A column pre-ordering strategy for the unsymmetric-pattern multifrontal method *

Timothy A. Davis †

January 2, 2002

Technical report TR-02-001. Department of Computer and Information Science and Engineering, University of Florida.

Abstract

A new method for sparse LU factorization is presented that combines a left-looking column pre-ordering strategy with a right-looking unsymmetric-pattern multifrontal numerical factorization. The column ordering is selected to give a good a priori upper bound on fill-in and then refined during numerical factorization (while preserving the bound). Pivot rows are selected to maintain numerical stability and to preserve sparsity. Left-looking methods cannot select pivot rows to preserve sparsity. As a result, the new method nearly always obtains better orderings than both left-looking methods (such as that used in MATLAB), and the prior unsymmetric-pattern multifrontal method on which it is based (MA38, or UMFPACK Version 2).

Categories and Subject Descriptors: G.1.3 [Numerical Analysis]: Numerical Linear Algebra – *linear systems (direct methods), sparse and very large systems* G.4 [Mathematics of Computing]: Mathematical Software – *algorithm analysis, efficiency*

General terms: Algorithms, Experimentation, Performance.

Keywords: sparse nonsymmetric matrices, linear equations, multifrontal method, ordering methods.

*This work was supported by the National Science Foundation, under grants DMS-9504974 and DMS-9803599.

†Dept. of Computer and Information Science and Engineering, Univ. of Florida, Gainesville, FL, USA. email: davis@cise.ufl.edu. http://www.cise.ufl.edu/∼davis.

1 Introduction

This paper considers the direct solution of systems of linear equations, $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is sparse and unsymmetric. The matrix $\mathbf{PAQ}$ is factorized into the product $\mathbf{LU}$. The column ordering $\mathbf{Q}$ is selected to give a good a priori upper bound on fill-in and then refined during numerical factorization (while preserving the upper bound on fill-in). The row ordering $\mathbf{P}$ is selected to maintain numerical stability and to preserve sparsity. The method is a combination of a column pre-ordering and symbolic analysis phase based on COLAMD [16, 17, 39], and a numerical factorization phase based on a modification of the unsymmetric-pattern multifrontal method (UMFPACK Version 2, or MA38 [14, 15], which does not have a pre-ordering and symbolic analysis phase).

The pre-ordering and symbolic analysis of the method presented here is similar to that used by left-looking methods such as SuperLU [18] or MATLAB's LU [33, 34]. In these methods, the column pre-ordering $\mathbf{Q}$ is selected to provide a good upper bound on fill-in, no matter how the row ordering $\mathbf{P}$ is chosen during numerical factorization. However, left-looking methods cannot select the row ordering to preserve sparsity. MA38 can select both the row and column ordering to preserve sparsity, but it lacks an analysis phase that gives good a priori bounds on fill-in. It can thus experience unacceptable fill-in for some matrices. In contrast to both of these strategies, the numerical factorization in the method described here has the same a prior upper bound on fill-in as left-looking methods (something that MA38 lacks), and the new method can select the row ordering $\mathbf{P}$ based on sparsity preserving criteria (something that left-looking methods cannot do).

Section 2 gives an overview of the prior methods that the new method is based on or related to: column pre-orderings, a priori upper bounds on fill-in, left-looking methods, and right-looking multifrontal methods. Section 3 describes the new algorithm. Performance results and comparisons with other codes are given in Section 4, before a few concluding remarks and information on the availability of the code are given in Section 5.

2 Background

The new method is related to left-looking methods, since it uses a column pre-ordering that gives the same a priori bounds on fill-in. The numerical

factorization phase is based on the right-looking multifrontal method, guided by the supernodal column elimination tree. These related methods are described below.

2.1 Column pre-orderings

Fill-in is the introduction of new nonzero entries in $\mathbf{L}$ and $\mathbf{U}$ whose corresponding entries in $\mathbf{A}$ are zero. The row and column orderings, $\mathbf{P}$ and $\mathbf{Q}$, determine the amount of fill-in that occurs. Finding the best ordering is an NP-complete problem [45], and thus heuristics are used.

Suppose the column ordering $\mathbf{Q}$ is fixed, and let $\mathbf{C} = \mathbf{AQ}$. Sparse Gaussian elimination with partial pivoting selects $\mathbf{P}$ via standard partial pivoting with row interchanges, and factorizes $\mathbf{PC}$ into $\mathbf{LU}$. If $\mathbf{C}$ has a zero-free diagonal the nonzero pattern of $\mathbf{U}$ is a subset of the nonzero pattern of the Cholesky factor $\mathbf{L}_C$ of $\mathbf{C}^\mathsf{T}\mathbf{C}$ [30]. The entries in each column of $\mathbf{L}$ can be rearranged so that their nonzero pattern is a subset of the nonzero pattern of $\mathbf{L}_C$. This subset relationship holds no matter how $\mathbf{P}$ is chosen during Gaussian elimination on $\mathbf{C}$.

This observation leads to a useful method for finding an ordering $\mathbf{Q}$ that gives a good upper bound on the fill-in in the LU factors of $\mathbf{C} = \mathbf{AQ}$. Simply use for $\mathbf{Q}$ an ordering that reduces fill-in in the Cholesky factorization of $(\mathbf{AQ})^\mathsf{T}\mathbf{AQ}$ [28, 30, 31]. The COLMMD [33] and COLAMD [16, 17, 39] routines in MATLAB find an ordering $\mathbf{Q}$ without constructing the nonzero pattern of $\mathbf{A}^\mathsf{T}\mathbf{A}$.

2.2 Left-looking methods

Left-looking methods such as LU organize their computation with the column elimination tree (the elimination tree of $\mathbf{C}^\mathsf{T}\mathbf{C}$ [40]). SuperLU uses the supernodal column elimination tree to reduce execution time by exploiting dense matrix kernels (the BLAS [19]) in the computation of each supercolumn (a group of columns of $\mathbf{L}$ with the same upper bound on their nonzero pattern). MA48 in the Harwell Subroutine Library is another example of a left-looking method [26]. It differs from LU and SuperLU by using a partial right-looking numerical factorization as its pre-ordering strategy.

At the kth step of factorization of an n-by-n matrix $\mathbf{A}$, the kth column of $\mathbf{U}$ is computed. The pivot entry is chosen in the kth column, permuted to the diagonal, and the kth column of $\mathbf{L}$ is computed. Columns $k+1$ to n of

A are neither accessed nor modified in the kth step. The advantage of this approach is that it can be implemented in time proportional to the number of floating-point operations [34]. This is not known to be true of right-looking methods such as the multifrontal method. However, the disadvantage is that the kth pivot row cannot be selected on the basis of sparsity, since the nonzero patterns of the candidate pivot rows are unknown. The pre-ordering **Q** is found by assuming that all candidate pivot rows at the kth step have the same upper bound nonzero pattern. The pivot row is selected solely on the basis of maintaining numerical accuracy. Only a right-looking method (one that modifies the columns $k+1$ through n at the kth step of factorization) has access to the true nonzero patterns of candidate pivot rows at the kth step of factorization.

2.3 Right-looking multifrontal methods

The multifrontal method is one example of a right-looking method. Once the kth pivot row and column are found, the elimination is performed and the outer-product is applied to the remaining $(n-k)$-by-$(n-k)$ submatrix that has yet to be factorized.

