THE USE OF GRAPH THEORY IN A PARALLEL MULTIFRONTAL METHOD FOR SEQUENCES OF UNSYMMETRIC PATTERN SPARSE MATRICES

STEVEN M. HADFIELD* AND TIMOTHY A. DAVIS†

Computer and Information Science and Engineering Department, University of Florida, Technical Report TR-95-016, April 1995.

Abstract. Multifrontal matrix factorization methods used for solving large, sparse systems of linear equations decompose sparse matrices into overlapping dense submatrices which can be represented by vertices with relationships between submatrices shown via various types of edges. This paper describes the use of graph theory in a new parallel, distributed memory multifrontal method for the LU factorization of sequences of matrices with an identical, unsymmetric pattern. The directed acyclic graphs formed by these vertices and the various edge sets are used to structure the computations, schedule the parallel factorization, and provide a robust capability to dynamically change the pivot ordering to maintain numerical stability. Pivot reordering determines necessary permutations based on a path analysis of two component edge sets. The path properties represented by these edge sets define the impacts of these permutations on the structures of the submatrices and the number of nonzeros in the matrix factors. Transitive reductions of these edge sets provide the communications paths needed for parallel implementation.

1. Introduction. Multifrontal techniques for solving large, sparse systems of linear equations are becoming extremely popular because of their unique capabilities to take advantage of high performance computer architectures. Until recently, these methods always assumed a symmetric structure in the system. Recently Davis and Duff have developed an unsymmetric pattern multifrontal method based on LU factorization which can significantly reduce the amount of required computations for systems with an unsymmetric structure [3]. Comparison studies have shown this method to frequently be the most efficient method for solving such systems [14]. Furthermore, this method has demonstrated significant potential for parallelism [10, 11] and can be used effectively to solve sequences of systems of linear equations that maintain an identical structure such as those that can occur when solving systems of differential-algebraic equations. These equations arise in many application areas including circuit simulation, chemical engineering, magnetic resonance spectroscopy, and air pollution modeling [4, 12, 15, 17].

The use of graph theory is fundamental to multifrontal methods. Specifically, graph theory constructs and techniques are used to define the multifrontal decomposition, to facilitate the use of parallelism in the method, and to allow the decomposition to be repeatly used when solving sequences of systems. This paper describes these uses of graph theory in an extension to Davis and Duff's unsymmetric pattern

* Department of Mathematical Sciences, US Air Force Academy, Colorado, USA. phone: (719) 472-4470, email: hadfieldsm%dfms%usafa@dfmail.usafa.af.mil

† Computer and Information Sciences Department, University of Florida, Gainesville, Florida, USA. phone: (904) 392-1481, email: davis@cis.ufl.edu. This project is supported by the National Science Foundation (ASC-9111263, DMS-9223088), and by Cray Research, Inc., through the allocation of supercomputer resources.

multifrontal method that implements the parallel LU factorization of sequences of systems that maintain an identical structure. We start off by reviewing LU factorization and the unsymmetric pattern multifrontal method. The various uses of graph theory are then summarized with particular attention paid to its use when dynamically reordering the pivots to maintain numerical stability. Some performance results are presented to illustrate the effectiveness of the approach both in sequential and parallel environments.

2. LU Factorization. LU Factorization is a common and popular technique based on Gaussian elimination used to solve general systems of linear equations in the form $A\bar{x} = \bar{b}$. The technique first factors the coefficient matrix A into the product of two triangular matrices L and U where L is unit lower triangular and U is upper triangular. The factored system, $LU\bar{x} = \bar{y}$, is then used to solve for $\bar{x}$ using a forward substitution solving for $\bar{y}$ in $L\bar{y} = \bar{b}$ and then a back substitution solving for $\bar{x}$ in $U\bar{x} = \bar{y}$. Permutations to the coefficient matrix can be included during the factorization to improve the numerical stability of the method. The most common technique is partial pivoting where the largest magnitude entry in the current pivot column is permuted to the pivot entry.

