          Add those whose LU-parent is greater than root to the
          sorted list of  $F^{root}$ 's extra rows while checking for duplicates;
      else if k is an LU-child then
        if k has failed pivots then
          Add them to the sorted list of  $F^{root}$ 's extra pivots;
        if  $C^k$  has extra columns then
          Add those whose LU-parent is greater than root to the
          sorted list of  $F^{root}$ 's extra columns while checking for duplicates;
        if  $C^k$  has extra rows then
          Add those whose LU-parent is greater than root to the
          sorted list of  $F^{root}$ 's extra rows while checking for duplicates;
      end if
    end for
  /* 3. Initialize root's frontal matrix */
  Allocate  $F^{root}$  of appropriate size and fill it with zeros;
  Populate  $F^{root}$  with entries from A corresponding to supernode root;
  /* 4. Assembly from children's contribution matrices into  $F^{root}$  */
  for each child k of root in DdagP do
    Copy appropriate contribution from  $C^k$  into  $F^{root}$ ;
    if root is the last parent of k to pick up  $C^k$ 's contribution then
      Free the space occupied by  $C^k$ ;
    end for
  /* 5. Numerical factorization */
  Factor the pivot block of  $F^{root}$  and compute  $C^{root}$ ;
end function uns_mf.

```

FIG. 5.2. A simple and efficient unsymmetric multifrontal algorithm.

the algorithm proceeds to factor the pivot block of the frontal matrix and updates the remainder of the frontal matrix. The leading successfully factored rows and columns are saved as portions of U and L for use during triangular solves. The remaining contribution matrix is eventually assembled into the frontal matrices of its parents and is released by the last parent to pick up its contribution.

The frontal matrix of the LU-parent of a supernode picks up all its failed pivot row-column pairs as well as the entire trailing submatrix of its contribution matrix with row and column indices greater than or equal to the first index of the parent supernode. The remaining rows and columns of a supernode's contribution matrix are assembled into the frontal matrices of its L- and U-parents in $Ddag^P$. It is possible for more than one L- or U-parents' frontal matrices to have the same row or column indices in common with the child's contribution matrix. However, each element of a contribution matrix must be added into exactly one frontal matrix. Some simple bookkeeping to keep track of rows and columns that have been assembled suffices to ensure this condition for the relatively few rows and columns that have the potential to be copied into the frontal matrices of multiple L- and U-parents, respectively.

Fig. 5.2 and the description in this section shows that WSMP's unsymmetric multifrontal algorithm is fairly straightforward to implement. The apparent complexity of the unsymmetric multifrontal algorithms described in [21] and [8] was probably a deterrent to the development of commercial software packages based on this algorithm, despite its theoretical advantage over symmetric pattern multifrontal codes such as MUMPS [2, 3] for unsymmetric matrices. The algorithm of Fig. 5.2 is not only relatively simple in description, but is also computationally lean because it can handle pivot failures very efficiently. A powerful and completely separate symbolic phase that generates the data-DAG $Ddag^P$ is the key to the simplicity and efficiency of our multifrontal LU factorization algorithm.