The factorization is performed in a sequence of *frontal matrices*. Each frontal matrix is a small dense submatrix that holds one or more pivot rows and their corresponding pivot columns. Consider the first frontal matrix. The original entries in the corresponding rows and columns of **A** are assembled into the frontal matrix. The corresponding eliminations are performed, and the contribution block (a Schur complement) is computed. This contribution block is placed on a stack for use in a later frontal matrix. The factorization of subsequent frontal matrices is the same, except that it is preceded by an assembly step in which contribution blocks (or portions of them) are assembled (added) into the current frontal matrix. After the assembly step, the current frontal matrix has a complete representation of a set of pivot rows and columns. In all multifrontal methods, more than one pivot row and column can be held in a frontal matrix. Computing the Schur complement can be done with dense matrix-matrix multiplication (DGEMM [19]), an operation that can obtain near-peak performance on high-performance computers.

Many approaches have been taken to apply the multifrontal method to different classes of matrices:

1. symmetric positive definite matrices [8, 41],

2. symmetric indefinite matrices (MA27) [23, 24],

3. unsymmetric matrices with actual or implied symmetric nonzero pattern (MA41) [3, 21, 25],

4. unsymmetric matrices where the unsymmetric nonzero pattern is partially preserved (MA41u) [5, 6],

5. unsymmetric matrices where the unsymmetric nonzero pattern is fully preserved (MA38) [14, 15],

6. and QR factorization of rectangular matrices [4, 42].

There are significant differences among these various approaches. For the first four approaches, the frontal matrices are related to one another by the elimination tree of $\mathbf{A}$, or the elimination tree of $\mathbf{A} + \mathbf{A}^\mathsf{T}$ if $\mathbf{A}$ is unsymmetric [40, 41]. The elimination tree has n nodes; each node corresponds to one pivot row and column. The parent of node k is node p, where p is the smallest row index of nonzero entries below the diagonal in the kth column of $\mathbf{L}$. A frontal matrix corresponds to a path in the elimination tree whose columns of $\mathbf{L}$ have similar or identical nonzero pattern; the tree with one node per frontal matrix is called the assembly tree [22] or the supernodal elimination tree. Each frontal matrix is designed so that it can fully accommodate the contribution blocks of each of its children in the assembly tree. Thus, the assembly step adds the contribution blocks of each child into the current frontal matrix. For symmetric positive definite matrices, all of the pivots originally assigned to a frontal matrix by the symbolic analysis phase are numerically factorized within that frontal matrix. For other classes of matrices, some pivots might not be eliminated, and the contribution block can be larger than predicted. The uneliminated pivot is delayed, and its elimination is attempted in the parent instead.

In the first three approaches, the frontal matrices are square. In a recent approach by Amestoy, Duff, and Puglisi [5, 6] (approach #4 in the list above), it was noted that rows and columns in the frontal matrix that contain only zero entries can be detected during numerical factorization and removed from the frontal matrix. The frontal matrix is rectangular, and the assembly tree is still used.

The first four approaches precede the numerical factorization with a symmetric reordering of $\mathbf{A}$ or $\mathbf{A} + \mathbf{A}^\mathsf{T}$, typically with a minimum degree [1, 29]

or nested-dissection ordering [9, 28, 37, 38] as part of a symbolic analysis phase.

The fifth approach, used in MA38, does not use a pre-ordering or symbolic analysis phase. Rectangular frontal matrices are constructed during numerical factorization, using an approximate Markowitz ordering. The first pivot within a frontal matrix defines the pivot row and column pattern and the size of the frontal matrix. Extra room is added to accommodate subsequent pivot rows and columns. Subsequent pivots are then sought that can be factorized using the same frontal matrix, allowing the use of dense matrix kernels. The frontal matrices are related to one another via a directed acyclic graph (DAG) rather than an elimination tree.

The last approach, multifrontal QR factorization [4, 42], is based on the column elimination tree of $\mathbf{A}$.

3 The algorithm

An overview of the new algorithm (UMFPACK Version 3.2) is given below, followed with details of its implementation. It uses the fifth approach in the list above (like MA38). Unlike MA38, it precedes the numerical factorization with a column pre-ordering and symbolic analysis phase.

3.1 Overview

UMFPACK3 first finds a column pre-ordering that reduces fill-in, without regard to numerical values. Next, the analysis phase breaks the factorization of the matrix $\mathbf{A}$ down into a sequence of dense rectangular frontal matrices. The frontal matrices are related to each other by a supernodal column elimination tree, in which each node in the tree represents one frontal matrix. This phase also determines upper bounds on the memory usage, the floating-point operation count, and the number of nonzeros in the LU factors.

In its numerical phase, UMFPACK3 factorizes each *chain* of frontal matrices in a single work array, similar to how the unifrontal method [27] factorizes the whole matrix. A chain of frontal matrices is a sequence of fronts where the parent of front i is $i + 1$ in the supernodal column elimination tree. Like all multifrontal methods, UMFPACK3 is an outer-product based, right-looking method. At the k-th step of Gaussian elimination, it represents the updated submatrix $\mathbf{A}_k$ as an implicit summation of a set of dense con-

tribution blocks (also referred to here as *elements*, borrowing a phrase from finite-element methods) that arise when the frontal matrices are factorized and their pivot rows and columns eliminated.

Each frontal matrix represents the elimination of one or more columns; each column of $\mathbf{A}$ will be eliminated in a specific frontal matrix, and which frontal matrix will be used for each column is determined by the analysis phase. This is in contrast to prior multifrontal methods for unsymmetric or symmetric indefinite matrices, in which pivots can be delayed to the parent frontal matrix (and further up the tree as well). It differs from MA38, which has no symbolic analysis at all. The pivot rows are not known ahead of time as they are for the multifrontal method for symmetric positive definite matrices, however.

The analysis phase determines the worst-case size of each frontal matrix so that they can hold any candidate pivot column assigned to them, and any candidate pivot row. From the perspective of the analysis phase, any candidate pivot column in the frontal matrix is identical (in terms of nonzero pattern), and so is any candidate pivot row. A left-looking numerical factorization method does not have any additional information.

However, the right-looking numerical factorization phase of UMFPACK3 has more information than its analysis phase. It uses this information to reorder the columns within each frontal matrix to reduce fill-in. Similarly, since the number of nonzeros in each row and column are maintained (more precisely, COLMMD-style approximate degrees [33]), a pivot row can be selected based on sparsity-preserving criteria as well as numerical considerations (relaxed threshold partial pivoting). This information about row and column degrees is not available to left-looking methods.

Thus, the numerical factorization refines the column ordering $\mathbf{Q}$ by reordering the pivot columns within each front, and it computes the row ordering $\mathbf{P}$, which has the dual role of reducing fill-in and maintaining numerical accuracy (via relaxed partial pivoting and row interchanges).

3.2 Column pre-ordering and symbolic analysis

The column pre-ordering is a slightly modified version of COLAMD [16, 17, 39]. COLAMD finds a symmetric permutation $\mathbf{Q}$ of the matrix $\mathbf{A}^{\mathsf{T}}\mathbf{A}$ (without forming $\mathbf{A}^{\mathsf{T}}\mathbf{A}$ explicitly), and is based on an approximate minimum degree method [1].