When the coefficient matrix is large and sparse, the amount of required computations can be significantly reduced by selecting pivot orderings that reduce the number of zero entries in the coefficient matrix that become nonzero in the corresponding LU factorization. This phenomena is know as *fill-in*. To provide more flexibility in pivot ordering to reduce fill-in, the partial pivoting strategy is typically relaxed to allow the choice from multiple alternative pivots in a particular pivot column. A *pivot threshold* is a tolerance factor between 0 and 1 with entries whose magnitudes are greater than the pivot threshold times the maximum magnitude entry in the column to be viable as pivots.

3. Multifrontal Concepts. Multifrontal methods structure sparse matrix factorization by decomposing the sparse matrix into a set of overlapping dense submatrices called *frontal matrices*. Each frontal matrix is partially factored by one or more pivots. Entries in the unfactored portion of the frontal matrix (called the *contribution block*) are uniquely assembled (added) into subsequent frontal matrices. An *assembly directed acyclic graph (DAG)* is used to define the computational structure with vertices representing frontal matrices and edges representing the passing of contribution block entries from one frontal matrix to another. With symmetric pattern multifrontal methods, the assembly DAG is a tree (or forest) as each frontal matrix's contribution block can be completely absorbed within a single subsequent frontal matrix [1, 5, 6, 7, 8, 13]. With the unsymmetric pattern multifrontal method, the contribution block can be fragmented and portions must be passed to different subsequent frontal matrices which results in more generalized DAG structure.

An example of an unsymmetric pattern multifrontal decomposition is shown in Figure 3.1. The matrix A is decomposed using its first two pivots; P_1 and P_2. The pattern of nonzeroes in the first row and column are used to define the first frontal

matrix E_1. One step of LU factorization is performed on E_1 producing the first column of L and row of U. The rest of the frontal matrix E_1 is passed to subsequent frontal matrices as a contribution block. The entry represented by a '*' was initially a zero that became nonzero from the factorization step of E_1. Importantly, this requires the pattern of columns associated with the second pivot's frontal matrix, E_2, to be extended by that column.

$$A = \begin{pmatrix} P_1 & \cdot & a & b & \cdot \\ c & P_2 & d & * & e \\ f & g & \cdot & \cdot & \cdot \\ \cdot & h & i & j & \cdot \\ k & \cdot & \cdot & \cdot & l \end{pmatrix} \quad E_1 = \begin{pmatrix} P_1 & a & b \\ c & d & * \\ f & \cdot & \cdot \\ k & \cdot & \cdot \end{pmatrix} \quad E_2 = \begin{pmatrix} P_2 & \bar{d} & * & e \\ g & \cdot & \cdot & \cdot \\ h & i & \cdot & \cdot \end{pmatrix}$$

FIG. 3.1. *Unsymmetric Pattern Multifrontal Decomposition Example*

The relationship between E_1 and E_2 in Figure 3.1 is referred to as an *L relationship* and forces the pattern of nonpivotal columns in the first frontal matrix to be included in the pattern of columns in the subsequent frontal matrix. This occurs as a direct result of the fill-in that was illustrated with the '*' entry. The four possible types of relations that can exist between frontal matrices are shown in Figure 3.2. No relationship implies that the two frontal matrices are independent and provides for a high level parallelism. The U relationship is the transpose of the L relationship and implies an inclusion property for the pattern of rows similar to the pattern of columns inclusion property for the L relationship. The LU relationship implies both inclusion properties and is the reason why the symmetric case results in an assembly tree as the contribution block of the first matrix can be fully absorbed within the LU ancestor frontal matrix.

$$\text{No relationship} \begin{pmatrix} P_1 & 0 & \cdots \\ 0 & P_2 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \quad \text{L relationship} \begin{pmatrix} P_1 & 0 & \cdots \\ X & P_2 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

$$\text{U relationship} \begin{pmatrix} P_1 & X & \cdots \\ 0 & P_2 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \quad \text{LU relationship} \begin{pmatrix} P_1 & X & \cdots \\ X & P_2 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