5.1. Experimental results. We now compare the unsymmetric LU factorization time of WSMP with that of two state-of-the-art multifrontal sparse direct solvers, namely, MUMPS version 4.1.6 [2, 3] and UMFPACK version 3.0 [6]. A detailed comparative study that includes more solvers can be found in [16, 20]. The softwares compared in this section employ different variants of the multifrontal algorithm. MUMPS contains a symmetric-pattern multifrontal factorization code based on the classical multifrontal algorithm [13]. UMFPACK contains an unsymmetric-pattern multifrontal code [8, 21]. Apart from the factorization algorithm, there are other differences among the three softwares that affect their performance. First, they use different schemes for fill-reducing ordering. By default, WSMP uses a symmetric permutation based on a nested-dissection ordering [17] computed on the structure of $A + A'$. MUMPS uses a symmetric permutation based on the approximate minimum degree (AMD) algorithm [7] applied to the structure of $A + A'$. UMFPACK uses a column approximate minimum degree algorithm [9] to prepermute only the columns of A and computes a row permutation based on numerical and sparsity criteria during factorization. The second difference is the use of a maximal matching algorithm [12] to permute the rows of the coefficient matrix to maximize the product of the magnitudes of its diagonal entries. As shown in [4, 16], this can affect factorization times because it changes the amount of structural symmetry and the amount of numerical pivoting during factorization. WSMP uses this preprocessing on all matrices, MUMPS uses it only if the structural symmetry in the original matrix is less than 50%, and UMFPACK does not use it at all. The third difference is that WSMP and UMFPACK reduce the coefficient matrix into a block triangular form, while MUMPS does not.

| Matrix | MUMPS | | UMFPACK 3 | | WSMP (default) | | WSMP (AMD) | |
|----------|----------------|----------------------|----------------|----------------------|----------------|----------------------|----------------|----------------------|
| | time (sec.) | ops $\times 10^9$ | time (sec.) | ops $\times 10^9$ | time (sec.) | ops $\times 10^9$ | time (sec.) | ops $\times 10^9$ |
| af23560 | <u>4.05</u> | 2.56 | 9.19 | 3.49 | 3.37 | 2.66 | 4.31 | 3.48 |
| av41092 | 12.0 | 8.42 | 124. | 34.1 | 4.12 | 1.92 | <u>7.05</u> | 3.95 |
| bayer01 | 1.10 | .125 | 1.14 | .023 | <u>0.94</u> | .040 | 0.92 | .023 |
| bbmat | 48.0 | 41.4 | 86.8 | 37.8 | 23.0 | 20.2 | <u>23.4</u> | 20.8 |
| comp2c | 13.7 | 4.22 | 556. | 107. | 1.70 | .677 | <u>2.63</u> | .780 |
| e40r0000 | 0.83 | .172 | 6.18 | 2.14 | <u>0.53</u> | .219 | 0.48 | .194 |
| e40r5000 | <u>0.85</u> | .172 | 6.74 | 2.12 | 0.80 | .362 | 1.08 | .548 |
| ec132 | 64.7 | 64.6 | 190. | 111. | 23.0 | 20.5 | <u>49.6</u> | 44.2 |
| epb3 | 2.70 | 1.17 | 5.33 | 1.14 | 1.63 | .431 | <u>1.72</u> | .547 |
| fidap011 | 8.73 | 7.01 | 17.8 | 8.20 | 3.62 | 2.87 | <u>6.54</u> | 5.74 |
| fidapm11 | <u>11.6</u> | 9.67 | 41.1 | 19.9 | 6.93 | 5.68 | 16.0 | 14.7 |
| invextr1 | 38.9 | 35.6 | 170. | 85.3 | 10.2 | 7.54 | <u>33.3</u> | 22.4 |
| mil053 | 42.8 | 31.8 | 107. | 45.8 | 24.6 | 16.0 | <u>29.7</u> | 22.2 |
| mixtank | 64.8 | 64.4 | 390. | 238. | 21.5 | 19.1 | <u>45.6</u> | 44.2 |
| nasasrb | 13.1 | 9.45 | 58.0 | 28.8 | 7.12 | 5.58 | <u>10.4</u> | 8.78 |
| onetone1 | 3.66 | 2.29 | 5.16 | 2.09 | 2.08 | 1.10 | <u>2.50</u> | 1.58 |
| onetone2 | 1.17 | .510 | <u>0.77</u> | .076 | 0.75 | .212 | 0.82 | .346 |
| pre2 | fail | fail | fail | fail | 155. | 122. | <u>360.</u> | 327. |
| raefsky3 | <u>4.56</u> | 2.90 | 16.3 | 7.86 | 2.99 | 2.36 | 4.86 | 4.17 |
| raefsky4 | 13.0 | 10.9 | 27.0 | 12.9 | 4.78 | 4.02 | <u>8.33</u> | 7.51 |
| rma10 | 4.13 | 1.39 | 9.19 | 3.49 | 2.39 | 1.41 | <u>2.59</u> | 1.57 |
| tib | 0.40 | .044 | 28.6 | .229 | <u>0.25</u> | .031 | 0.23 | .031 |
| twotone | 43.5 | 29.3 | 32.0 | 10.8 | <u>12.2</u> | 7.69 | 3.42 | 1.97 |
| wang3 | 15.1 | 13.8 | 50.0 | 28.1 | 7.16 | 6.46 | <u>11.4</u> | 10.6 |
| wang4 | 11.8 | 10.5 | 53.7 | 30.6 | 8.09 | 7.15 | <u>8.86</u> | 7.94 |

TABLE 5.1

LU Factorization times and operation counts of MUMPS, UMFPACK 3, and WSMP. The best time is in boldface and the second best time is underlined.

