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

Timothy A. Davis*

May 6, 2003

Abstract

A new method for sparse LU factorization is presented that combines a 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. The method analyzes the matrix and automatically selects one of three pre-ordering and pivoting strategies. The number of nonzeros in the LU factors computed by the method is typically less than or equal to those found by a wide range of unsymmetric sparse LU factorization methods, including left-looking methods and prior multifrontal methods.

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.

*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. This work was supported by the National Science Foundation, under grants ASC-9111263, DMS-9223088, and DMS-0203270. Portions of the work were done while on sabbatical at Stanford University and Lawrence Berkeley National Laboratory (with funding from Stanford University and the SciDAC program).

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 pre-ordering and symbolic analysis of the method presented here is similar to that used by left-looking methods such as SuperLU [21] or LU [42, 43] in MATLAB (prior to version 6.5). 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, existing left-looking methods do not 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 priori 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 existing left-looking methods do not do).

UMFPACK factorizes the matrix $\mathbf{PAQ}$, $\mathbf{PRAQ}$, or $\mathbf{PR}^{-1}\mathbf{AQ}$ into the product $\mathbf{LU}$, where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular, respectively, $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices, and $\mathbf{R}$ is a diagonal matrix of row scaling factors. Both $\mathbf{P}$ and $\mathbf{Q}$ are chosen to reduce fill-in (new nonzeros in $\mathbf{L}$ and $\mathbf{U}$ that are not present in $\mathbf{A}$). The permutation $\mathbf{P}$ has the dual role of reducing fill-in and maintaining numerical accuracy (via relaxed partial pivoting). UMFPACK analyzes the matrix, and then automatically selects one of three strategies for pre-ordering the rows and columns: *unsymmetric*, *2-by-2*, and *symmetric*. Equipped with these three strategies, the number of nonzeros in the LU factors computed by UMFPACK is typically less than or equal to that computed by left-looking methods [21, 42, 43], the symmetric-pattern multifrontal method MA41 [4, 25, 31, 32], and the asymmetrized version of MA41 [7]. UMFPACK nearly always uses less memory than these other methods as well.

Section 2 provides a background of the new method: 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 are given in Section 4, and qualitative observations about these results are made in Section 5. A few concluding remarks and information on the availability of the code are given in Section 6.

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 [59], and thus heuristics are used.

Suppose the column ordering $\mathbf{Q}$ is fixed, and let $\mathbf{C} = \mathbf{AQ}$. Sparse Gaussian elimination 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 $\mathbf{C}^\mathsf{T}\mathbf{C}$ [37]. 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 $(\mathbf{AQ})^\mathsf{T}\mathbf{AQ}$ [35, 37, 38]. The COLMMD [42] and COLAMD [19, 20, 50] routines in MATLAB find an ordering $\mathbf{Q}$ without constructing the nonzero pattern of $\mathbf{A}^\mathsf{T}\mathbf{A}$.

For unsymmetric matrices with substantial entries on the diagonal (or diagonally dominant) and a mostly symmetric nonzero pattern, it is often better to use a strategy that finds a fill-reducing ordering $\mathbf{Q}$ that minimizes the fill-in in the Cholesky factorization of a matrix whose nonzero pattern is the same as the matrix $\mathbf{Q}^\mathsf{T}(\mathbf{A} + \mathbf{A}^\mathsf{T})\mathbf{Q}$. This pre-ordering assumes that the diagonal entry can typically be selected as a pivot. This method is used in the right-looking multifrontal method MA41 [4, 25, 31, 32] and its asymmetrized version [7]. If this $\mathbf{Q}$ is used as a column pre-ordering for sparse Gaussian elimination with standard partial pivoting, the upper bound on fill-in can be high, but the actual fill-in is similar to the related Cholesky factorization.

2.2 Left-looking methods

Left-looking methods such as Gilbert and Peierls' LU organize their computation with the column elimination tree (the elimination tree of $\mathbf{C}^\mathsf{T}\mathbf{C}$ [52]). SuperLU uses the supernodal column elimination tree to reduce execution time by exploiting dense matrix kernels (the BLAS [22]) in the computation of each super-column (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 [33]. It differs from LU and SuperLU by using a partial right-looking numerical factorization as its pre-ordering strategy (the matrix is numerically factorized up to but not including a switch to a dense matrix factorization method).

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 $\mathbf{A}$ 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 [43]. 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 $\mathbf{Q}$ 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.

It would be possible for a left-looking method to maintain a right-looking representation of the Schur complement in order to select rows based on sparsity-preserving criteria and to rearrange columns within each super-column. No existing left-looking method maintains this representation, however, since it is not needed to compute the kth column of $\mathbf{L}$ and $\mathbf{U}$ at the kth step of factorization. It would be used only for pivot row selection, which existing methods do solely for numerical accuracy and not to reduce fill-in. The result would be a hybrid algorithm, one that does its numerical computations in a left-looking manner, but maintains the pattern of the Schur complement in a right-looking manner.

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 $\mathbf{A}$ 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 prior 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. Multiple elimination steps are done within the frontal matrix, which allows the the Schur complement to be computed with a dense matrix-matrix multiplication (DGEMM [22]), 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 [9, 53],
2. symmetric indefinite matrices (MA27, MA47, and MA57) [29, 30, 26],
3. unsymmetric matrices with actual or implied symmetric nonzero pattern (MA37 and MA41) [4, 25, 31, 32],
4. unsymmetric matrices where the unsymmetric nonzero pattern is partially preserved (MA41u) [7],
5. unsymmetric matrices where the unsymmetric nonzero pattern is fully preserved (MA38 and WSMP) [17, 18, 45],
6. and QR factorization of rectangular matrices [6, 54].

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 [52, 53]. 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 [27] 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 and Puglisi [7] (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 may be rectangular, but 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, 36] or nested-dissection ordering [10, 35, 48, 49] as part of a symbolic analysis phase.

MA38 (UMFPACK Version 2.2.1) is based on the fifth approach. It 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. WSMP differs from MA38 in that it computes the DAG in a symbolic analysis phase.

The last approach, multifrontal QR factorization [6, 54], is based on the column elimination tree of $\mathbf{A}$.

3 The algorithm

An overview of the new algorithm (UMFPACK Version 4.1, or simply UMFPACK4) is given below, followed by details of its implementation.

3.1 Overview

UMFPACK4 first finds a column pre-ordering that reduces fill-in. It scales and analyzes the matrix, and then automatically selects one of three strategies for pre-ordering the rows and

columns: *unsymmetric*, *2-by-2*, and *symmetric*. These strategies are described below.

First, all pivots with zero Markowitz cost are eliminated and placed in the LU factors. These are pivots whose pivot row or column (or both) have only one nonzero entry, and can thus be eliminated without causing any fill-in in the remaining submatrix. A permutation to block triangular form [23] would also reveal these pivots, but UMFPACK4 does not perform this permutation.

The remaining submatrix $\mathbf{S}$ is then analyzed. If the nonzero pattern of the matrix $\mathbf{S}$ is very unsymmetric, the unsymmetric strategy is used. If the pattern is nearly symmetric and the matrix has a zero-free diagonal, the symmetric strategy is used. Otherwise, the 2-by-2 strategy is attempted. The 2-by-2 strategy finds a row permutation $\mathbf{P}_2$ which attempts to reduce the number of small diagonal entries of $\mathbf{P}_2\mathbf{S}$. If s_{ii} is numerically small, the method attempts to swap two rows i and j such that both s_{ij} and s_{ji} are large. Once these rows are swapped they remain in place. The four entries s_{ii}, s_{ij}, s_{ji}, and s_{jj} are analogous to the 2-by-2 pivoting strategy used for symmetric indefinite matrices [13] and have similar fill-in properties. This permutation is not guaranteed to result in a zero-free diagonal, but it tends to preserve symmetry better than a complete zero-free matching [24, 28]. If the nonzero pattern of $\mathbf{P}_2\mathbf{S}$ is sufficiently symmetric, and its diagonal is mostly zero-free, the 2-by-2 strategy is used. Otherwise, the unsymmetric strategy is used.

Each strategy is described below:

- *unsymmetric*: The column pre-ordering of $\mathbf{S}$ is computed by a modified version of COLAMD [19, 20, 50]. The method finds a symmetric permutation $\mathbf{Q}$ of the matrix $\mathbf{S}^\mathsf{T}\mathbf{S}$ (without forming $\mathbf{S}^\mathsf{T}\mathbf{S}$ explicitly). This is a good choice for $\mathbf{Q}$, since the Cholesky factors of $(\mathbf{SQ})^\mathsf{T}(\mathbf{SQ})$ are an upper bound (in terms of nonzero pattern) of the factor $\mathbf{U}$ for the unsymmetric LU factorization ($\mathbf{PSQ} = \mathbf{LU}$) regardless of the choice of $\mathbf{P}$ [37, 38, 39]. This modified version of COLAMD also computes the column elimination tree and post-orders the tree. It finds the upper bound on the number of nonzeros in $\mathbf{L}$ and $\mathbf{U}$. It also has a different threshold for determining dense rows and columns. During factorization, the column pre-ordering can be modified. Columns within a single super-column can be reshuffled, to reduce fill-in. Threshold partial pivoting is used with no preference given to the diagonal entry. Within a given pivot column j, an entry a_{ij} can be chosen if $|a_{ij}| \geq 0.1 \max |a_{*j}|$. Among those numerically acceptable entries, the sparsest row i is chosen as the pivot row.

- *symmetric*: The column ordering is computed from AMD [1], applied to the pattern of $\mathbf{S} + \mathbf{S}^\mathsf{T}$ followed by a post-ordering of the supernodal elimination tree of $\mathbf{S} + \mathbf{S}^\mathsf{T}$. No modification of the column pre-ordering is made during numerical factorization. Threshold partial pivoting is used, with a strong preference given to the diagonal entry. The diagonal entry is chosen if $a_{jj} \geq 0.001 \max |a_{*j}|$. Otherwise, a sparse row is selected, using the same method used by the unsymmetric strategy (find the sparsest pivot row, using a threshold of 0.1).

- *2-by-2*: The symmetric strategy is applied to the matrix $\mathbf{P}_2\mathbf{S}$, rather than $\mathbf{S}$.

Once the strategy is selected, the factorization of the matrix $\mathbf{A}$ is broken down into the factorization of 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 (the symmetric and 2-by-2 strategies thus use both the elimination tree and the column elimination tree). This analysis phase also determines upper bounds on the memory usage, the floating-point operation count, and the number of nonzeros in the LU factors.

UMFPACK4 factorizes each *chain* of frontal matrices in a single working array, similar to how the unifrontal method [34] 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. For the nonsingular matrices factorized with the unsymmetric strategy, there are exactly the same number of chains as there are leaves in the supernodal column elimination tree. UMFPACK4 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 sub-matrices (referred to 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 pre-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. With UMFPACK4's unsymmetric strategy, the pivot rows are not known ahead of time as they are for the multifrontal method for symmetric positive definite matrices, however.