Next, the symbolic analysis phase constructs the supernodal column elimination tree. The tree is post-ordered, with the largest child of each node being ordered just before its parent. Next, each frontal matrix is assigned to a unifrontal chain. After the post-ordering, two frontal matrices i and $i+1$ are in the same chain if $i+1$ is the parent of i. The largest frontal matrix in each chain is found; this determines the size of the work array to be used to factorize the chain. In the numerical factorization phase, the unifrontal method will be applied to each chain, with as few as a single contribution block being stacked per chain (more may be created if this results in too large of a contribution block with too many explicitly zero entries). The symbolic phase determines upper bounds on the memory usage, the number of nonzeros in $\mathbf{L}$ and $\mathbf{U}$, and the floating-point operation count. This entire phase, including the ordering, is computed in space proportional to number of nonzeros in $\mathbf{A}$.

MA38 attempts to find unifrontal chains on the fly. The post-ordering of UMFPACK3 finds much longer unifrontal chains, which is why it is able to achieve much higher performance than MA38. Post-ordering the tree also reduces memory usage of the contribution block stack. Performance results are discussed in more detail in Section 4.

3.3 Numerical factorization

The numerical factorization phase starts by allocating several temporary data structures, including a work array that can hold the largest frontal matrix in the supernodal column elimination tree, and a stack to hold the elements. During numerical factorization, the active submatrix $\mathbf{A}_k$ is held as a collection of rectangular elements, one for each non-pivotal column of $\mathbf{A}$ and one for each contribution block created during numerical factorization. To facilitate the scanning of rows and columns, *element lists* [14] for each row and column hold the list of elements that contribute to that row and column. These are also used to compute the COLMMD-style approximate degrees used during numerical factorization.

Let C_i denote the set of $|C_i|$ candidate pivot columns in the ith frontal matrix. The set of non-pivotal columns that can appear in the ith frontal is N_i. Let R_i denote the set of $|R_i|$ candidate pivot rows for the ith frontal matrix. The sum of $|C_i|$ for all i is equal to n; this is not the case for R_i. The upper bound on the size of the frontal matrix is $|R_i|$-by-$(|C_i| + |N_i|)$. If the matrix is structurally nonsingular, $|R_i| \geq |C_i|$ for all i will hold. The

parent of node i is the smallest numbered node that contains a column in N_i as one of its own candidate pivot columns. The Algorithm 1 is an outline of the method (n_B is a parameter, 24 by default).

3.4 Frontal matrix strategy and local pivot search

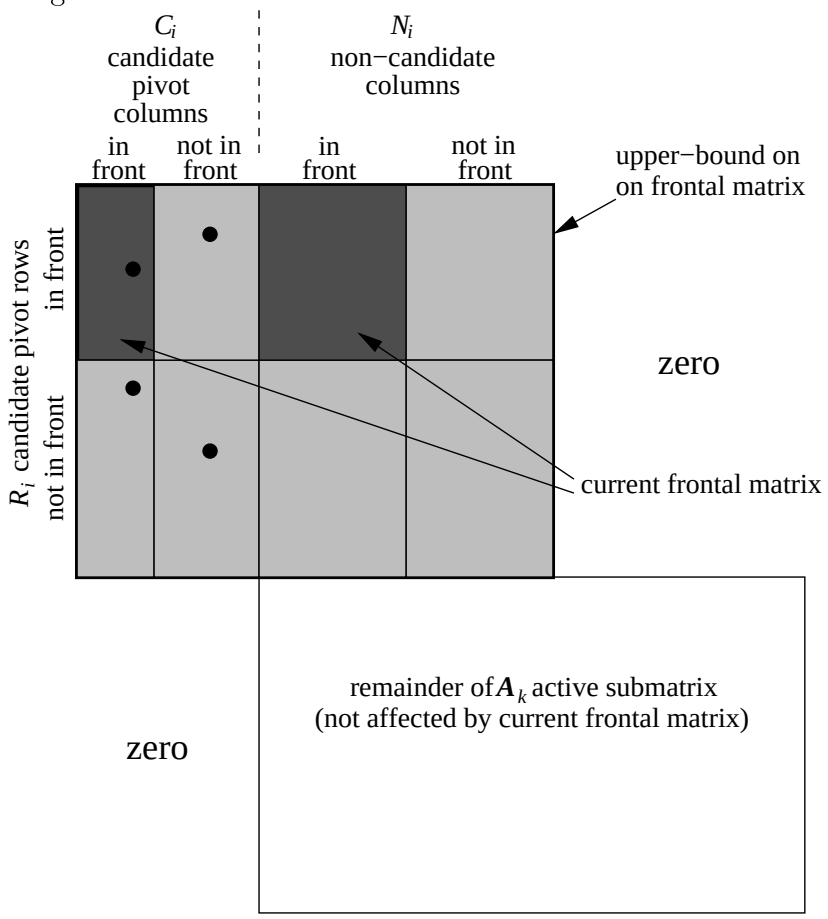
Figure 1 shows the $(n-k)$-by-$(n-k)$ active submatrix $\mathbf{A}_k$ being factorized, and the portion of that matrix that may be held in the work array (the shaded region, which is the upper-bound of the current frontal matrix). The current frontal matrix occupies only part of the work array, shown as the two dark shaded regions. These two regions are actually stored as one contiguous block. During factorization within this frontal matrix, some of the candidate rows in R_i will appear in the frontal matrix, and some may not (some of these may never appear). Likewise, some of the columns in C_i will appear in the front and some will not yet appear (but they are all guaranteed to appear by the time the frontal matrix is fully factorized). Finally, some of the columns in N_i will currently be in the front, but some will not (and like R_i, some may never appear). The frontal matrix has been permuted in this perspective so that the candidate pivot columns C_i are placed first followed by the non-pivotal columns in the set N_i. Note that the work array is large enough to hold all candidate pivot rows (R_i), and all candidate pivot columns (C_i); the part of these rows and columns outside the work array is zero in the active submatrix $\mathbf{A}_k$. The $(k-1)$st and prior pivots are not shown, but some of these are held in the frontal matrix as well and are removed when their pending updates are applied to the contribution block. These outer-product updates are applied via a dense matrix-matrix multiply.

The search for the kth pivot row and column is limited, but it is this step that allows the method to typically obtain orderings that are better than left-looking methods. Up to two candidate pivot columns are examined: the column of least approximate degree in C_i in the current front, and the one of least approximate degree in C_i but not in the current front. In each of these two columns, up to two candidate pivot entries are sought: the candidate row of least approximate degree in the current front, and the row of least approximate degree not in the current front. The pivot entry must also be numerically acceptable (by default, an absolute value of 0.1 times the absolute value of the largest entry in the candidate pivot column, or larger). The row and column degrees are not exact; COLMMD-style approximate degrees are used, which is simply the sum of the sizes of the contribution

Algorithm 1: UMFPACK3 numerical factorization

```
initializations
k = 0
i = 0
for each chain:
    current frontal matrix is empty
    for each frontal matrix in the chain:
        i = i + 1
        for |C_i| iterations:
            k = k + 1
            find the kth pivot row and column
            if too many zero entries in new contribution block
                apply pending updates
                create new contribution block and place on stack
                start a new frontal matrix
            else
                if too many zero entries in new LU part of frontal matrix
                    apply pending updates
                end if
                extend the frontal matrix
            end if
            assemble contribution blocks into current frontal matrix
            scale pivot row and column
            save kth column of L and kth row of U
            if # pivots in current frontal matrix ≥ n_B
                apply pending updates
            end if
        end for |C_i| iterations
    end for each frontal matrix in the chain
    apply pending updates
    create new contribution block and place on stack
end for each chain
```

Figure 1: The active submatrix and current frontal matrix



blocks in each row and column. Tighter approximations were tried (as in COLAMD and AMD), but this was not found to improve the ordering quality. Since the tighter AMD-style approximation requires a second pass over the element lists of the rows and columns in the current frontal matrix, the simpler COLMMD-style approximation was used instead. The tighter AMD-style degree approximation is used only by the column pre-ordering in the symbolic analysis phase.