Table 5.1 shows factorization times and operation counts of MUMPS, UMFPACK, and WSMP. To mitigate the differences due to ordering, we also include WSMP factorization statistics with AMD ordering, which is used in MUMPS (symmetrically on rows and columns) and UMFPACK (on columns only). The fastest factorization time for each matrix is in boldface and the second fastest time is underlined. Although differences other than the factorization algorithm itself affect the performance of these codes, it is easy to see the broad picture that emerges from Table 5.1. Most of the boldface entries are in the column for WSMP with default ordering and the remaining ones are in the WSMP column with AMD ordering. All but five underlined entries are also in one of the WSMP columns. For many matrices, the affect of the algorithmic choices of the softwares is evident in the factorization statistics. MUMPS usually requires more floating-point operations for factorization than WSMP with AMD ordering because it uses artificially symmetrized frontal matrices padded with zeros. For the same reason, UMFPACK beats MUMPS for very unsymmetric matrices such as *onetone2* and *twotone*; however, it is a lot slower for matrices with more structural symmetry. WSMP, with its default nested-dissection ordering, is usually the fastest code because nested-dissection is more effective in reducing fill-in than AMD. WSMP with AMD ordering is usually the second fastest code in Table 5.1, primarily because of its efficient unsymmetric multifrontal algorithm. For some matrices such

as *fidapm11* and *raefsky3* with highly symmetric structures, its performance is hurt by the row-prepermutation to maximize the product of the magnitude of the diagonal entries, which tends to destroy structural symmetry and increase the operation count.

6. Concluding remarks. This paper describes sparse unsymmetric symbolic and numerical factorization algorithms that are both simpler and faster than previous similar algorithms. Our symbolic factorization phase, in particular, is more powerful than others described in the literature. It inexpensively computes minimal elimination structures that are transitive reductions of the upper- and lower-triangular factors of the original coefficient matrix. In addition, it computes near-minimal data-dependency DAGs for unsymmetric multifrontal factorization with and without pivoting. A data-DAG that has only a slightly higher number of edges than a minimal task-DAG and that is capable of expressing all possible data-dependencies in the face of dynamic pivoting is a key feature of our symbolic phase. We show how this data-DAG aids a simple but very high-performance implementation of unsymmetric multifrontal LU factorization algorithm. The static nature of this data-DAG would also be a boon for potential parallel implementations of unsymmetric multifrontal factorization where changing the data-DAG dynamically could be cumbersome and inefficient.

REFERENCES

- [1] Patrick R. Amestoy and Iain S. Duff. Vectorization of a multiprocessor multifrontal code. *International Journal of Supercomputer Applications*, 3:41–59, 1989.
- [2] Patrick R. Amestoy, Iain S. Duff, Jacko Koster, and J. Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.