The pre-analysis phase also determines the worst-case size of each frontal matrix so that they can hold any candidate pivot column and any candidate pivot row. The set of candidate pivot columns for a single frontal matrix forms a single super-column. 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 row. Existing left-looking numerical factorization methods do not have any additional information. They do not keep track of the nonzero pattern of the Schur complement. In terms of reducing fill-in, they cannot decide between candidate pivot columns in the same super-column, nor can they decide between candidate pivot rows.

In contrast, the right-looking numerical factorization phase of UMFPACK4 has more information than its analysis phase. It uses this information to reorder the columns within each frontal matrix to reduce fill-in. UMFPACK4 reorders only those columns within a single super-column, so no change is made in the upper bounds found in the symbolic ordering and analysis phase. Since the number of nonzeros in each row and column are maintained (more precisely, COLMMD-style approximate degrees [42]), a pivot row can be selected based on sparsity-preserving criteria as well as numerical considerations (relaxed threshold partial pivoting).

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). When the symmetric or 2-by-2 strategies are used, the column preordering is not refined during numeric factorization. Row pivoting for sparsity and numerical accuracy is performed only if the diagonal entry is too small.

3.2 Column pre-ordering and symbolic analysis

When the unsymmetric strategy is used, the column pre-ordering is found with a slightly modified version of COLAMD [19, 20, 50]. 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]. The modified COLAMD routine used in UMFPACK4 constructs the supernodal column elimination tree, and determines the upper bound of the size of each frontal matrix. It then post-orders the tree, 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). A frontal matrix $i+1$ in the middle of a chain can have more than one child. With a column post-ordering, each chain starts as a leaf in the tree, and there are exactly as many chains in the tree as there are leaves in the tree. There can be more chains than leaves if a post-ordering of the column elimination tree is not performed. The symbolic phase determines upper bounds on the memory usage, the upper bound on the number of nonzeros in each column of $\mathbf{L}$ and each row of $\mathbf{U}$, and the upper bound on the floating-point operation count. This entire phase, including the ordering, is computed in space proportional to number of nonzeros in $\mathbf{A}$.

For the symmetric strategy, the column pre-ordering is found via a symmetric permutation of the matrix $\mathbf{A}+\mathbf{A}^\mathsf{T}$, using the approximate minimum degree algorithm [1, 2], followed by a post-ordering of the elimination tree of $\mathbf{A}+\mathbf{A}^\mathsf{T}$. Next, the analysis phase finds the column elimination tree and the frontal matrix chains in that tree. No post-ordering of the column elimination tree is computed, since it does not preserve the fill-in of the Cholesky factorization of $\mathbf{A}+\mathbf{A}^\mathsf{T}$.

The COLAMD routine available in MATLAB (version 6.0 and later) performs the column ordering, and then does a post-order of the tree via the COLETREE function (`sparsfun ('coletree', ...)`). It does not consider the size of the frontal matrices when post-ordering the tree, so the post-ordering is different than the modified version of COLAMD used in UMFPACK4. It does not do a symbolic analysis of the LU factorization. The upper bound on fill-in is identical and its ordering quality is essentially the same as the modified COLAMD routine, however.

MA38 attempts to find unifrontal chains on the fly. The post-ordering of UMFPACK4 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 Sections 4 and Sections 5.

The supernodal column elimination tree also captures the potential pivot row ordering that will be performed during numerical factorization [51]. After the column ordering is applied, suppose that the nonzero a_{ij} is the "leftmost" nonzero in row i. That is, j is the smallest column index such that a_{ij} is nonzero. Consider the frontal matrix f to which column j is assigned. This frontal matrix corresponds to one node in the supernodal column elimination tree. Column j will be factorized at this node, but row i might not be. Row i

will be first considered as a candidate row at this node f. If it is not selected here, it will be considered at the parent of node f, and so on. Row i will be chosen as a pivot row at some node on the path from f to the root of the supernodal column elimination tree.

3.3 Numerical factorization

This section presents a detailed overview of the numerical factorization method used in UMFPACK4. A small example is given in the following section.

The numerical factorization phase starts by allocating several temporary data structures. 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* [17] 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 matrix 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 for an n-by-n matrix $\mathbf{A}$; 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, 32 by default). The frontal matrix holds up to n_B prior pivot rows and columns. When the updates for these pivots are computed, the corresponding columns of $\mathbf{L}$ and $\mathbf{U}$ are saved in a separate data structure for the LU factors.

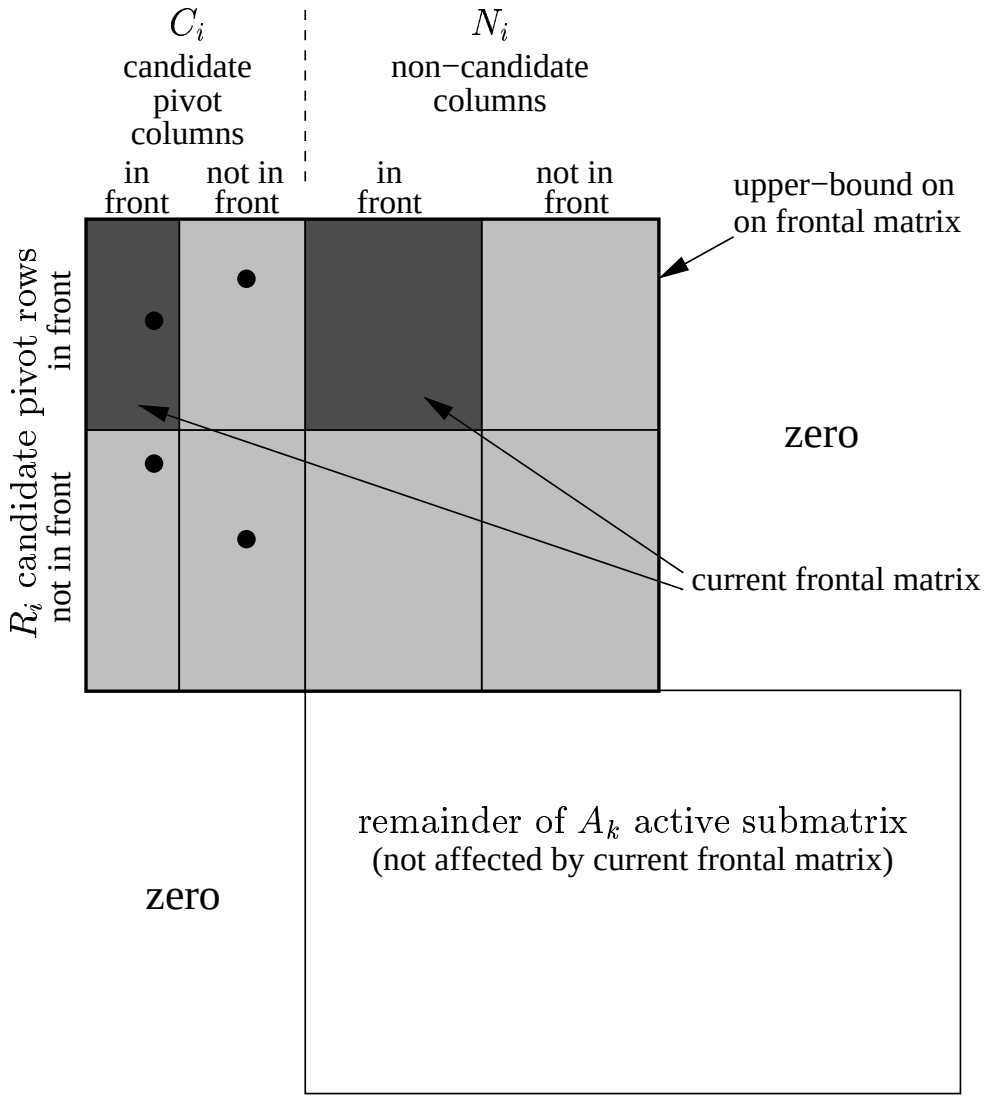
Figure 1 shows the $(n-k)$-by-$(n-k)$ active submatrix $\mathbf{A}_k$ being factorized, and the portion of that matrix that may be held in the work array (the shaded region, which is the upper-bound of the current frontal matrix). The current frontal matrix occupies only part of the work array, shown as the two dark shaded regions. 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). These rows and columns may not appear in the actual frontal matrix, since they are determined in the symbolic analysis phase. That phase finds an upper bound on the size of each frontal matrix, which is $|R_i|$-by-$|C_i|+|N_i|$. This will be further illustrated in an example in the next section.