FIG. 3.2. *Possible Relationships between Frontal Matrices*

4. Use of Graph Theory. The use of graph theory by the unsymmetric pattern multifrontal method focuses on the assembly DAG. The assembly DAG tracks the L

and U relationships which directly track the relationships between the frontal matrix patterns. These edges also define the data passing and data dependencies that exist within the developed computational structure. The assembly DAG also serves as a task graph for scheduling the parallel factorization. A critical path analysis of the assembly DAG is used to define task priorities for each frontal matrix task and a static schedule for the factorization is derived from these priorities using list scheduling techniques. (Recall the target environment is a distributed memory hypercube which justifies the use of static scheduling techniques). Furthermore, each frontal matrix task can be allocated multiple processors in subcube configurations. Assignment of specific subcubes is also done using the assembly DAG edges in a manner that attempts to minimize the amount of inter-processor passing of contribution blocks.

Some of the most interesting uses of graph theory occur when having to accommodate dynamic pivot reordering in the factorization of sequences of identically structured sparse matrices. Such sequences arise when solving systems of differential algebraic equations. The linearization of these systems can create sequences of matrices with an identical, unsymmetric pattern where the values of the nonzeroes of the pattern change across matrices in the sequence. An assembly DAG can be created for the first matrix in the sequence and then reused for the subsequent matrices. However, as the values of the nonzeroes change, entries selected as pivots in the original assembly DAG may take on values that make them no longer numerically acceptable as pivots and a dynamic pivot reordering is required.

5. Dynamic Pivot Reordering. The strategy to deal with dynamic pivot reordering uses three techniques: avoidance, intra-frontal matrix recovery, and inter-frontal matrix recovery. *Avoidance* modifies the pivot threshold (discussed earlier) to reduce the need for the more complex techniques. Specifically, the pivot threshold is set higher (closer to one) for the first matrix in the sequence from which the assembly DAG is constructed. This pivot threshold is then relaxed to smaller values for subsequent matrices to improve the likelihood that anticipated pivots will be numerically acceptable. While this technique does worsen the error bounds, the exact effect on the bounds is well-defined and to a significant degree controllable.

When anticipated pivot entries are not numerically acceptable per the pivot threshold criteria, the next recourse would be *intra-frontal matrix recovery* which searches for alternative pivot entries in the frontal matrix's block of potential pivots. If such an entry is found, non-symmetric permutations are performed to position the replacement pivot. Such permutations are possible only if the frontal matrix has more than one potential pivot. For a frontal matrix with k potential pivots, alternative pivots may be selected from any entry in the leading principal $k \times k$ submatrix (called the pivot block). These permutations will not alter the pattern of rows or columns that define the frontal matrix and will not affect the structure of other frontal matrices or the assembly DAG.

The last recourse is *inter-frontal matrix recovery* where the remaining rows and columns of the pivot block in which acceptable pivots were not present are symmetrically permuted to a subsequent frontal matrix in the assembly DAG. In this technique,

the assembly DAG and subsequent frontal matrices will be affected. Graph theory is an invaluable tool used to define both the scope and details of inter-frontal matrix recovery. An example will illustrate this type of recovery before we delve deeper into the detailed use of graph theory.

The matrix in Figure 5.1 is a sparse matrix with rows and columns labeled by the frontal matrix ID (A through E) that holds the row/column as a potential pivot. This figure also shows the symmetrically permuted version of the matrix where the A_l rows and columns associated with the loss of the second pivot in the frontal matrix A are moved to become part of frontal matrix D. The entries shown with '*' in the second matrix are new fill-in that occurs as a result of the recovery. Figure 5.2 shows the assembly DAG associated with the original matrix and Figure 5.3 shows the frontal matrices themselves. In general, we want to select a recovery frontal matrix to which there is an LU relationship from the failed frontal matrix. The inclusion properties of the patterns of rows and columns that is implied by the LU relationship assures that the lost pivot columns and rows will be fully absorbed into such a matrix. The LU relationship does not need to be direct. In Figure 5.2 we see that there is not a direct LU relationship between the failed frontal matrix A and the recovery matrix D, but there is an 'implied' LU relationship from A to D which results from the path of L edges from A to B to D and the path of U edges from A to C to D. These L and U paths imply the same inclusion properties as a direct LU edge would. Furthermore, the L and U paths define where the additional fill-in will occur in intervening frontal matrices. Specifically, L ancestors (frontal matrices to which there is a path of L edges from the failed frontal matrix) must be extended by the lost pivot columns and will result in new fill-in within these columns and U ancestors must be extended by the lost pivot rows and will also result in new fill-in. Examples are shown in Figure 5.3 with the L ancestor B and the U ancestor C.