The candidate pivot entries are shown as four dots in Figure 1. Anywhere from one to four of these candidates may exist. These candidates are evaluated, and the exact degrees of up to two candidate pivot columns and up to four candidate pivot rows are computed. The best one of the candidate pivot entries is chosen as the kth pivot row and column. The metric used to evaluate these candidates is a form of approximate minimum fill-in [43, 44]; the pivot entry that causes the least growth in the size of the actual frontal matrix is chosen.

Increasing the size of the current frontal matrix to include the new pivot row and column may create new zero entries in the frontal matrix, in either the pending pivot rows and columns, or the contribution block, or both. Pending updates are applied if the number of zero entries in the pending pivot rows and columns (not shown in Figure 1) increase beyond a threshold. The updates are also applied, and the current contribution block stacked, if too many zero entries would be included in the old contribution block; in this case a new frontal matrix is started. The latter step also occurs at the end of a chain.

After the pivot search and possible update and/or extension of the frontal matrix, prior contribution blocks are assembled into the current frontal matrix. These are found by scanning the element lists, in the same manner as MA38. The assembly DAG used by MA38 is neither used nor computed in UMFPACK3; its role is replaced by the simpler supernodal column elimination tree computed in the analysis phase. The *kth* pivot row and column are computed and a copy is saved in a separate compressed-index data structure for $\mathbf{L}$ and $\mathbf{U}$. Finally, pending updates are applied if sufficient work has accumulated.

4 Experimental results

In this section the new method, UMFPACK3, is compared with LU, SuperLU, MA38, and the latest "unsymmetric" version of MA41 [5, 6], referred to here as MA41u. Each method (except for LU) was compiled with identical compiler parameters (the highest level of optimization), and executed on a Sun Ultra 80 with 4GB of main memory and four processors. Only one processor was used, although SuperLU and MA41u both have parallel versions. The Sun Performance Library BLAS was used. LU was used within MATLAB Version 6.0. It does not make use of the BLAS.

With the exception of MA38 and LU, all methods used their default parameter settings and ordering methods. MA38 can permute a matrix to block triangular form [11, 20, 22] and then factorize each irreducible diagonal block. This can improve performance for some matrices. The other methods do not include this option, but can be easily adapted to do so via a short MATLAB script. This was tested, and the overall relative results presented here do not change very much. MATLAB uses LU with COLMMD [33] to solve sparse linear systems (`x=A\b`), although its built-in COLAMD routine is faster and generates better orderings [16, 17, 39].

The UF sparse matrix collection [12] includes 357 real, square, unsymmetric sparse matrices. All methods were tested on all but three of the very largest matrices (APPU, PRE2, and XENON2) which could not be factorized on this computer. Statistics gathered for each method included:

- The CPU time for the pre-ordering and symbolic analysis phase, and the numerical factorization phase. The total run time is the sum of these two times.
- The number of nonzeros in $\mathbf{L} + \mathbf{U}$. This excludes the zero entries that most methods explicitly store in their data structures for $\mathbf{L}$ and $\mathbf{U}$. It also excludes the unit diagonal of $\mathbf{L}$, which does not need to be explicitly stored.
- The "canonical" floating-point operation count. This was computed based solely on the nonzero pattern of $\mathbf{L}$ and $\mathbf{U}$,

$$\sum_{k=1}^{n} 2L_k U_k + \sum_{k=1}^{n} L_k$$

where L_k is the number of off-diagonal nonzeros in column k of $\mathbf{L}$, and U_k is the number of off-diagonal nonzeros in row k of $\mathbf{U}$. Both L_k and U_k exclude explicitly stored entries that are numerically zero. The flop count is a function of the quality of the pivot ordering found by the method ($\mathbf{P}$ and $\mathbf{Q}$), and not a function of how the factorization is actually computed.

- The total memory usage, excluding the space required to hold $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{b}$. LU and MA41u do not report this metric. LU's memory usage when $\mathbf{A}$ is real and square is 12 bytes per nonzero in $\mathbf{L}+\mathbf{U}$ (excluding the unit diagonal of $\mathbf{L}$, which is explicitly stored), plus $53n$ bytes for $\mathbf{P}$, $\mathbf{Q}$, the rest of the data structures for $\mathbf{L}$ and $\mathbf{U}$, and temporary work space [32]. The memory usage of MA41u was found via binary search on the size of its two work arrays (one double precision array and one integer array).

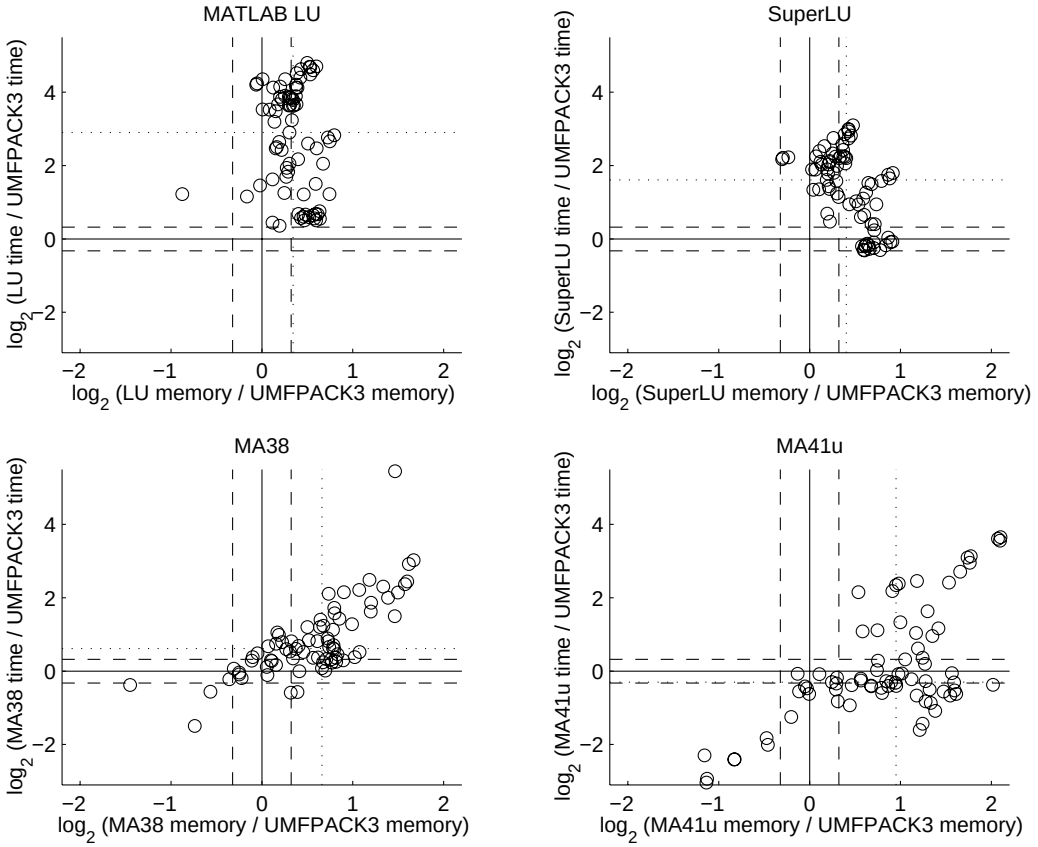
- The norm of the residual, $||\mathbf{Ax}-\mathbf{b}||_\infty$.