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 level-3 BLAS operations: a

Algorithm 1: UMFPACK4 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
                apply pending updates to just the kth pivot column
                if too many zero entries in new frontal matrix (or new LU part)
(*)                 apply all pending updates
                    copy pivot rows and columns into LU data structure
                end if
                if too many zero entries in new frontal matrix
(**)                create new contribution block and place on stack
                    start a new frontal matrix
                else
(***)               extend the frontal matrix
                end if
                assemble contribution blocks into current frontal matrix
                scale pivot column
                if # pivots in current frontal matrix ≥ n_B
                    apply all pending updates
                    copy pivot rows and columns into LU data structure
                end if
            end for |C_i| iterations
        end for each frontal matrix in the chain
(****)  apply all pending updates
        copy pivot rows and columns into LU data structure
        create new contribution block and place on stack
    end for each chain
```

Figure 1: The active submatrix and current frontal matrix



dense lower-triangular solve (DTRSM) to compute the rows of **U**, and dense matrix-matrix multiply (DGEMM) to compute the Schur complement.

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 existing left-looking methods. This step is where the column pre-ordering is refined, and where the row ordering is determined. 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. Among those pivot entries that are numerically acceptable, the candidate row of least approximate degree in the current front and the row of least approximate degree not in the current front are found. The default numerical test requires a candidate pivot entry to have an absolute value greater than or equal to 0.1 times the absolute value of the largest entry in the candidate pivot column. For the symmetric strategy, a relative threshold of 0.001 is used for the diagonal entry, and 0.1 is used for all off-diagonal pivot candidates. The row and column degrees are not exact; COLMMD-style approximate degrees are used, which are simply the sum of the sizes of the contribution 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. The exact degrees of these candidate pivot columns and pivot rows are then computed, and any pending updates to the pivot column candidate in the front are applied (via level-2 BLAS operations, DTRSV and DGEMV). The metric used to select among these four candidates is a form of approximate minimum fill-in [55, 56]; the pivot entry that causes the least growth in the size of the actual frontal matrix is chosen as the kth pivot. If there is a tie, preference is given to pivot rows and columns that are already in the current frontal matrix.

Each of the one to four candidate pivots resides in the upper bound frontal matrix determined during the symbolic analysis. The column refinement is only done within the candidate columns C_i. As far as the symbolic analysis, each of these columns in C_i has identical upper bound nonzero pattern, and thus they can be rearranged in any order within this front during numerical factorization. Likewise, all of the candidate pivots reside in the candidate pivot rows that were considered during symbolic analysis. This local pivot search strategy of considering one to four pivot candidates, in at most two columns and four rows, is a trade-off between quality and efficiency. We already know a good upper bound on the fill-in, regardless of how what local strategy is used. For a small amount of additional search, based on the known approximate pivot row and column degrees, a better ordering can be typically be obtained.

To summarize, a mix of pivot strategies is used in UMFPACK4. The method starts with a good column pre-ordering, from COLAMD or AMD. Each of these methods uses a high-quality approximate minimum degree metric. Existing left-looking methods stop here. UMFPACK4 refines this choice by evaluating up to four pivot candidates within the existing frontal matrix, which are considered identical as far as the column-preordering is

concerned. These four are selected based on a low-quality but cheap-to-compute approximate minimum degree strategy (COLMMD-style). Then, with these four candidates, exact degrees are computed. With these exact degrees and the size of the current frontal matrix, an approximation on upper bound on fill-in is computed, and the best of the four is selected based on this metric.

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, or if the next pivot candidate is not in the current frontal matrix. The updates are also applied, and the current contribution block stacked, if too many zero entries would be included in the contribution block; in this case a new frontal matrix is started. The latter step also occurs at the end of a chain. The end of each chain is known from the analysis phase.

The test for "too many" zero entries is controlled by a *relaxed amalgamation* heuristic similar to those used in other multifrontal methods [30, 32]. The heuristic tries to balance the trade-off between extra floating-point operations and efficient use of dense matrix kernels. If no extra zero entries are allowed, then there are more dense matrix multiplications with smaller inner dimension of the two matrices being multiplied. If extra zeros are allowed, then fewer dense matrix multiplications are performed, each with a higher inner dimension[1]. The result is faster performance in terms of floating-point operations per second, but more floating-point operations are being performed. The heuristic has been optimized by testing on a wide range of sparse matrices from real applications. The performance of UMFPACK4 is fairly insensitive to the parameter settings as long as they are within a reasonable range.

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 UMFPACK4; its role is replaced by the simpler supernodal column elimination tree computed in the analysis phase. The DAG does describe the data flow between frontal matrices in UMFPACK4, however. The pivot row and column remain in the current frontal matrix as pending updates, unless sufficient work has accumulated. Pivot rows and columns are copied into a separate compressed-index data structure for $\mathbf{L}$ and $\mathbf{U}$ and removed from the frontal matrix only when their pending updates have been applied to the contribution block.

The data structure for $\mathbf{L}$ and $\mathbf{U}$ is independent of the relaxed amalgamation heuristic, or how many zeros are included in the frontal matrices. The nonzero pattern of column k of $\mathbf{L}$ is stored in one of two ways: it can be derived from the $k-1$st column of $\mathbf{L}$, or it can be stored independently. The method that leads to the smallest memory usage for the kth column is selected. This is a local greedy heuristic since the $k+1$st column is not yet known.

Consider the $k-1$st column of $\mathbf{L}$ (for $k > 1$). The kth column will often have very similar nonzero pattern as this prior column, minus the kth pivot row index itself. If we assume that the $k-1$st column of $\mathbf{L}$ pattern is a subset of the kth column, we can store the

[1] A matrix multiply $C = C + A * B$ has an inner dimension, or "rank," corresponding to the number of columns of A and the number of rows of B.

location of the kth pivot row index in the prior pattern (if it exists) and the nonzero indices of rows in the kth column but not in the $k-1$st column. Extra zero entries will appear if the subset relationship does not hold, but this still may save space as compared to storing the entire pattern of the kth column of $\mathbf{L}$. Columns $k-1$ and k do not need to be part of the same frontal matrix nor even part of the same chain to exploit this strategy (although if this strategy is exploited it is likely that they are). A similar strategy is used to store each row of $\mathbf{U}$. The storage scheme is selected independent of both the frontal matrix relationships and the supernodal column elimination tree.

3.4 Example numerical factorization