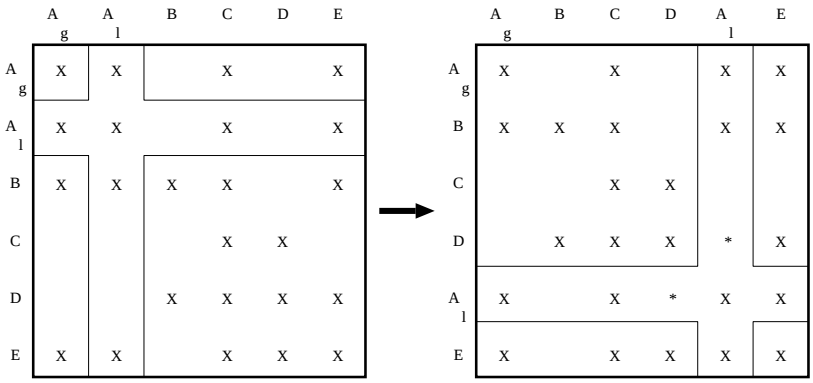

FIG. 5.1. *Sparse Matrix with Lost Pivots*

With the basic understanding of inter-frontal matrix recovery provided in the previous paragraph, we can now progress to some of the theoretical results that establish the sufficiency of this technique to resolve any required pivot reordering. Only the key results are described and done so without proofs. Intermediate results, proofs, and other details can be found in [9]. These results make extensive use of

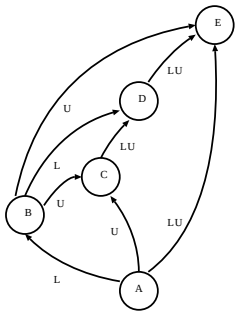

FIG. 5.2. *Sample Assembly DAG*

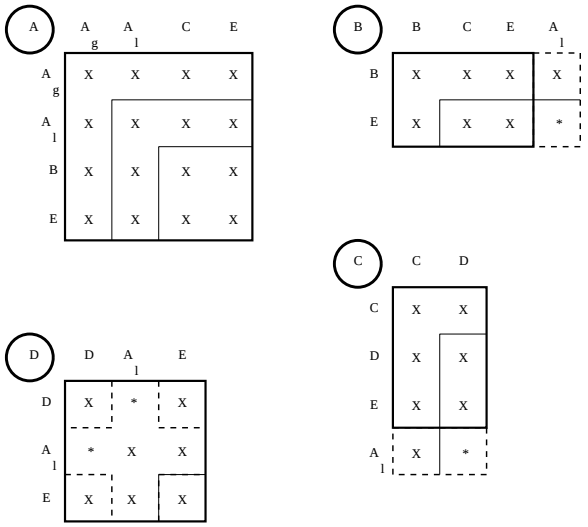

FIG. 5.3. *Frontal Matrices Extended by Lost Pivot Recovery*

graph structures (primarily the assembly DAG) and graph theory. They start off with some key foundational results and some results that define the scope of resulting fill-in. Then the more complex issues of multiple and repeated failures are addressed. Finally, the edge sets required for inter-frontal matrix recovery are analyzed.