- [3] Patrick R. Amestoy, Iain S. Duff, and J. Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computational Methods in Applied Mechanical Engineering*, 184:501–520, 2000.
- [4] Patrick R. Amestoy, Iain S. Duff, J. Y. L'Excellent, and Xiaoye S. Li. Analysis, tuning, and comparison of two general sparse solvers for distributed memory computers. Technical Report RT/APO/00/2, ENSEEIHT-IRIT, Toulouse, France, 2000. Also available as Technical Report 45992 from Lawrence Berkeley National Laboratory.
- [5] Cleve Ashcraft and Roger G. Grimes. The influence of relaxed supernode partitions on the multifrontal method. *ACM Transactions on Mathematical Software*, 15(4):291–309, 1989.
- [6] Timothy A. Davis. UMFPACK software for unsymmetric multifrontal method. *NA Digest*, 01(11), March 18, 2001. <http://www.cise.ufl.edu/research/sparse/umfpack>.
- [7] Timothy A. Davis, Patrick R. Amestoy, and Iain S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.
- [8] Timothy A. Davis and Iain S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM Journal on Matrix Analysis and Applications*, 18(1):140–158, January 1997.
- [9] Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, and Esmond G.-Y. Ng. A column approximate minimum degree ordering algorithm. Technical Report TR-00-005, Computer and Information Sciences Department, University of Florida, Gainesville, FL, 2000.
- [10] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W.-H. Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3):720–755, 1999.
- [11] Iain S. Duff, A. M. Erisman, and John K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, Oxford, UK, 1990.
- [12] Iain S. Duff and Jacko Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. Technical Report RAL-TR-1999-030, Rutherford Appleton Laboratory, April 19, 1999.
- [13] Iain S. Duff and John K. Reid. The multifrontal solution of unsymmetric sets of linear equations. *SIAM Journal on Scientific and Statistical Computing*, 5(3):633–641, 1984.

- [14] Stanley C. Eisenstat and Joseph W.-H. Liu. Exploiting structural symmetry in unsymmetric sparse symbolic factorization. *SIAM Journal on Matrix Analysis and Applications*, 13(1):202–211, 1992.
- [15] John R. Gilbert and Joseph W.-H. Liu. Elimination structures for unsymmetric sparse LU factors. *SIAM Journal on Matrix Analysis and Applications*, 14(2):334–352, 1993.
- [16] Anshul Gupta. Recent advances in direct methods for solving unsymmetric sparse systems of linear equations. Technical Report RC 22039 (98933), IBM T. J. Watson Research Center, Yorktown Heights, NY, April 20, 2001. (Available at <ftp.cs.umn.edu/users/kumar/anshul/solver-compare.ps>).
- [17] Anshul Gupta. Fast and effective algorithms for graph partitioning and sparse matrix ordering. *IBM Journal of Research and Development*, 41(1/2):171–183, January/March, 1997.
- [18] Anshul Gupta. WSMP: Watson sparse matrix package (Part-II: direct solution of general sparse systems). Technical Report RC 21888 (98472), IBM T. J. Watson Research Center, Yorktown Heights, NY, November 20, 2000. <http://www.cs.umn.edu/~agupta/wsmg.html>.
- [19] Anshul Gupta, George Karypis, and Vipin Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):502–520, May 1997.
- [20] Anshul Gupta and Yanto Muliadi. An experimental comparison of some direct sparse solver packages. In *Proceedings of International Parallel and Distributed Processing Symposium*, 2001.
- [21] Steven M. Hadfield. *On the LU Factorization of Sequences of Identically Structured Sparse Matrices within a Distributed Memory Environment*. PhD thesis, University of Florida, Gainesville, FL, 1994.
- [22] Joseph W.-H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11:134–172, 1990.
- [23] Joseph W.-H. Liu. The multifrontal method for sparse matrix solution: Theory and practice. *SIAM Review*, 34:82–109, 1992.