A small example matrix is shown in Figure 2. For simplicity, no pivoting is done in either the column pre-ordering and symbolic analysis phase or in the numerical factorization phase. The example shows the nonzero pattern of an 8-by-8 sparse matrix $\mathbf{A}$, its LU factorization, and its supernodal column elimination tree. The nodes of the supernodal column elimination tree are labeled with the columns that are pivotal at each node. The example continues in Figure 3, which shows the nonzero pattern of each frontal matrix in place of its corresponding node in the supernodal column elimination tree. There are three frontal matrix chains in the graph: node 1, nodes 2, 3, and 4, and the two nodes $\{5, 6\}$ and $\{7, 8\}$. Nonzero values are shown as small black circles. A box represents the upper bound nonzero pattern of each frontal matrix, with the sides of each box labeled with row and column indices. Figure 4 shows the upper bound pattern of the LU factors. At each step of symbolic factorization, the upper bound on the pivot row pattern is the union of all candidate pivot rows. This pivot row then causes fill-in so that all candidate pivot rows take on the same nonzero pattern. Figure 5 shows the actual LU factorization, assuming no pivoting during numerical factorization.

Numerical factorization proceeds as follows. To simplify this example, no relaxed amalgamation is permitted (no extra zero entries are allowed in the frontal matrices). UMFPACK4 does perform relaxed amalgamation, however.

1. Original entries from the matrix $\mathbf{A}$ are assembled into the 2-by-3 frontal matrix 1. It is factorized, and the 1-by-2 contribution block is placed on the stack (see the line marked (****) in Algorithm 1). This node is the end of a chain.

2. A working array of size 4-by-5 is sufficient for the next chain (nodes 2, 3, and 4). For any frontal matrix in this chain, the array can hold the frontal matrix and all of the prior pivot rows and columns from this chain. The worst case size occurs at node 4, where the frontal matrix is at most 2-by-3, and at most 2 prior pivot rows and columns need to be held in the array. No more than $n_B = 32$ prior pivots are permitted in the current working array, but this parameter does not have any effect in this small example. Frontal matrix 2 is assembled into a 2-by-3 array inside the 4-by-5 working array. The updates for the second pivot are not yet applied, and the second contribution block is not yet stacked. Note that column 8 was determined to be a non-candidate column in N_2 that might appear in the second frontal matrix. It would have appeared if row 7 were picked as the second pivot row, instead of row 2. The upper bound on the second frontal matrix is 2-by-4, but the actual size found during numerical factorization is 2-by-3.

Figure 2: Example numerical factorization: matrices and tree

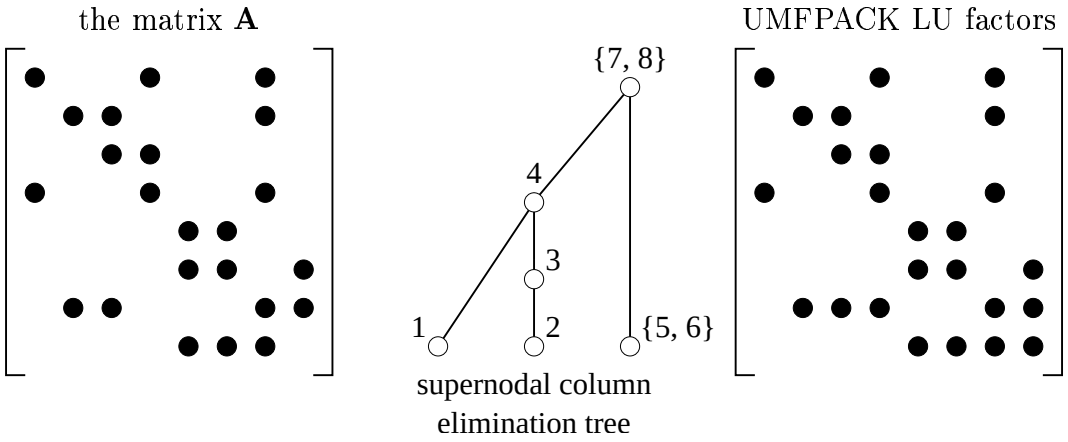


Figure 3: Example numerical factorization: details of frontal matrices

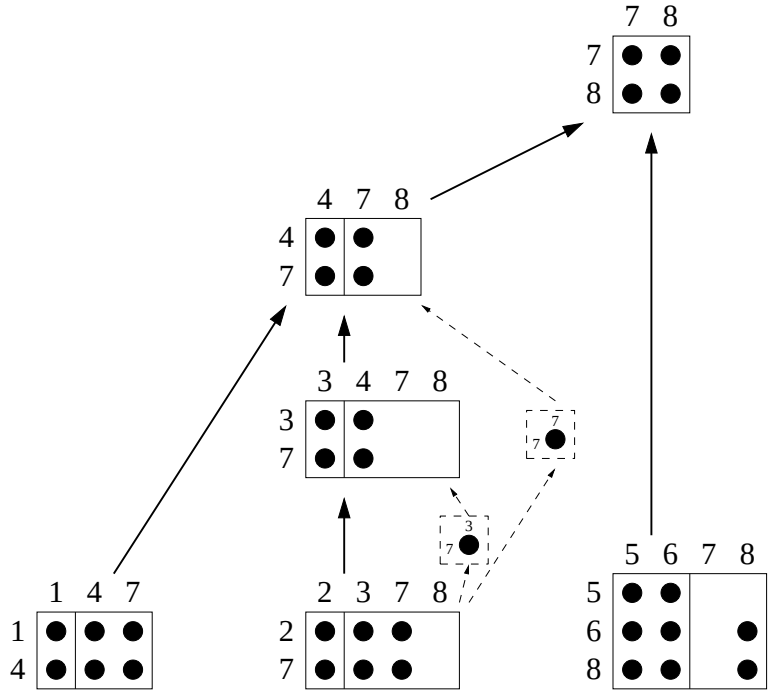


Figure 4: Upper bound pattern of LU factorization

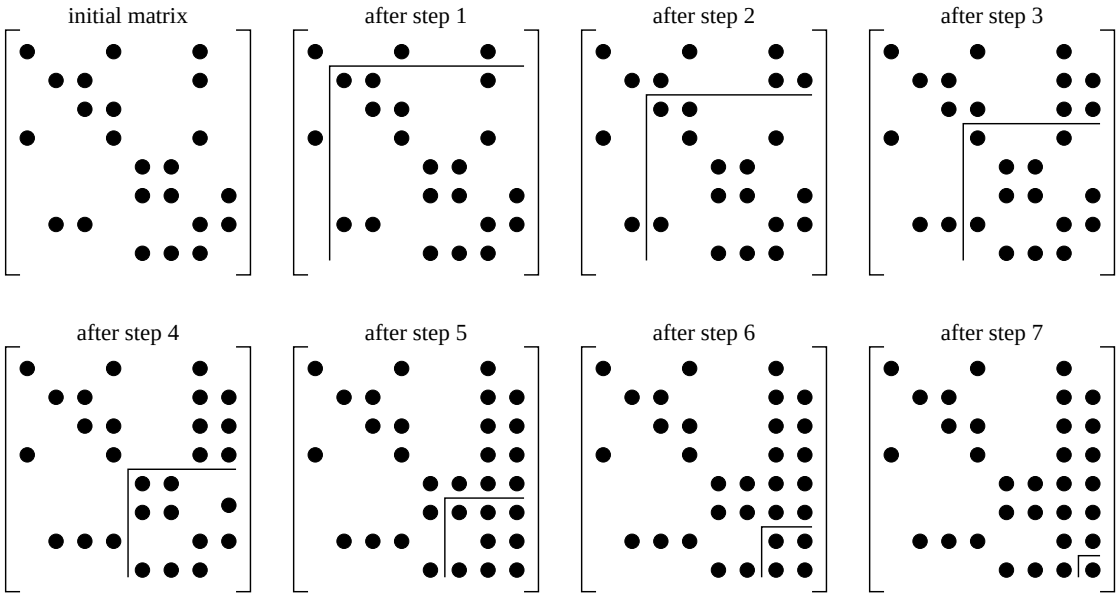
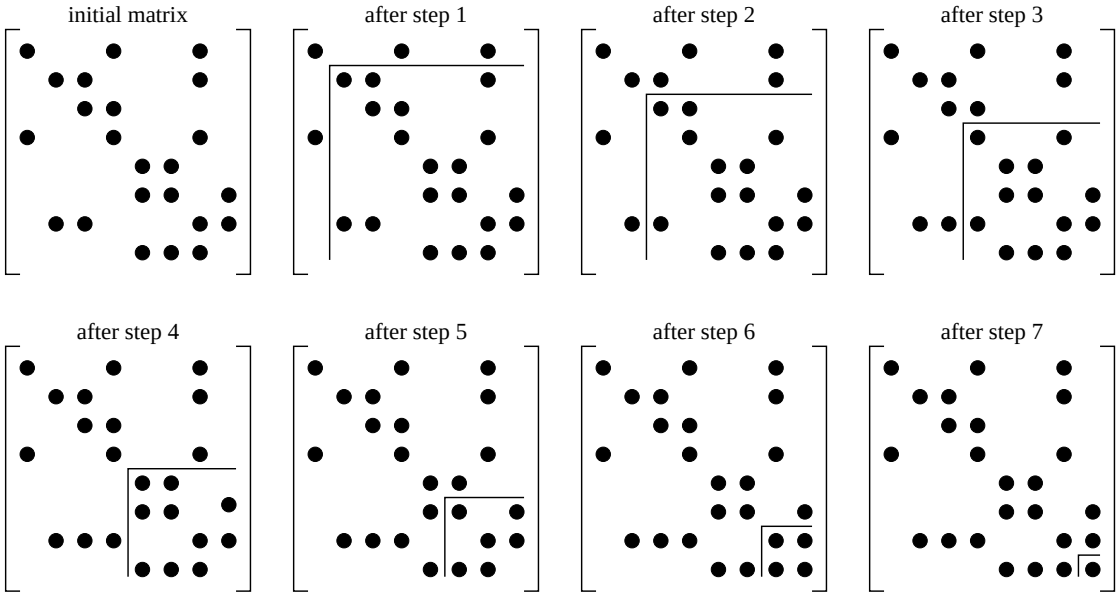


Figure 5: Actual LU factorization



3. The factorization of frontal matrix 3 starts with the 2-by-3 frontal matrix from the prior pivot. Extending the second frontal matrix to include the third pivot would create zero entries (line (*) in Algorithm 1) so the pending updates from pivot 2 are applied, and the 1-by-2 contribution block is placed on the stack (line (**)). The current working array is now empty, and the third frontal matrix is constructed, of size 2-by-2. The entry $a_{73}^{(2)}$ is assembled from the prior contribution block into the current front. Node 2 is a *Uson* of its *Uparent* node 3 in the DAG, because one or more columns of node 2 can be completely removed from the contribution block of node 2 and assembled into node 3, but not every column can be assembled [46]. The frontal matrix at node 2 makes a contribution to the pivot column of node 3, but not to the pivot row. Node 2 is a *Uson* of node 3 because u_{23} is nonzero and l_{32} is zero. All other edges in the DAG are between *LUsons* and *LUparents*. The updates for the third pivot are not yet applied, and the third contribution block (the 1-by-1 matrix holding just the entry $a_{74}^{(2)}$) is not yet stacked.

4. The fourth step of factorization starts with the prior 2-by-2 frontal matrix in the current working array. Since the fourth pivot is not in the current front, the pending update from the third pivot is applied (line (*)). The remaining 1-by-1 contribution block is extended to hold the current 2-by-2 frontal matrix. Since this is the end of the chain, the pending updates (from the fourth pivot) are applied, and the remaining 1-by-1 entry $a_{77}^{(4)}$ is stacked as the fourth contribution block (line (****)). Note that the third contribution block was never stacked, but was assembled in place into the fourth frontal matrix. Node 3 is an LUson of node 4, because its entire contribution block can be assembled into node 4, even though l_{43} is zero.

5. The fifth frontal matrix is used to factorize columns 5 and 6. First, a 3-by-2 frontal matrix (columns 5 and 6, and rows 5, 6, and 8) is constructed in a 4-by-4 working array. If the sixth pivot row and column are included in this frontal matrix prior to applying the pending update from pivot row and column 5, then a single zero entry would be present (u_{58}) in the frontal matrix. A single matrix-matrix multiply could be used to apply the pending updates for steps 5 and 6, but this would lead to extra computations with this zero entry. Since no explicit zeros are allowed in this small example, the update for the fifth pivot row and column are applied. No contribution block is stacked, however. The frontal matrix is then extended (see line (***)) and the sixth pivot row and column is constructed in a 2-by-2 frontal matrix. The sixth contribution block is not stacked, and the pending updates for the sixth pivot is not yet applied.

6. Since the seventh pivot entry $a_{77}^{(6)}$ does not reside in the current frontal matrix, the pending update for the sixth pivot row and column is applied (line (*)). The frontal matrix is then extended in size from 1-by-1 to 2-by-2 (line (***)), and the seventh pivot row and column are assembled. The eighth pivot row and column are then found. The chain ends at this frontal matrix, and the factorization is complete.

The example would change with fully relaxed amalgamation. Only two contribution blocks would be stacked, one for the first frontal matrix, and one for the fourth frontal

matrix. A rank-2 update would be applied for the $\{5, 6\}$ frontal matrix, rather than two separate rank-1 updates.

4 Experimental results

In this section the experimental design and its results are presented. Qualitative observations about these results are made in the subsequent section.

The new method, UMFPACK v4.1, is compared with LU, SuperLU, MA38, and the latest "unsymmetric" version of MA41 [7], referred to here as MA41u. Each method was tested on a Dell Latitude C840 with a 2 GHz Pentium 4M processor, 1 GB of main memory, and 512 KB of cache. The BLAS by Goto and Van de Geijn was used for all methods [44]. This BLAS increases the performance of UMFPACK4 by about 50% for large sparse matrices on this computer, as compared with the ATLAS 3.4.1 BLAS [58]. Gilbert and Peierls' sparse LU was used within MATLAB Version 6.5. It does not make use of the BLAS. MATLAB 6.5 includes UMFPACK v4.0, which does not have the symmetric or 2-by-2 strategies, and takes less advantage of level-3 BLAS. It is not included in these comparisons since the two versions are otherwise very similar. UMFPACK v4.1 is nearly always as fast or faster than v4.0, and uses the same amount or less memory. The difference is quite large for symmetric-patterned matrices, such as those arising from finite-element problems.

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 [14, 24, 27] 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. For LU with the unsymmetric strategy, we used the COLAMD pre-ordering, and scaled the rows of $\mathbf{A}$ the same way that UMFPACK4 performs its default scaling. Row scaling typically leads to a sparser and more accurate LU factorization. The unsymmetric strategy is as follows:

```
n = size (A, 1) ;
q = colamd (A) ;
C = A (:,q) ;
R = spdiags (full (sum (abs (C), 2)), 0, n, n) ;
[L,U,P] = lu (R\C) ;
```

In collaboration with Sherry Li, we introduced the symmetric strategy into SuperLU, using AMD on $\mathbf{A} + \mathbf{A}^{\mathsf{T}}$ as the pre-ordering, followed by a post-ordering of the elimination tree of $\mathbf{A} + \mathbf{A}^{\mathsf{T}}$, rather than a post-ordering of the column elimination tree. Results from both the original SuperLU (which is better for unsymmetric-patterned matrices) and the modified SuperLU (which is better for matrices with mostly symmetric nonzero pattern) are included here. We also applied the same strategy to Gilbert and Peierls' LU. We did not evaluate an automatic selection strategy for SuperLU or LU. Thus, for any given matrix, the strategy that gives the best result for SuperLU or LU is presented. The symmetric strategy applied to LU is as follows, where `amd (A)` computes the AMD ordering on $\mathbf{A} + \mathbf{A}^{\mathsf{T}}$ and then post-orders the elimination tree:

```
n = size (A, 1) ;
q = amd (A) ;
C = A (q,q) ;
R = spdiags (full (sum (abs (C), 2)), 0, n, n) ;
[L,U,P] = lu (R\C, 0.001) ;
```

The March 2003 release of the UF sparse matrix collection [15] includes 389 real, square, unsymmetric sparse matrices. Fifteen singular matrices were excluded. Ten large matrices which could not be factorized on this computer were also excluded[2]. Hollinger's economic modeling matrices were tested, and then excluded from the comparisons. They demonstrate an extreme sensitivity to minor tie-breaking decisions in COLAMD and AMD. Making a minor change in how the degree lists are initialized in AMD, for example, cut UMFPACK's total run time in half for one matrix, and increased it by a factor of four for a very similar matrix in the set. The only other matrix in the collection with such sensitivity is the FINAN512 matrix from Mulvey and Rothberg. It is also an economic modeling matrix. Berger et al. [11] have shown that the FINAN512 matrix requires a specialized tree-dissection ordering, and that the minimum degree algorithm (either approximate or exact) finds a very poor ordering. Since all of the methods here were tested with variants of the minimum degree algorithm, it is likely that none of them found good orderings for Hollinger's matrices. They are thus excluded from the comparisons (the results are not unlike the results for unsymmetric-patterned matrices presented below, however).

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 CPU time is indirectly related to the floating-point operation count, as discussed below.

- 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. A method may perform extra floating-point operations to get better performance in the dense matrix kernels. The time it takes to perform a single floating-point operation can vary widely, even within the same code. The best measure for computational efficiency is thus the actual CPU time, not the floating-point operation count. Also, some of the methods compared here do not return an actual floating-point operation count.

- The total memory usage, excluding the space required to hold $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{b}$. This statistic includes all memory used by the method, not just the amount of memory

[2] APPU, LI, CAGE11 through CAGE15, PRE2, XENON2, and TORSO3.

required to hold the LU factors. For example, it includes the space for the frontal matrix stack for the multifrontal methods. LU's memory usage when $\mathbf{A}$ is real and square is 12 bytes per nonzero in $\mathbf{L}+\mathbf{U}$ plus $53n$ bytes for $\mathbf{P}$, $\mathbf{Q}$, the rest of the data structures for $\mathbf{L}$ and $\mathbf{U}$, and temporary work space [40]. Any method requires eight bytes to hold the numerical value of each entry $\mathbf{L}$ or $\mathbf{U}$ itself. An integer row or column index takes four bytes. The bytes-per-entry ratio cannot be less than 8, but it can be less than 12 since most methods do not require a companion integer index for each floating-point value in $\mathbf{L}$ and $\mathbf{U}$. The total amount of memory required by each method is thus a function of two things: the ordering quality, and the overhead required by the data structures.

- 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. This is a measure of ordering quality, and only indirectly a measure of memory efficiency. Each method uses a different storage scheme, and some store explicit zero entries to either save space (the pattern is simpler) or to save time (dense matrix kernels can be used in the forward and back solves).

- The norm of the residual, $||\mathbf{Ax}-\mathbf{b}||_\infty$. All methods except LU use iterative refinement with sparse backward error [8] in their forward/backward solve step. UMFPACK4 was found to be just as accurate as LU on the matrices in this test set, or more accurate in many cases.

The test set is split into three sets of matrices, depending on which of the three strategies UMFPACK automatically selected for each matrix (unsymmetric, 2-by-2, or symmetric). Tables 1 through 3 list the largest matrices in the three sets. The "size" of a matrix in this sense reflects the smallest floating-point operation count for each of the five methods reported here. The matrices are sorted in increasing size. Matrices for which all methods report nearly identical results as other matrices in the tables were excluded[3]. Each table lists the matrix group, name, dimension, number of nonzeros, and the symmetry of the nonzero pattern. 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.

Most matrices in the "unsymmetric" set include matrices from chemical process simulation, frequency-domain circuit simulation, and computational fluid dynamics. Nearly all matrices in the "2-by-2" set are computational fluid dynamics problems with fluid-structure interaction. Some include chemistry or heat transfer. Most matrices in the "symmetric" set come from computational fluid dynamics, structural analysis, and electromagnetic problems.

Results for these matrices are given in Tables 4 through 8, which list the run time in seconds (including the symbolic analysis and ordering phase), the canonical floating-point operation count (in millions), the number of nonzeros in $\mathbf{L}+\mathbf{U}$ (in thousands; this count excludes the unit diagonal of $\mathbf{L}$), the total amount of memory used (in megabytes), and the

[3] HEART3, PSMIGR_3, VENKAT25, VENKAT50, AND WANG3

Table 1: Matrix statistics: unsymmetric set

Group	Name	n	nonzeros (in 1000's)	sym.	description
Mallya	lhr14c	14270	307.9	0.007	light hydrocarbon recovery
Mallya	lhr17c	17576	382.0	0.002	light hydrocarbon recovery
AT&T	onetone2	36057	222.6	0.116	harmonic balance method
Graham	graham1	9035	335.5	0.718	Navier-Stokes, finite-element
Mallya	lhr34c	35152	764.0	0.002	light hydrocarbon recovery
Shen	e40r0100	17281	553.6	0.308	
Mallya	lhr71c	70304	1528.1	0.002	light hydrocarbon recovery
FIDAP	ex40	7740	456.2	1.000	Navier-Stokes, finite-element (3D)
AT&T	onetone1	36057	335.6	0.076	harmonic balance method
Vavasis	av41092	41092	1683.9	0.001	unstructured finite-element
AT&T	twotone	120750	1206.3	0.246	harmonic balance method
HB	psmigr_2	3140	540.0	0.479	population migration
Simon	bbmat	38744	1771.7	0.529	2D airfoil, turbulence

Table 2: Matrix statistics: 2-by-2 set

Group	Name	n	nonzeros (in 1000's)	sym.	description
Goodwin	goodwin	7320	324.8	0.635	fluid mechanics, finite-element
Averous	epb2	25228	175.0	0.670	plate-fin heat exchanger
Garon	garon2	13535	373.2	0.999	2D finite-element, Navier-Stokes
Goodwin	rim	22560	1015.0	0.639	fluid mechanics, finite-element
Norris	heart2	2339	680.3	1.000	quasi-static FEM, human heart
Averous	epb3	84617	463.6	0.667	plate-fin heat exchanger
Bova	rma10	46835	2329.1	1.000	3D model of Charleston Harbor
Norris	heart1	3557	1385.3	1.000	quasi-static FEM, human heart
HB	psmigr_1	3140	543.2	0.479	population migration

Table 3: Matrix statistics: symmetric set

Group	Name	n	nonzeros (in 1000's)	sym.	description
Norris	torso2	115967	1033.5	0.992	2D human torso, electro-phys., finite-diff.
Simon	olafu	16146	1015.2	1.000	structure problem
Simon	venkat01	62424	1717.8	1.000	unstructured 2D Euler problem
Bai	af23560	23560	460.6	0.944	airfoil
Simon	raefsky3	21200	1488.8	1.000	fluid-structure, turbulence