LU uses a default threshold partial pivoting ratio of 1.0 (true partial pivoting), while UMFPACK3 uses its default of 0.1 (entries in the $\mathbf{L}$ computed by UMFPACK3 have a magnitude of 10 or less). All methods except LU use iterative refinement with sparse backward error [7] in their forward/backward solve step. UMFPACK3 was found to be just as accurate as LU.

The symmetry of the pattern a sparse matrix is defined as the number of matched off-diagonal entries over the total number of off-diagonal entries. An entry a_{ij} is matched if $i \neq j$ and a_{ji} is also an entry. A matrix with a symmetric pattern has a symmetry of one; a completely asymmetric pattern has a symmetry of zero. The test set is split into two parts, unsymmetric matrices (symmetry < 0.5) and "symmetric" (symmetry ≥ 0.5). Only the larger matrices are considered, defined as those for which LU requires 10^7 or more floating-point operations. Numerically singular matrices are discarded. This leads to a test set of 77 unsymmetric matrices and 94 symmetric ones.

Results for all 77 unsymmetric matrices are shown in Figure 2. Each of the four plots depicts the relative results of one method as compared to UMFPACK3. Each circle on the plot is a single matrix. The x-axis is the $\log_2$ of ratio of the memory required to factorize the matrix using the specific method over the memory required by UMFPACK3 to factorize the matrix. The y-axis is the $\log_2$ of the relative total run time. Thus, a circle in the upper right quadrant depicts a matrix for which the specific method requires

Figure 2: Results relative to UMFPACK3 (unsymmetric-patterned matrices)



more time and more memory than UMFPACK3. The x-y axes are drawn as solid lines; these are bracketed by dashed lines representing relative results of 0.8 and 1.25 respectively. Within these two dashed lines, the results for the specific method and UMFPACK3 are comparable. Median relative run time and memory usage results are shown as dotted lines. Figure 3 reports the same results for the 94 symmetric-patterned matrices. All eight plots in the two figures use the same x-y axes for ease of comparison. This means a few outliers are not shown in some plots of Figure 3. Table 1 reports the median relative results for these two sets of matrices, as well as two metrics not shown in Figures 2 and 3: the relative number of canonical floating point operations, and the relative number of nonzeros in $\mathbf{L} + \mathbf{U}$.

A selection of matrices is given Tables 2 through 4. These are the

Figure 3: Results relative to UMFPACK3 (symmetric-patterned matrices)

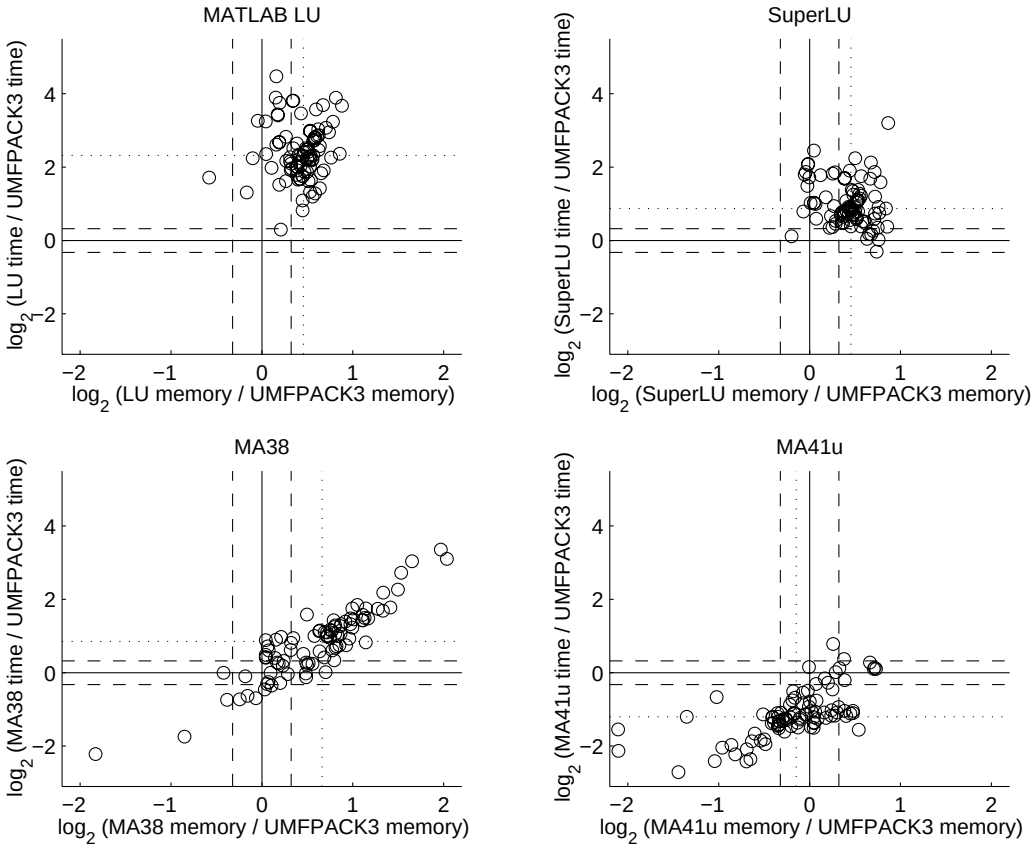


Table 1: Median relative results vs. UMFPACK3

	LU	SuperLU	MA38	MA41u
unsymmetric test set				
time:	7.47	3.05	1.53	0.81
flop:	1.59	1.58	1.88	1.70
memory:	1.27	1.32	1.58	1.94
nnz in $\mathbf{L}+\mathbf{U}$:	1.28	1.27	1.30	1.18
symmetric test set				
time:	4.99	1.83	1.76	0.43
flop:	1.26	1.26	2.06	0.56
memory:	1.37	1.37	1.58	0.90
nnz in $\mathbf{L}+\mathbf{U}$:	1.13	1.13	1.27	0.68

Table 2: Selected unsymmetric-patterned matrices

Source	Name	n	nnz in 1000's	sym.	description
Bai	RW5151	5151	20.2	0.490	Markov chain, random walk
Grund	BAYER01	57735	275.1	0.000	chemical process simulation
Mallya	LHR71C	70304	1528.1	0.002	chemical process simulation
Zitney	RDIST1	4134	94.4	0.059	chemical process simulation
Vavasis	AV41092	41092	1683.9	0.001	irregular finite-element problem
Hollinger	G7JAC200	59310	717.6	0.032	economics, G7 social security
Hollinger	MARK3JAC140	64089	376.4	0.074	economics, Intl. Monetary Fund