5.1. Foundational Results. The foundational theorems used to establish the inter-frontal matrix recovery are basic extensions to the inclusions properties implied by the various edge types. Specifically these results are:

- Paths of L edges imply inclusion of pattern of columns.
- Paths of U edges imply inclusion of pattern of rows.
- An L path from A to B and a U path from A to C imply either an L path from B to C (if B<C) or a U path from C to B (if C<B).

5.2. Inter-Frontal Matrix Recovery Fill-In. A key component of inter-frontal matrix recovery is the ability to track and explicitly define the extent of the additional fill-in that results. The results below show that this can be accomplished using the L and U relationships that exist between frontal matrices and are represented as edges in the assembly DAG.

- The only fill-in in the lost pivot columns is due to L ancestors.
- The only fill-in in the lost pivot rows is due to U ancestors.
- Recovery frontal matrices need only have their pivot rows and columns extended.
- No additional contributions will be made to the lost pivot block by L or U ancestors.

5.3. Multiple and Repeated Failures. When multiple frontal matrices experience lost pivots, subsequent frontal matrices can be affected by multiple recoveries. When such a subsequent frontal matrix is an L ancestor of one frontal matrix and a U ancestor of another frontal matrix, it must be extended by both rows and columns. Their intersection is called the *overlap* and must be specially handled. Furthermore, pivots that failed once may fail again in the recovery matrix. Such repeated failures are another source of complication. The results below address both of these issues and resolve them in a complete and elegant manner.

- Resolution of extended contribution block overlap due to failed L and U predecessors can be resolved by ascending topological levels of the respective recovery matrices.
- Once a recovery has reached its recovery matrix (vertex), that recovery is complete. Repeated failures can be handled as failures in the recovery matrix. (Recoveries need not progress past the topological level of the recovery matrix).
- It is sufficient to order multiple recoveries by ascending topological level of their recovery matrices.

5.4. Required Edge Sets. Up until this point we have been fairly loose in our definition of the assembly DAG's edge set. We have implied that it is made up of the 'true' relationships that exist between frontal matrices as defined by the L and U relationships. Actually, the assembly DAG edge set is a much smaller subset of these true relations and can include 'early assembly edges' that allow contributions to be combined earlier (See [3] for details). In order to establish criteria for a sufficient edge set for inter-frontal matrix recovery, we will now distinguish between the 'true relations' and the assembly edges as two distinct edge sets. The following results establish when these edge sets are sufficient as well as when they are not. Furthermore, when they are not sufficient, the results definitize how they must be augmented to become sufficient.

- No additional edges are needed if recovery matrix is an LU ancestor.
- If no LU ancestor exists, new L edges from failed frontal matrix and from its U ancestors are needed to the recovery matrix. Also, new U edges from failed frontal matrix and from its L ancestors are needed to the recovery matrix.
- In the general case, the assembly DAG edge set and even the true relation edges are insufficient for lost pivot recovery.
- A sufficient edge set can be built from the true relation edges by augmenting them such that there is only one terminal vertex. Furthermore, a transitive

reduction of this augmented edge set is sufficient for lost pivot recovery.

- If the overall matrix permuted to block triangular form, then there is always an LU ancestor and both the assembly edge set and the true relations (or their transitive reductions) are sufficient for lost pivot recovery.

The last results based on block triangular form are significant in that they preclude the need to augment the basic edge sets but also in that this form creates independent connected components within the assembly DAG which provides an additional, very high level of parallelism.

6. Performance Results. Performance results are provided for both sequential and parallel execution time. The three matrix sequences used for this evaluation are RDIST1, RDIST2, and RDIST3A which come from chemical engineering applications [15, 16]. Their characteristics are shown in Table 6.1.