Zhao	Zhao1	33861	166.5	0.922	electromagnetics
Zhao	Zhao2	33861	166.5	0.922	electromagnetics
FIDAP	ex11	16614	1096.9	1.000	3D fluid flow, cylinder and plate
Simon	raefsky4	19779	1316.8	1.000	container buckling problem
Wang	wang4	26068	177.2	1.000	3D MOSFET semiconductor
Ronis	xenon1	48600	1181.1	1.000	zeolite, sodalite crystals
vanHeukelum	cage10	11397	150.6	1.000	DNA electrophoresis
Norris	stomach	213360	3021.6	0.848	3D electro-physical, human duodenum

total memory used (in bytes) divided by the number of nonzero entries in $\mathbf{L} + \mathbf{U}$. Results within 25% of the best result for a particular matrix are shown in bold. The last part of each table lists the median ratios relative to UMFPACK4, and the median number of bytes per nonzero entry. These results are also plotted in Figures 6 through 8. Each of the three logarithmic plots depicts the relative results of one method as compared to UMFPACK4. Each circle on the plot is a single matrix listed in one of the tables Tables 4 through 8. A circle in the upper right quadrant depicts a matrix for which the specific method requires more time and memory than UMFPACK4. The four dashed lines represent relative results similar to UMFPACK4 (0.8 and 1.25). The solid lines represent the median relative results.

For matrices in the unsymmetric set, SuperLU and LU typically used the unsymmetric strategy. The exception for both methods was the PSMIGR_2 matrix. SuperLU and LU typically used the symmetric strategy for matrices in the two other tables, with the exception of HEART2 for LU, ZHAO2 for SuperLU, and RIM for both SuperLU and LU. LU and MA38 ran out of memory for the STOMACH matrix.

The full results for all three of UMFPACK4's strategies is not reported here, since the automatic strategy selection nearly always selects the best strategy for these matrices. The worst-case exception is the RMA10 matrix. LU and SuperLU use the symmetric strategy for this matrix, since they do not have the 2-by-2 strategy and the unsymmetric strategy leads to too much fill-in. UMFPACK4 selects its 2-by-2 strategy, but the symmetric strategy is better. UMFPACK4 with its symmetric strategy finds an ordering with the identical fill-in and floating point operation count as LU, as reported in Table 6. The memory usage drops to 75.8 MB (9.1 bytes per nonzero entry in $\mathbf{L}$ and $\mathbf{U}$). The run time is 3.3 seconds, which is the same as SuperLU.

The peak performance of each method for these matrices is 214 Mflops for LU, 690 Mflops for SuperLU, 1.65 Gflops for UMFPACK4, 1.21 Gflops for MA38, and 1.96 Gflops for MA41u. The theoretical peak performance of the computer used in this experiment is 4 Gflops. Goto and van de Geijn's DGEMM routine has a peak performance of 3.3 Gflops. This is obtained on relatively small dense matrices, of the size that are typically used in multifrontal methods (3.2 Gflops for $C = C + A * B$, where A is 100-by-32 and B is 32-by-100, for example).

5 Experimental comparisons

The above 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, UMFPACK4, and MA38 do not. UMFPACK4 is based on the supernodal column elimination tree, which could be used to guide a parallel version of UMFPACK, however. 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 [3, 46, 47]. With this caveat in mind, a few observations on the results can be made.

The left-looking methods LU and SuperLU typically find nearly identical orderings because they use the same pre-ordering and pivoting strategy. In terms of fill-in and floating-point operation count, UMFPACK4 behaves most similarly to the left-looking methods. This is to be expected, since all three methods use the same column pre-ordering, are based on

Table 4: Results for unsymmetric set

Matrix		LU	SuperLU	UMFPACK4	MA38	MA41u
MALLYA	time:	1.4	**0.6**	**0.6**	0.8	0.9
LHR14C	flop:	83.3	85.1	**58.1**	85.1	82.1
	nz LU:	**1372.1**	**1381.3**	**1144.7**	**1309.6**	**1331.2**
	mem:	16.4	17.6	**11.4**	15.9	25.3
	mem/nz:	**12.6**	13.3	**10.5**	**12.8**	20.0
MALLYA	time:	1.8	**0.7**	**0.7**	1.1	1.4
LHR17C	flop:	105.1	110.3	**79.5**	130.6	123.6
	nz LU:	**1666.6**	**1703.4**	**1427.0**	**1700.8**	**1731.0**
	mem:	20.0	21.5	**14.2**	20.8	33.2
	mem/nz:	**12.6**	13.3	**10.4**	**12.8**	20.1
AT&T	time:	1.3	**0.7**	**0.7**	2.3	**0.7**
ONETONE2	flop:	**93.8**	**94.0**	**87.0**	705.9	245.3
	nz LU:	**1032.1**	**1049.8**	**903.4**	1914.3	1379.1
	mem:	**13.6**	18.8	**12.0**	31.7	29.0
	mem/nz:	**13.9**	18.8	**13.9**	17.4	22.0
GRAHAM	time:	8.1	3.3	2.0	5.9	**0.2**
GRAHAM1	flop:	1151.0	1395.5	549.4	2567.4	**109.3**
	nz LU:	4568.3	4936.3	2858.8	5406.8	**1091.2**
	mem:	52.7	53.9	31.8	60.5	**17.3**
	mem/nz:	**12.1**	**11.4**	**11.7**	**11.7**	16.6
MALLYA	time:	4.7	**1.7**	**1.5**	8.8	10.2
LHR34C	flop:	221.9	238.0	**167.8**	393.4	231.2
	nz LU:	**3421.0**	**3529.6**	**2915.8**	3871.7	**3400.6**
	mem:	40.9	44.4	**28.5**	44.7	64.5
	mem/nz:	**12.5**	13.2	**10.2**	**12.1**	19.9
SHEN	time:	9.0	2.8	1.3	5.5	**0.9**
E40R0100	flop:	1236.9	1029.3	518.6	2589.0	**284.9**
	nz LU:	5614.4	5080.7	3769.2	6453.5	**2186.6**
	mem:	65.1	54.9	**37.1**	71.1	**30.3**
	mem/nz:	**12.2**	**11.3**	**10.3**	**11.6**	14.5
MALLYA	time:	9.4	**3.5**	**3.0**	14.1	19.8
LHR71C	flop:	469.3	498.4	**343.0**	823.9	501.9
	nz LU:	**6953.4**	**7165.1**	**5828.4**	7933.8	**6981.1**
	mem:	83.1	89.9	**56.4**	89.9	128.9
	mem/nz:	**12.5**	13.2	**10.1**	**11.9**	19.4

Table 5: Results for unsymmetric set, continued

Matrix		LU	SuperLU	UMFPACK4	MA38	MA41u
FIDAP	time:	7.5	2.5	**1.2**	10.8	**1.0**
EX40	flop:	1053.2	856.9	**419.4**	6062.6	1140.7
	nz LU:	4249.6	3723.5	**2553.5**	8498.4	3553.9
	mem:	49.0	40.0	**22.9**	115.8	38.7
	mem/nz:	12.1	**11.3**	**9.4**	14.3	**11.4**
AT&T	time:	17.2	9.1	2.5	5.9	**1.8**
ONETONE1	flop:	2561.8	2555.7	1951.3	2270.4	**1140.3**
	nz LU:	4466.9	4476.2	**3637.4**	4269.7	**2947.7**
	mem:	**52.9**	**54.1**	**46.5**	69.5	64.9
	mem/nz:	**12.4**	**12.7**	**13.4**	17.1	23.1
VAVASIS	time:	318.1	145.5	43.3	56.0	**11.4**
AV41092	flop:	65520.2	74016.0	28126.0	42612.7	**3264.1**
	nz LU:	38670.4	42795.4	33862.5	33418.1	**8796.7**
	mem:	444.6	436.7	340.2	358.8	**161.3**
	mem/nz:	**12.1**	**10.7**	**10.5**	**11.3**	19.2
AT&T	time:	41.1	40.5	**5.2**	34.6	13.2
TWOTONE	flop:	5746.1	5798.6	**3633.4**	17395.3	8307.9
	nz LU:	13055.9	13551.0	**7048.5**	15573.3	10921.2
	mem:	155.5	223.0	**91.5**	223.9	315.0
	mem/nz:	**12.5**	17.3	**13.6**	**15.1**	30.2
HB	time:	56.6	30.2	**8.6**	12.3	**7.4**
PSMIGR_2	flop:	11877.0	13079.3	10256.1	**7416.7**	9895.2
	nz LU:	7518.0	7980.8	6718.3	**5371.6**	**6394.3**
	mem:	**86.2**	**80.9**	**100.7**	160.4	194.1
	mem/nz:	**12.0**	**10.6**	15.7	31.3	31.8
SIMON	time:	204.8	81.6	**32.6**	242.7	**26.8**
BBMAT	flop:	**40489.4**	**41479.3**	**33325.7**	247934.7	**38714.1**
	nz LU:	**46729.8**	**47302.6**	**41911.9**	110391.9	**43670.3**
	mem:	536.7	489.6	**356.4**	1196.0	455.0
	mem/nz:	12.0	**10.9**	**8.9**	11.4	**10.9**
median ratios	time:	6.09	2.07	1.00	3.33	0.86
	flop:	1.37	1.45	1.00	2.40	1.38
	nz LU:	1.19	1.23	1.00	1.36	1.16
	mem:	1.44	1.54	1.00	1.60	1.93
median	mem/nz:	12.43	12.66	10.46	12.76	19.87

Table 6: Results for 2-by-2 set

Matrix		LU	SuperLU	UMFPACK4	MA38	MA41u
GOODWIN	time:	12.0	6.4	0.6	2.0	**0.2**
GOODWIN	flop:	2177.9	2466.9	**131.9**	843.1	**120.3**
	nz LU:	4281.1	4776.8	**1104.8**	3054.3	**1048.5**
	mem:	49.4	49.7	**11.0**	32.9	16.5
	mem/nz:	**12.1**	**10.9**	**10.4**	**11.3**	16.5
AVEROUS	time:	2.4	0.9	1.0	3.7	**0.4**
EPB2	flop:	**277.7**	**277.7**	**302.9**	1671.5	**304.8**
	nz LU:	**1905.4**	**1905.4**	**1917.7**	4503.3	**2002.2**
	mem:	**23.1**	**23.7**	**21.5**	50.7	29.2
	mem/nz:	**12.7**	**13.0**	**11.7**	**11.8**	15.3
GARON	time:	4.1	1.6	1.1	1.5	**0.4**
GARON2	flop:	543.7	572.6	504.5	586.0	**292.2**
	nz LU:	2842.1	2931.4	2737.0	2878.2	**2103.5**
	mem:	33.2	30.9	**24.4**	34.5	**26.0**
	mem/nz:	12.3	**11.1**	**9.3**	12.6	13.0
GOODWIN	time:	30.8	13.6	2.6	13.2	**0.8**
RIM	flop:	4732.2	6204.4	911.3	6432.6	**420.3**
	nz LU:	15219.4	17688.1	4910.0	14953.1	**3423.9**
	mem:	175.3	190.6	**45.2**	151.7	**51.0**
	mem/nz:	12.1	**11.3**	**9.7**	**10.6**	15.6
NORRIS	time:	6.4	2.0	**0.8**	2.9	**0.8**
HEART2	flop:	1138.1	1149.3	**430.2**	1589.7	746.6
	nz LU:	2196.4	2166.9	**1329.1**	2679.5	1717.0
	mem:	25.3	22.8	**17.8**	41.6	30.4
	mem/nz:	**12.1**	**11.0**	14.1	16.3	18.6
AVEROUS	time:	5.0	2.3	2.6	16.0	**1.2**
EPB3	flop:	**562.6**	**562.6**	**562.6**	7721.9	1011.2
	nz LU:	**4401.0**	**4401.0**	**4401.0**	15759.0	5797.1