Table 3: Selected circuit simulation matrices

Source	Name	n	nnz in 1000's	sym.	description
Grund	MEG1	2904	58.1	0.002	1MB memory circuit
Grund	MEG4	5860	25.3	1.000	4MB memory circuit
AT&T	TWOTONE	120750	1206.3	0.245	frequency domain, harmonic balance
Bomhof	CIRCUIT_1	2624	35.8	1.000	differential algebraic equations
Bomhof	CIRCUIT_4	80209	307.6	0.829	differential algebraic equations
Hamm	MEMPLUS	17758	99.1	1.000	memory circuit
Hamm	SCIRCUIT	170998	958.9	1.000	digital circuit with parasitics

Table 4: Selected symmetric-patterned matrices

Source	Name	n	nnz in 1000's	sym.	description
Li	LI	22695	1215.2	1.000	3D, magnetohydrodynamics
Wang	WANG4	26068	177.2	1.000	3D MOSFET semiconductor
Zhao	ZHAO2	33861	166.5	0.922	electromagnetic system
Ronis	XENON1	48600	1181.1	1.000	complex zeolite/sodalite crystal
Simon	BBMAT	38744	1771.7	0.530	2D airfoil
Simon	RAEFSKY4	19779	1316.8	1.000	buckling problem for container model
FIDAP	EX11	16614	1096.9	1.000	3D, cylinder & flat plate heat exch.

largest matrices within each class (matrix source or problem type). Circuit simulation matrices occurred in both the unsymmetric and symmetric sets; a selection of these are listed separately in Table 3.

Results for these matrices are given in Tables 5 through 7, which lists the run time in seconds (including the symbolic analysis and ordering phase), the canonical floating-point operation count (in millions), the total amount of memory used (in megabytes), and the number of nonzeros in $\mathbf{L}+\mathbf{U}$ (in thousands; this count excludes the unit diagonal of $\mathbf{L}$) for LU, SuperLU, UMFPACK3, MA38, and MA41u. Table 6 also shows the results for UMFPACK3 with a non-default column ordering from SYMAMD applied to $\mathbf{A}+\mathbf{A}^{\mathsf{T}}$, and with a non-default option that prefers the diagonal entry as pivot. Results within 25% of the best result for a particular matrix (excluding the non-default ordering for UMFPACK3) are shown in bold. Note that a few of the circuit simulation matrix results do not fit in the axes chosen for Figure 3.

These results must be interpreted with caution; the test set is not a statistical sample of all sparse matrices encountered in practice, and the run time results can differ depending on the computer used. SuperLU and MA41u both have parallel versions. LU, UMFPACK3, and MA38 do not. UMFPACK3 is based on the supernodal column elimination tree, which could be used to guide a parallel version of UMFPACK3. MA38 has no such tree, although a parallel re-factorize algorithm based on the elimination DAG found in a prior sequential numerical factorization has been developed [2, 35, 36].

The left-looking methods LU and SuperLU find nearly identical orderings because they use the same pre-ordering and pivoting strategy. UMFPACK3 behaves most similarly to the left-looking methods. This is to be expected, since all three methods use the same pre-ordering, are based on the column elimination tree, and exploit the same worst-case upper bound on fill-in. Because UMFPACK3 can further refine the row and column ordering based on the row and column degrees available during factorization, it is typically able to find a better pivot ordering than LU and SuperLU, leading to fewer floating-point operations, fewer nonzeros in $\mathbf{L}$ and $\mathbf{U}$, and less memory usage. UMFPACK3 is typically faster than LU and SuperLU in this particular environment, more so for the largest matrices.

UMFPACK3 makes efficient use of memory, even though it needs a large work array to hold a contribution block stack, and another large work array for the current frontal matrix. Table 8 lists the median number of bytes per nonzero entry in $\mathbf{L}+\mathbf{U}$; this is the total memory usage including all work

Table 5: Results for selected unsymmetric-patterned matrices

Matrix		LU	SuperLU	UMF3	MA38	MA41u
Bai	time:	1.3	0.6	**0.4**	0.8	**0.5**
RW5151	flop:	28.1	28.3	**21.6**	48.9	37.7
	mem:	**4.1**	5.0	**3.4**	5.6	5.7
	nnz LU:	**333.6**	**336.0**	**295.7**	392.2	**312.4**
Grund	time:	3.9	**2.5**	**3.1**	3.7	**2.6**
BAYER01	flop:	33.4	33.6	**18.3**	26.1	37.7
	mem:	**18.0**	27.0	**15.8**	**17.0**	34.3
	nnz LU:	1320.8	1305.4	**1004.9**	**1044.2**	1283.2
Mallya	time:	181.2	163.4	123.7	151.4	**45.7**
LHR71C	flop:	492.4	492.1	**346.8**	932.9	528.7
	mem:	84.7	89.3	**54.3**	101.1	128.7
	nnz LU:	**7086.7**	**7086.6**	**5960.4**	8783.0	**7087.4**
Zitney	time:	1.0	0.4	0.4	**0.3**	**0.4**
RDIST1	flop:	14.9	14.9	**6.6**	7.0	**5.6**
	mem:	4.3	4.7	**2.6**	3.4	3.8
	nnz LU:	357.6	357.6	**234.1**	**230.2**	**210.4**
Vavasis	time:	3438.9	843.9	197.1	346.7	**40.1**
AV41092	flop:	74015.6	73965.0	33310.9	63999.3	**3543.3**
	mem:	491.7	436.7	374.9	469.7	**168.3**
	nnz LU:	42782.8	42764.2	39140.1	38139.6	**9055.7**
Hollinger	time:	3164.7	1057.9	**254.1**	331.5	-
G7JAC200	flop:	61457.9	61821.6	**54740.5**	**44344.0**	-
	mem:	532.1	546.4	**416.4**	**388.2**	-
	nnz LU:	46235.8	46358.0	**40575.3**	**32484.7**	-
Hollinger	time:	7006.6	2156.2	**270.2**	2198.1	1485.3
MARK3JAC140	flop:	119173.3	110564.0	**63216.1**	289004.0	391125.9
	mem:	641.0	598.7	**444.7**	1415.0	1007.7
	nnz LU:	55730.1	53352.7	**40215.3**	79813.1	90927.8

Table 6: Results for selected circuit simulation matrices

Matrix		LU	SuperLU	UMF3 (default)	MA38	MA41u	UMF3 w/ SYMAMD
Grund	time:	1.2	1.3	0.5	**0.4**	0.5	4.1
MEG1	flop:	19.5	42.2	**2.5**	3.2	30.2	60.3
	mem:	3.0	5.9	5.5	**2.0**	9.2	10.8
	nnz LU:	249.0	355.3	**122.0**	**130.9**	303.9	410.1
Grund	time:	1.2	3.5	0.4	**0.1**	**0.1**	0.3
MEG4	flop:	28.1	28.1	14.2	**0.2**	**0.3**	0.1
	mem:	4.0	10.8	6.0	**1.1**	**1.4**	0.2
	nnz LU:	322.3	322.3	242.6	**45.4**	**45.0**	37.1
AT&T	time:	449.8	169.9	**60.2**	**67.5**	**57.6**	3717.5
TWOTONE	flop:	**8445.3**	**8480.3**	10787.2	**10126.3**	**9360.6**	281806.2
	mem:	**190.7**	248.2	**154.7**	**172.2**	312.8	1043.0
	nnz LU:	16131.4	16201.0	14975.0	**14061.6**	**11544.5**	132972.7
Bomhof	time:	16.2	4.6	1.3	0.4	**0.3**	0.3
CIRCUIT_1	flop:	442.4	442.4	2.1	**0.9**	**0.8**	0.8
	mem:	11.8	10.5	6.4	**1.2**	1.5	3.4
	nnz LU:	1019.9	1019.1	**54.5**	**44.0**	**44.6**	42.9
Bomhof	time:	5241.7	2090.5	1114.3	**12.1**	**10.8**	37.7
CIRCUIT_4	flop:	103079.9	101322.0	139762.6	**6.3**	**5.8**	6.3
	mem:	435.8	474.7	2279.2	**14.4**	18.5	55.0
	nnz LU:	37728.7	39799.7	43599.1	**443.0**	**487.6**	440.6
Hamm	time:	152.5	37.4	6.9	0.9	**0.4**	2.0
MEMPLUS	flop:	3422.3	3422.2	10.9	**1.6**	**1.8**	1.6
	mem:	38.3	35.4	34.2	**4.0**	**4.6**	9.4
	nnz LU:	3264.8	3264.8	220.9	**133.0**	**140.9**	122.4
Hamm	time:	30.0	15.4	11.9	7.7	**4.2**	157.7
SCIRCUIT	flop:	582.0	582.0	307.6	93.7	**57.7**	59.3
	mem:	80.4	100.7	**55.7**	**49.9**	**57.4**	75.1
	nnz LU:	6272.3	6272.3	4819.4	**2963.1**	**2942.3**	2552.3

Table 7: Results for selected symmetric-patterned matrices

Matrix		LU	SuperLU	UMF3	MA38	MA41u
Li	time:	5210.4	1889.9	542.6	1085.8	**385.1**
LI	flop:	109145.9	109180.0	115764.5	193152.6	**83254.0**
	mem:	**663.9**	**660.8**	**685.7**	1024.8	**676.9**
	nnz LU:	**57910.2**	**57924.3**	**56959.6**	91050.9	**47011.6**
Wang	time:	1586.7	388.8	106.7	197.1	**25.9**
WANG4	flop:	32171.2	32171.2	30497.4	40076.3	**10472.1**
	mem:	292.7	255.5	263.1	271.3	**135.3**
	nnz LU:	25460.7	25460.7	23876.1	24309.5	**11472.2**
Zhao	time:	691.5	179.5	**49.5**	327.1	85.2
ZHAO2	flop:	**11546.8**	**11932.4**	**11121.2**	60281.0	15056.6
	mem:	196.8	**189.3**	**155.7**	450.6	**186.3**
	nnz LU:	**17045.3**	**17231.3**	**14967.0**	33259.2	**17250.9**
Ronis	time:	4082.7	999.1	290.4	780.1	**62.1**
XENON1	flop:	88811.1	88811.0	88844.1	157709.8	**26790.6**
	mem:	702.3	600.1	552.5	955.6	**313.2**
	nnz LU:	61153.7	61153.6	59024.2	67816.8	**30118.5**
Simon	time:	2054.7	645.6	172.6	1767.6	**125.9**
BBMAT	flop:	**45069.5**	**45092.9**	**37749.5**	342827.6	**40415.2**
	mem:	571.8	514.4	**378.5**	1482.2	**450.8**
	nnz LU:	**49795.9**	**49810.1**	**44086.8**	131470.3	**44886.1**
Simon	time:	539.2	173.2	49.0	147.0	**20.6**
RAEFSKY4	flop:	13408.8	13408.8	12888.4	29327.7	**8247.9**
	mem:	245.4	216.9	**181.9**	256.3	**151.7**
	nnz LU:	21354.5	21354.5	20933.5	23812.3	**13336.5**
FIDAP	time:	496.1	145.9	33.4	274.6	**19.8**
EX11	flop:	11822.3	11888.1	**6756.5**	53393.5	**7278.7**
	mem:	217.5	197.6	**123.8**	389.2	**130.5**
	nnz LU:	18928.0	18967.1	**13983.8**	33196.1	**12520.6**

Table 8: Memory usage (median bytes per nonzero in $\mathbf{L}+\mathbf{U}$)

Test set	LU	SuperLU	UMF3	MA38	MA41u
unsymmetric	12.3	12.7	12.6	14.3	19.0
symmetric	12.3	11.9	10.4	12.8	13.8

arrays (but excluding the matrix $\mathbf{A}$) divided by the total number of nonzeros in $\mathbf{L}+\mathbf{U}$. Eight bytes are required to hold the numerical value of each entry $\mathbf{L}$ or $\mathbf{U}$ itself; an integer row or column index takes four bytes.

UMFPACK3 is more memory-efficient that its predecessor, MA38, in terms of bytes per entry in the resulting LU factors, primarily because of its use of a post-ordered supernodal column elimination tree. UMFPACK3 tends to be faster than MA38 and finds better orderings for most matrices, with the notable exception of circuit simulation matrices. These matrices often have dense rows, and thus $\mathbf{A}^\mathsf{T}\mathbf{A}$ tends to be dense. In this case, the worst-case upper bound is too pessimistic. MA38 uses an approximate-degree Markowitz criteria on the pattern of the active submatrix $\mathbf{A}_k$ itself, and does not consider $\mathbf{A}^\mathsf{T}\mathbf{A}$. This is also true of MA41u, which uses an approximate-degree ordering on the pattern of $\mathbf{A}+\mathbf{A}^\mathsf{T}$ (AMD, [1]). Both MA38 and MA41u attempt to reduce an optimistic lower bound on fill-in. The performance of UMFPACK3 on these matrices can be improved dramatically when a non-default symmetric ordering is used instead.

For matrices with unsymmetric nonzero pattern, UMFPACK3 typically uses less memory than MA41u. The latter method ran out of memory for one matrix in the test set. In terms of run time, MA41u and UMFPACK3 are roughly split, in terms of which method is fastest on which particular unsymmetric matrix. For symmetric-patterned matrices, MA41u is nearly always the fastest method of those considered here, and it also tends to find a better ordering in terms of fill-in and floating-point operation count. UMFPACK3 has less memory overhead per nonzero in $\mathbf{L}$ and $\mathbf{U}$, however, and thus it can still factorize some symmetric-patterned matrices using less memory than MA41u.

Brainman and Toledo [10] use a nested dissection dissection method (recursive graph partitioning) for finding good orderings for left-looking methods. Their method partitions $\mathbf{A}^\mathsf{T}\mathbf{A}$ without forming $\mathbf{A}^\mathsf{T}\mathbf{A}$ explicitly, using wide separators of $\mathbf{A}+\mathbf{A}^\mathsf{T}$. It is particularly well suited to matrices arising in 2D and 3D problems, finding better orderings than COLAMD for those

matrices. Most of the symmetric-patterned matrices reported in Table 4 fall in this category. Since UMFPACK3 exploits the same upper bound on fill-in as left-looking methods, and since UMFPACK3 can accept any given column pre-ordering instead of its default COLAMD ordering, their method can be applied to UMFPACK3 to improve its performance on 2D and 3D problems.