	mem:	**54.6**	**60.5**	**48.5**	178.1	78.9
	mem/nz:	**13.0**	**14.4**	**11.5**	**11.8**	**14.3**
BOVA	time:	12.2	3.3	4.2	27.7	**1.9**
RMA10	flop:	**1416.5**	**1417.2**	1961.3	18038.9	**1465.8**
	nz LU:	**8739.9**	**8750.6**	**10477.7**	28362.4	**8918.9**
	mem:	**102.4**	**92.7**	**90.0**	334.3	116.8
	mem/nz:	12.3	**11.1**	**9.0**	12.4	13.7
NORRIS	time:	24.7	7.5	**2.1**	5.7	**2.3**
HEART1	flop:	4812.7	5148.1	**1646.9**	3335.7	3108.9
	nz LU:	5140.1	5414.4	**3154.3**	4523.7	4229.5
	mem:	59.0	51.4	**39.0**	82.3	75.2
	mem/nz:	**12.0**	**9.9**	13.0	19.1	18.7
HB	time:	41.1	15.2	**8.0**	15.4	**6.7**
PSMIGR_1	flop:	**8577.7**	**8577.7**	**8578.9**	**9305.1**	**8613.9**
	nz LU:	**5820.7**	**5820.7**	**5821.7**	**6124.2**	**5856.9**
	mem:	**66.8**	**56.8**	131.1	173.1	165.5
	mem/nz:	**12.0**	**10.2**	23.6	29.6	29.6
median ratios	time:	5.15	1.90	1.00	3.70	0.45
	flop:	1.08	1.14	1.00	5.52	1.00
	nz LU:	1.04	1.07	1.00	2.35	1.01
	mem:	1.36	1.27	1.00	2.36	1.36
median	mem/nz:	12.09	11.05	11.55	12.36	15.62

Table 7: Results for symmetric set

Matrix		LU	SuperLU	UMFPACK4	MA38	MA41u
Norris	time:	19.0	**4.8**	**4.1**	168.1	25.6
torso2	flop:	**1336.7**	**1332.5**	**1331.4**	8538.3	**1453.9**
	nz LU:	**9742.7**	**9737.4**	**9736.1**	21619.7	**10421.6**
	mem:	**117.4**	119.0	**94.6**	227.8	124.3
	mem/nz:	**12.6**	12.8	**10.2**	**11.1**	**12.5**
Simon	time:	13.9	4.3	2.8	7.2	**1.7**
olafu	flop:	**2362.7**	2701.8	2573.7	4531.4	**1958.0**
	nz LU:	**6391.8**	**6732.3**	**6624.4**	8762.5	**5846.7**
	mem:	**74.0**	**66.9**	**67.1**	106.4	**72.1**
	mem/nz:	**12.1**	**10.4**	**10.6**	**12.7**	**12.9**
Simon	time:	16.2	5.1	3.8	9.3	**2.3**
venkat01	flop:	**2194.2**	**2194.2**	**2194.2**	5582.9	**2178.6**
	nz LU:	**11539.6**	**11539.6**	**11539.6**	16051.6	**11607.4**
	mem:	135.2	**122.8**	**101.3**	174.2	133.0
	mem/nz:	12.3	**11.2**	**9.2**	**11.4**	12.0
Bai	time:	16.4	4.6	3.5	28.8	**2.4**
af23560	flop:	**2675.9**	**2675.9**	**2675.9**	21293.3	**2688.2**
	nz LU:	**8317.9**	**8317.9**	**8317.9**	23421.3	**8391.7**
	mem:	96.4	**85.2**	**75.2**	270.6	**86.3**
	mem/nz:	12.2	**10.7**	**9.5**	12.1	**10.8**
Simon	time:	17.3	4.6	3.0	8.7	**2.2**
raefsky3	flop:	**2814.3**	**2814.3**	**2814.3**	6751.1	**2904.6**
	nz LU:	**8333.6**	**8333.6**	**8333.6**	12468.5	**8435.3**
	mem:	96.4	**83.1**	**76.5**	141.2	102.3
	mem/nz:	12.1	**10.5**	**9.6**	**11.9**	12.7
Zhao	time:	19.8	8.7	4.2	24.3	**2.7**
Zhao1	flop:	**3646.5**	**3646.5**	**3646.5**	20100.3	**3659.8**
	nz LU:	**6949.5**	**6949.5**	**6949.5**	16525.0	**7150.9**
	mem:	**81.2**	**73.7**	**69.1**	181.5	**79.4**
	mem/nz:	**12.3**	**11.1**	**10.4**	**11.5**	**11.6**
Zhao	time:	24.0	53.4	4.8	40.2	**2.6**
Zhao2	flop:	**4404.1**	12148.7	**4145.4**	29223.9	**3659.8**
	nz LU:	**8106.1**	17633.9	**7849.8**	24248.2	**7149.6**
	mem:	**94.5**	194.2	**81.6**	269.8	**79.4**
	mem/nz:	**12.2**	**11.5**	**10.9**	**11.7**	**11.6**

Table 8: Results for symmetric set, continued

Matrix		LU	SuperLU	UMFPACK4	MA38	MA41u
FIDAP	time:	32.5	17.5	**5.4**	81.3	**4.7**
EX11	flop:	**6001.2**	11797.4	**6001.2**	88326.4	**6660.4**
	nz LU:	**11392.0**	14527.8	**11392.0**	37810.4	**11900.3**
	mem:	**131.2**	143.2	**107.1**	536.5	**131.4**
	mem/nz:	**12.1**	**10.3**	**9.9**	14.9	**11.6**
SIMON	time:	66.7	19.7	9.9	31.7	**5.2**
RAEFSKY4	flop:	13387.7	13575.2	12899.0	30515.0	**8543.1**
	nz LU:	**16009.0**	**16110.2**	**15751.6**	24447.9	**13474.4**
	mem:	**184.2**	**159.7**	**170.9**	285.0	**152.3**
	mem/nz:	**12.1**	**10.4**	**11.4**	**12.2**	**11.9**
WANG	time:	45.7	20.1	**7.4**	34.7	**6.0**
WANG4	flop:	**9073.2**	**9073.2**	**9073.2**	34284.4	**10472.2**
	nz LU:	**10537.8**	**10537.8**	**10537.8**	22904.2	**11473.1**
	mem:	**121.9**	**106.4**	**106.5**	276.4	136.0
	mem/nz:	**12.1**	**10.6**	**10.6**	**12.7**	**12.4**
RONIS	time:	105.4	35.3	**16.1**	103.7	**15.0**
XENON1	flop:	**21124.2**	**21124.2**	**21124.2**	125373.1	26790.6
	nz LU:	**27131.9**	**27131.9**	**27131.9**	62683.6	**30118.5**
	mem:	313.0	**269.1**	**247.8**	626.6	314.2
	mem/nz:	12.1	**10.4**	**9.6**	**10.5**	**10.9**
VANHEUKELUM	time:	130.9	52.3	**16.9**	51.0	**13.7**
CAGE10	flop:	**27968.1**	**27968.1**	**27968.1**	53933.9	**26859.6**
	nz LU:	**15895.5**	**15895.5**	**15895.5**	22375.5	**15735.4**
	mem:	**182.5**	**154.3**	222.1	293.3	222.3
	mem/nz:	**12.0**	**10.2**	14.7	13.7	14.8
NORRIS	time:		162.2	72.4		**47.2**
STOMACH	flop:		**87040.3**	**87040.3**		**92290.0**
	nz LU:		**107262.5**	**107262.5**		**112364.6**
	mem:		**1069.7**	**878.1**		**1062.7**
	mem/nz:		10.5	8.6		9.9
median ratios	time:	5.02	2.05	1.00	4.71	0.66
	flop:	1.00	1.00	1.00	3.78	1.00
	nz LU:	1.00	1.00	1.00	2.17	1.01
	mem:	1.17	1.09	1.00	2.41	1.21
median	mem/nz:	12.13	10.46	10.19	11.88	11.86

Figure 6: Results relative to UMFPACK4 (unsymmetric set)

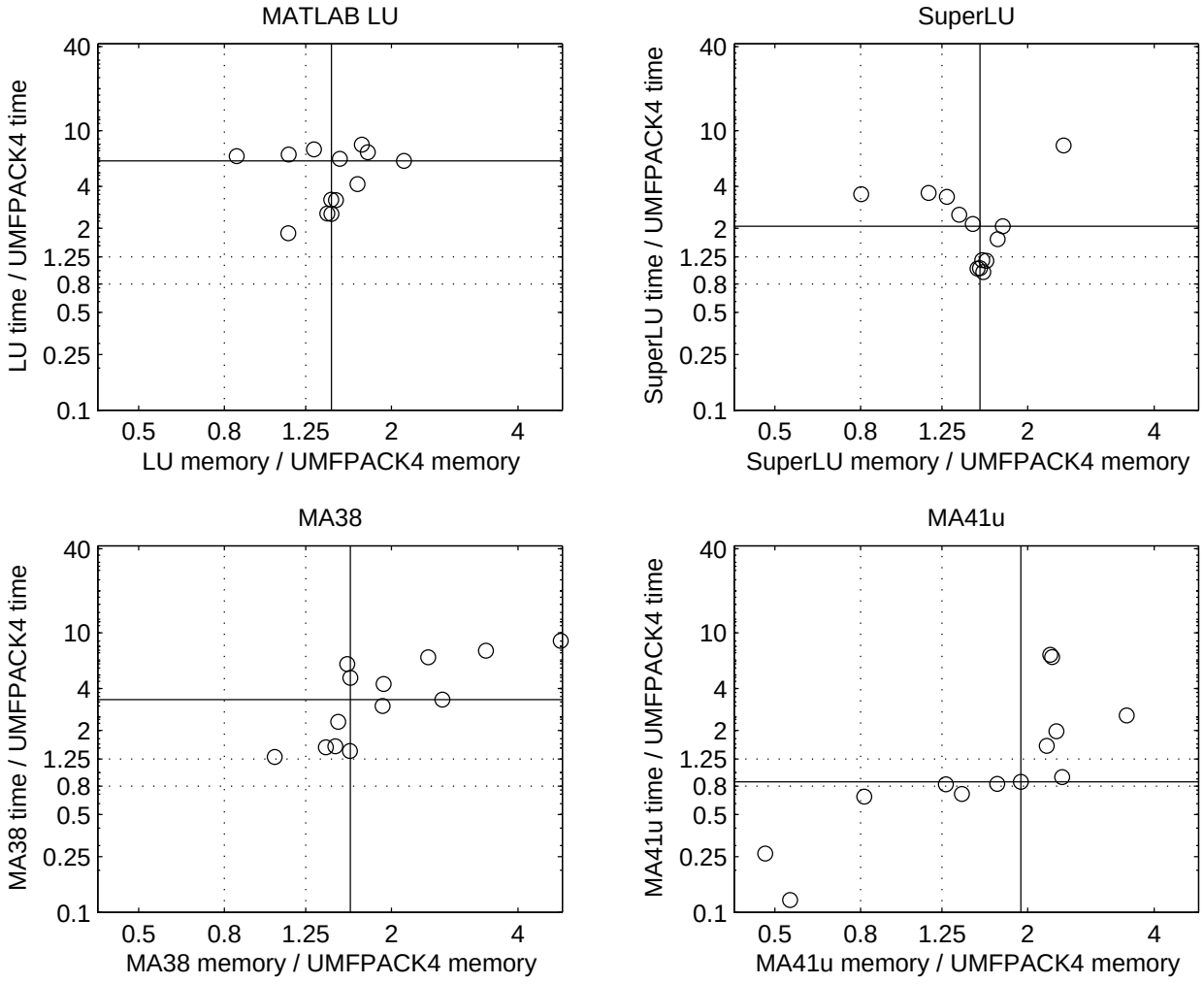


Figure 7: Results relative to UMFPACK4 (2-by-2 set)

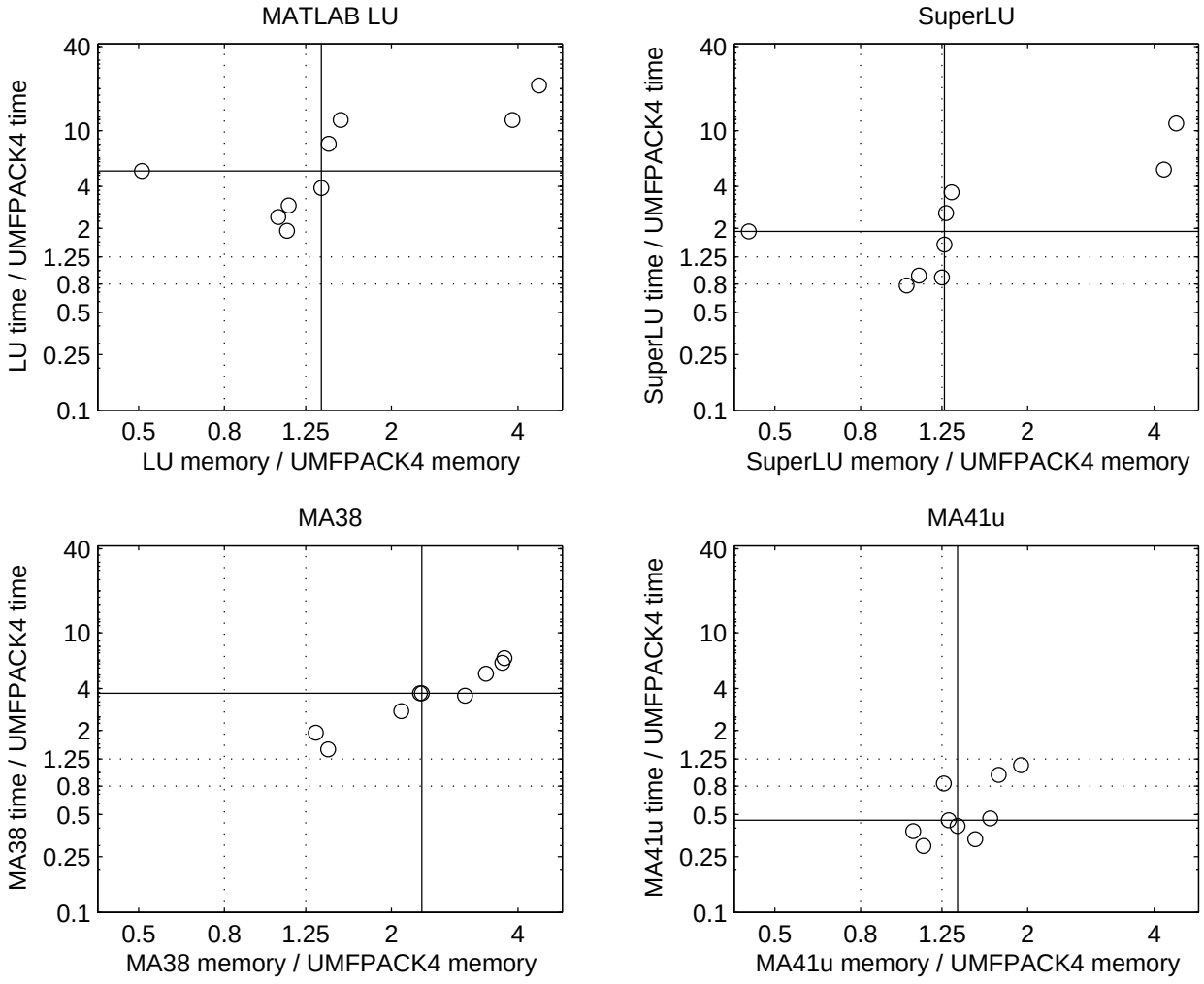
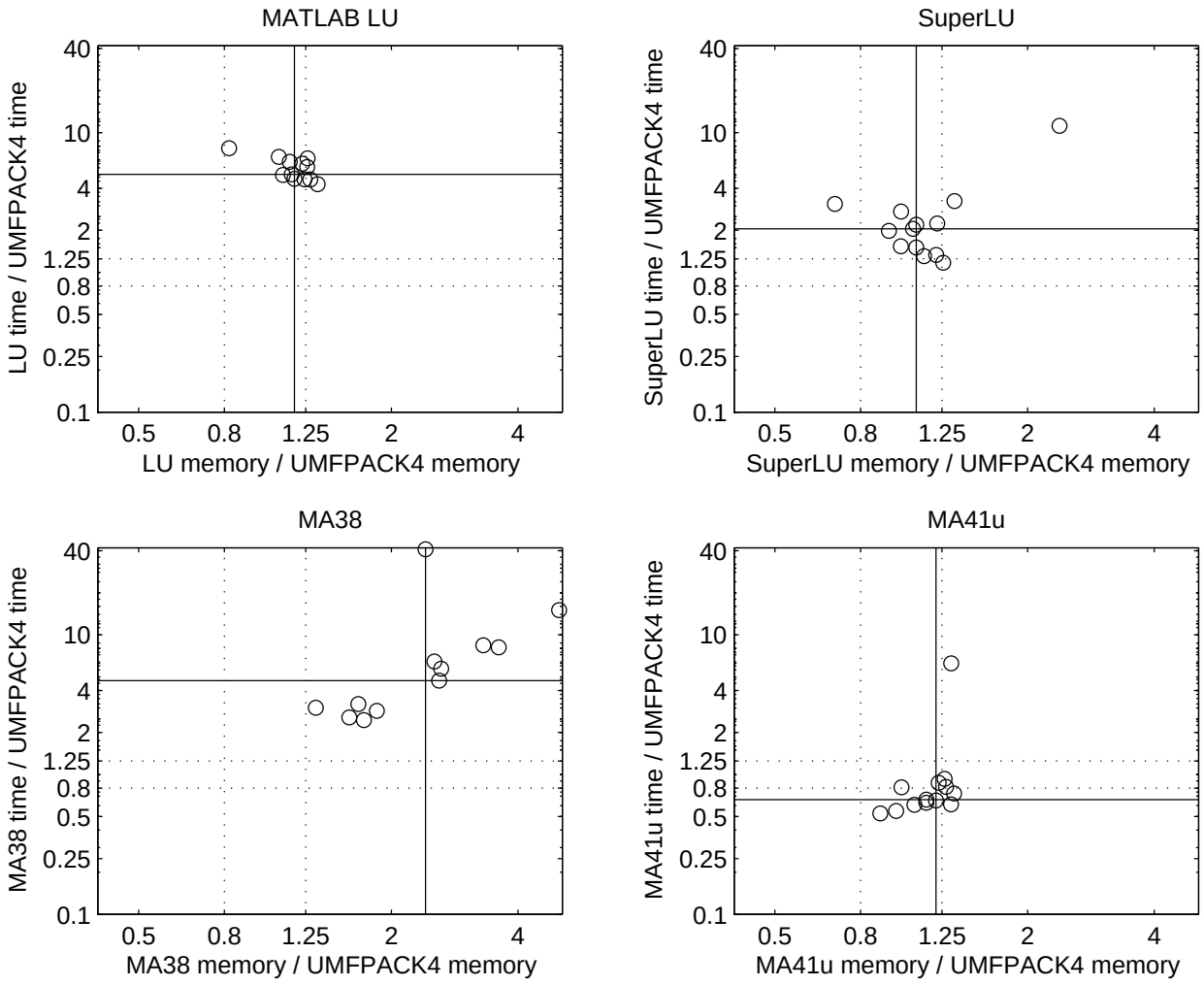


Figure 8: Results relative to UMFPACK4 (symmetric set)



the column elimination tree, and exploit the same worst-case upper bound on fill-in. Because UMFPACK4 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. UMFPACK4 is typically faster than LU and SuperLU in this particular environment for these matrices.

UMFPACK4 makes more efficient use of memory than all other methods in terms of bytes per nonzero entry in $\mathbf{L}$ and $\mathbf{U}$. Because it often finds better orderings than LU and SuperLU, and because its integer overhead for the nonzero patterns is typically lower, it nearly always uses less memory than either method. It typically finds orderings with as good a quality, or better, than MA41u.

UMFPACK4 is much faster than MA38, finds better orderings, and uses less memory for nearly all matrices, primarily because of its column pre-ordering strategies.

For matrices with unsymmetric nonzero pattern, UMFPACK4 typically uses almost half as much memory as MA41u and typically finds better orderings. In terms of run time, MA41u and UMFPACK4 are roughly split, in terms of which method is fastest on which particular unsymmetric matrix.

For matrices in the 2-by-2 set and symmetric set, MA41u is typically the fastest method of those considered here. UMFPACK4 is rarely more than twice as slow as MA41u, however, and almost always uses less memory than MA41u. With a few exceptions, LU, SuperLU, UMFPACK4, and MA41u find comparable orderings for matrices in the symmetric set, since all four are using the same pre-ordering method (AMD). Differences in scaling and pivoting strategies result in some variations. The symmetric strategy cannot be applied to MA38 since it lacks a pre-ordering and analysis phase.

With its symmetric strategy, and assuming no pivoting during numerical factorization, UMFPACK4 finds the same ordering as MA41u, and will construct nearly the same sequence of frontal matrices. They differ in their symbolic pre-analysis phase. With its symmetric strategy, UMFPACK follows its AMD ordering with an analysis of the upper bound fill-in and operation count, based on the Cholesky factorization of $\mathbf{A}^\mathsf{T}\mathbf{A}$. It uses a method similar to COLAMD, except without reordering the columns. This is slower than a recent method based on the union-find data structure for finding row and column counts for sparse QR or LU factorizations [41]. Also, UMFPACK4's numerical factorization phase is somewhat slower because of its unifrontal-style strategy of shifting pivot rows in and out of the current frontal matrix. This reduces memory usage, but swapping rows leads to non-stride-1 access, which has a significant performance penalty on the Pentium 4M. Finally, UMFPACK4 maintains element lists in case off-diagonal pivoting is needed. MA41 and MA41u do not use element lists. These three factors account for most of the difference in run time between UMFPACK4 and MA41u on matrices with symmetric nonzero pattern.

Brainman and Toledo [12] 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. Many of the matrices reported here fall in this category. Since UMFPACK4 exploits the same upper bound on fill-in as left-looking methods, and since UMFPACK4 can accept any given column pre-ordering instead of its default COLAMD or AMD orderings,

their method can be applied to UMFPACK4 to improve its performance on 2D and 3D problems. The other solvers discussed here (except MA38) can also significantly benefit from the use of a nested dissection based ordering of $\mathbf{A} + \mathbf{A}^\mathsf{T}$ [5] or $\mathbf{A}^\mathsf{T}\mathbf{A}$.

The BBMAT matrix obtained from Horst Simon has an interesting history [57]. In 1988 it took about 1000 seconds and slightly over 1 GB of memory to factorize on a Cray-2 using a bandwidth reducing ordering and a banded solver. The matrix represents a 2D torus, with structural asymmetry along the grid lines that run from the center to the outside (representing turbulence). The band reducing ordering was constructed manually, based on the knowledge of the mesh. Unsymmetric sparse LU factorization codes available to Simon and his colleagues at that time were unable to factorize the matrix. In 2003, all 5 of the methods presented here can easily factorize the matrix, most with much less memory. In 15 years, the run time has been cut to about 30 seconds, and the memory requirement has been reduced by a factor of 3. The Dell Latitude laptop used in these experiments is about twice as fast as a single processor of the Cray-2, and has the same amount of memory available to a single process (1 GB). The memory savings and an additional factor of 15 improvement in run-time are purely due to algorithmic improvements.

6 Summary

The new method presented here, UMFPACK4, 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, or comparable orderings for symmetric structured matrices. Based on the same criteria, it always outperforms its predecessor, MA38. Compared with MA41u, it typically finds an ordering that results in fewer nonzeros in $\mathbf{L}$ and $\mathbf{U}$ for unsymmetric structured problems, and identical fill-in for symmetric structured matrices. It uses significantly less memory than MA41u.

UMFPACK Version 4.1 (and earlier versions), COLAMD, and AMD are available at www.cise.ufl.edu/research/sparse, and as Collected Algorithms of the ACM [2, 16, 19]. UMFPACK Version 4.0, which includes only the unsymmetric strategy and takes less advantage of the level-3 BLAS, appears as a built-in routine in MATLAB 6.5, as the LU and the forward and backslash matrix operator (`x = A\b`).[4]

7 Acknowledgements

The MathWorks, Inc., supported the development of Version 4.0; namely, the ability to handle complex, rectangular, and singular matrices. The upgrade from Version 4.0 to 4.1, including the symmetric and 2-by-2 strategies, was done while on sabbatical at Stanford University and Lawrence Berkeley National Laboratory (with funding from the SciDAC

[4] MATLAB 6.5 uses default UMFPACK v4.0 parameter settings, with one important exception in the forward and backslash operator. If the estimate of the reciprocal of the condition number is smaller than machine epsilon, then the partial pivoting threshold is set to 1.0 and the matrix is refactorized, in an attempt to get a more accurate solution. The MATLAB user can also disable the COLAMD pre-ordering, and can modify the print level for diagnostic output, but by default these options are the same as the UMFPACK v4.0 defaults.

program). I would like to thank Gene Golub, Horst Simon, and Esmond Ng for making this sabbatical possible. I would like to thank Sherry Li for collaborating on the implementation of the symmetric strategy for SuperLU.

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, and I. S. Duff. Algorithm 8xx: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 2003 (under submission). Also TR-03-010 at www.cise.ufl.edu/tech-reports.

[3] 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.

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

[5] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Trans. Math. Softw.*, 27(4):388–421, 2001.

[6] 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.

[7] P. R. Amestoy and C. Puglisi. An unsymmetrized multifrontal LU factorization. *SIAM J. Matrix Anal. Applic.*, 24:553–569, 2002.

[8] 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.

[9] 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.

[10] 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.

[11] A. J. Berger, J. M. Mulvey, E. Rothberg, and R. J. Vanderbei. Solving multistage stochastic programs using tree dissection. Technical Report SOR-95-07, Dept. of Civil Eng. and Operations Research, Princeton Univ., Princeton, NJ, June 1995.

[12] I. Brainman and S. Toledo. Nested-dissection orderings for sparse LU with partial pivoting. *SIAM J. Matrix Anal. Applic.*, 23:998–1012, 2002.

[13] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 8:639–655, 1971.

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

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

[16] T. A. Davis. Algorithm 8xx: UMFPACK V4.1, an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 2003 (under submission). Also TR-03-007 at www.cise.ufl.edu/tech-reports.

[17] 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.

[18] 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.

[19] 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.).

[20] 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.).

[21] 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.

[22] 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.

[23] I. S. Duff. On permutations to block triangular form. *J. Inst. of Math. Appl.*, 19:339–342, 1977.

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

[25] 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.

[26] I. S. Duff. A new code for the solution of sparse symmetric definite and indefinite systems. Technical Report TR-2002-024, Rutherford Appleton Laboratory, 2002.

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

[28] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Applic.*, 22(4):973–996, 2001.

[29] 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.

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

[31] I. S. Duff and J. K. Reid. A note on the work involved in no-fill sparse matrix factorization. *IMA J. of Numerical Analysis*, 3:37–40, 1983.

[32] 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.

[33] 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.

[34] 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.

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

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

[37] 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.

[38] 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.

[39] J. Gilbert and E. G. Ng. Predicting structure in nonsymetric sparse matrix factorizations. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computations*, volume 56 of *The IMA Volumes in Mathematics and its Applications*. Springer-Verlag, 1993.

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

[41] J. R. Gilbert, X. S. Li, E. G. Ng, and B. W. Peyton. Computing row and column counts for sparse QR and LU factorization. *BIT*, 41(4):693–710, 2001.

[42] 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.

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

[44] K. Goto and R. van de Geijn. On reducing TLB misses in matrix multiplication, FLAME working note 9. Technical Report TR-2002-55, The University of Texas at Austin, Department of Computer Sciences, Nov. 2002.

[45] A. Gupta. Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices. *SIAM J. Matrix Anal. Applic.*, 24:529–552, 2002.

[46] 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.

[47] 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.

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

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

[50] 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.

[51] J. W. H. Liu. On general row merging schemes for sparse Givens transformations. *SIAM J. Sci. Statist. Comput.*, 7(4):1190–1211, Oct. 1986.

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

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

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

[55] 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.

[56] 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.

[57] H. Simon. personal communication.

[58] R. C Whaley, A. Petitet, and J. J. Dongarra. Automated emperical optimization of software and the ATLAS project. Technical Report LAPACK Working Note 147, Computer Science Department, The University of Tennessee, September 2000.

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