5 Summary

The new method presented here, UMFPACK3, tends to perform better than left-looking methods (LU and SuperLU) on a wide range of matrices. It typically uses less memory and finds better orderings. It takes less total time on the particular single-processor computer used for the comparisons (although a parallel version of SuperLU does exist). Based on the same criteria, it typically outperforms its predecessor, MA38, on all matrices with the exception of those arising in circuit simulation problems. For matrices with moderate or highly unsymmetric nonzero pattern, UMFPACK3 typically finds a better ordering that MA41u and uses less memory, but it is not always faster. MA41u is clearly superior for matrices with symmetric nonzero pattern (in terms of fill-in and run time), although UMFPACK3 can still factorize many symmetric matrices using less memory than MA41u, even a few very large ones.

UMFPACK Version 3.2, prior versions of UMFPACK, COLAMD, SYMAMD, and AMD are available at www.cise.ufl.edu/research/sparse, and as a collected algorithm of the ACM (Algorithm 8xx) [13]. UMFPACK3 will also appear as a built-in routine in a future release of MATLAB as a replacement for LU and the forward and backslash matrix operator (`x = A\b`).

References

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Applic.*, 17(4):886–905, 1996.

[2] P. R. Amestoy, T. A. Davis, I. S. Duff, and S. M. Hadfield. Parallelism in multifrontal methods for matices with unsymmetric structures. In

Abstracts of the Second SIAM Conference on Sparse Matrices, Snowbird, Utah, Oct. 1996.

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

[4] P. R. Amestoy, I. S. Duff, and C. Puglisi. Multifrontal QR factorization in a multiprocessor environment. *Numer. Linear Algebra Appl.*, 3(4):275–300, 1996.

[5] P. R. Amestoy, I. S. Duff, and C. Puglisi. MA41 subroutine (unsymmetrized version). Harwell Subroutine Library, Aug. 2000.

[6] P. R. Amestoy and C. Puglisi. An unsymmetrized multifrontal LU factorization. Technical Report RT/APO/0003, ENSEEIHT-IRIT, Toulouse, France, Sept. 2000.

[7] M. Arioli, J. W. Demmel, and I. S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. Applic.*, 10(2):165–190, 1989.

[8] C. C. Ashcraft and R. Grimes. The influence of relaxed supernode partitions on the multifrontal method. *ACM Trans. Math. Softw.*, 15(4):291–309, 1989.

[9] C. C. Ashcraft and J. W. H. Liu. Robust ordering of sparse matrices using multisection. *SIAM J. Matrix Anal. Applic.*, 19(3):816–832, 1998.

[10] I. Brainman and S. Toledo. Nested-dissection orderings for sparse LU with partial pivoting. 2001. www.math.tau.ac.il/∼stoledo.

[11] T. F. Coleman, A. Edenbrandt, and J. R. Gilbert. Predicting fill for sparse orthogonal factorization. *J. of the ACM*, 33:517–532, 1986.

[12] T. A. Davis. University of Florida sparse matrix collection. www.cise.ufl.edu/research/sparse.

[13] T. A. Davis. Algorithm 8xx: UMFPACK V3.2, an unsymmetric-pattern multifrontal method with a column pre-ordering strategy. Technical Report TR-02-002, Univ. of Florida, CISE Dept., Gainesville, FL, January 2002. (www.cise.ufl.edu/tech-reports. Submitted to *ACM Trans. Math. Softw.)*.

[14] T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Anal. Applic.*, 18(1):140–158, 1997.

[15] T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. Math. Softw.*, 25(1):1–19, 1999.

[16] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. Algorithm 8xx: COLAMD, a column approximate minimum degree ordering algorithm. Technical Report TR-00-006, Univ. of Florida, CISE Dept., Gainesville, FL, October 2000. (www.cise.ufl.edu/tech-reports. Submitted to *ACM Trans. Math. Softw.)*.

[17] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. A column approximate minimum degree ordering algorithm. Technical Report TR-00-005, Univ. of Florida, CISE Dept., Gainesville, FL, October 2000. (www.cise.ufl.edu/tech-reports. Submitted to *ACM Trans. Math. Softw.)*.

[18] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Applic.*, 20(3):720–755, 1999.

[19] J. J. Dongarra, J. J. Du Croz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 16:1–17, 1990.

[20] I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Trans. Math. Softw.*, 7:315–330, 1981.

[21] I. S. Duff. The solution of nearly symmetric sparse linear systems. In R. Glowinski and J.-L. Lions, editors, *Computing methods in applied sciences and engineering, VI*, pages 57–74, Amsterdam New York and London, 1984. North Holland.

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

[23] I. S. Duff and J. K. Reid. MA27 - a set of Fortran subroutines for solving sparse symmetric sets of linear equations. Technical Report

AERE-R-10533, AERE Harwell Laboratory, United Kingdom Atomic Energy Authority, 1982.

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

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

[26] I. S. Duff and J. K. Reid. The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Trans. Math. Softw.*, 22(2):187–226, 1996.

[27] I. S. Duff and J. A. Scott. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Math. Softw.*, 22(1):30–45, 1996.

[28] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

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

[30] A. George and E. G. Ng. An implementation of Gaussian elimination with partial pivoting for sparse systems. *SIAM J. Sci. Statist. Comput.*, 6(2):390–409, 1985.

[31] A. George and E. G. Ng. Symbolic factorization for sparse Gaussian elimination with partial pivoting. *SIAM J. Sci. Statist. Comput.*, 8(6):877–898, Nov. 1987.

[32] J. R. Gilbert. personal communication.

[33] J. R. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in MATLAB: design and implementation. *SIAM J. Matrix Anal. Applic.*, 13(1):333–356, 1992.

[34] J. R. Gilbert and T. Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Statist. Comput.*, 9:862–874, 1988.

[35] S. Hadfield. *On the LU Factorization of Sequences of Identically Structured Sparse Matrices within a Distributed Memory Environment.* PhD thesis, University of Florida, Gainesville, FL, April 1994.

[36] S. M. Hadfield and T. A. Davis. The use of graph theory in a parallel multifrontal method for sequences of unsymmetric pattern sparse matrices. *Congressus Numerantium*, 108:43–52, 1995.

[37] B. Hendrickson and E. Rothberg. Improving the runtime and quality of nested dissection ordering. *SIAM J. Sci. Comput.*, 20:468–489, 1999.

[38] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20:359–392, 1998.

[39] S. I. Larimore. An approximate minimum degree column ordering algorithm. Technical Report TR-98-016, Univ. of Florida, Gainesville, FL, 1998. www.cise.ufl.edu/tech-reports.

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

[41] J. W. H. Liu. The multifrontal method for sparse matrix solution: Theory and practice. *SIAM Review*, 34(1):82–109, 1992.

[42] P. Matstoms. Sparse QR factorization in MATLAB. *ACM Trans. Math. Softw.*, 20(1):136–159, 1994.

[43] E. G. Ng and P. Raghavan. Performance of greedy ordering heuristics for sparse Cholesky factorization. *SIAM J. Matrix Anal. Applic.*, 20(4):902–914, 1999.

[44] E. Rothberg and S. C. Eisenstat. Node selection strategies for bottom-up sparse matrix orderings. *SIAM J. Matrix Anal. Applic.*, 19(3):682–695, 1998.

[45] M. Yannakakis. Computing the minimum fill-in is NP-Complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.