TABLE 6.1
Characteristics of Matrix Sequences

NAME	ORDER	NONZEROS	# OF MATRICES
RDIST1	4134	94,408	41
RDIST2	3198	56,834	40
RDIST3A	2398	61,896	37

The effectiveness and efficiency of lost pivot recovery on a single processor was measured using single processor refactorizations of each matrix in the RDIST1, RDIST2, and RDIST3A sequences. Separate sets of runs were done where avoidance was used and where it was not. For the avoidance runs, the initial pivot threshold value of 0.1 was relaxed to 0.001. The runs without avoidance maintained the pivot threshold at 0.1. To put these times into perspective, they will be compared to an estimation of the analyze-factor time (where a distinct assembly DAG is built for each matrix in the sequence based on the changing values of the entries). Only an estimated analyze-factor time is possible, as the analyze-factor routine (taken from Davis' original unsymmetric multifrontal implementation: UMFPACK [2]) required too much memory to be run on an single nCUBE processing node. The estimate of the analyze-factor time was achieved by taking the ratio of the UMFPACK analyze-factor time to the UMFPACK refactor time as run on a single CRAY YMP processor and multiplying the ratio by the single nCUBE 2 processing node execution time of the fixed pivot order refactorization.

The sequential time savings were calculated by comparing the sum of the estimated analyze-factor times for all the matrices in the sequence to the observed times when using the refactorization code with dynamic pivot reordering as described in this paper. The percentage time savings are provided in Table 6.2.

In order to characterize the parallel performance, runs were done on selected matrices in each of the test sequences using a 32 processor configuration of the nCUBE 2 multiprocessor. Table 6.3 reports the range of speed ups achieved both without and with the use of the avoidance strategy described earlier.

TABLE 6.2
Sequential Performance Improvements

MATRIX NAME	IMPROVEMENT	
	W/O AVOIDANCE	WITH AVOIDANCE
RDIST1	58.8 %	60.3 %
RDIST2	48.3 %	54.3 %
RDIST3A	1.2 %	31.3 %

TABLE 6.3
Parallel Performance Speed Ups

MATRIX NAME	OBSERVED SPEED UP RANGE	
	W/O AVOIDANCE	WITH AVOIDANCE
RDIST1	12.2-12.4	12.3-12.4
RDIST2	7.3-10.6	8.1-10.6
RDIST3A	5.2-12.0	7.8-12.0

The maximum speed ups achieved with dynamic pivot reordering are about two less than the corresponding speed ups achieved using an earlier version of the algorithm that assumes the anticipated pivots are numerically acceptable and does no numerical checking or reordering. Furthermore, the achieved speed ups as well as the sequential execution times are highly dependent on the amount of intra- and inter-frontal matrix recovery that occurs. The RDIST3A matrices experienced the largest amount of inter-frontal matrix recovery with as many as 218 lost pivots across frontal matrices when avoidance was not employed.

The bottom line of these performance results is that the techniques outlined in this paper of reusing the assembly DAGs and dynamically reordering pivots as necessary are viable ways to improve overall factorization time. Furthermore, they parallelize well and can take further advantage of multiprocessor environments.

7. Conclusion. This paper has described the use of graph theory in a technique for the LU factorization of sequences of identically structured sparse matrices which is based on the highly successful unsymmetric pattern multifrontal method of Davis and Duff [3]. The graph theory centers on the concept of the assembly DAG which provides the basic computational structure of the method. Graph analysis techniques are used facilitate frontal matrix task scheduling. Various edge types in the assembly DAG are used to direct data passing and trace the frontal matrix pattern dependencies. When the unsymmetric pattern multifrontal method is applied to sequences of matrices, a key consideration is accommodating the dynamic pivot reordering necessary to maintain numerical stability. This is done using the three part strategy of avoidance, intra-frontal matrix recovery, and inter-frontal matrix recovery. Graph theory techniques are especially important in inter-frontal matrix recovery and the results in this area were highlighted by this paper. Finally, some of the achieved performance results indicated that the techniques described here are viable ways to

improve performance as well as to take advantage of parallelism.

REFERENCES

[1] P. R. Amestoy and I. S. Duff, *Vectorization of a multiprocessor multifrontal code*, International Journal of Supercomputing Applications, 3 (1989), pp. 41–59.

[2] T. A. Davis, *Users' guide for the unsymmetric-pattern multifrontal package (UMFPACK)*, Tech. Report TR-93-020, Computer and Information Sciences Department, University of Florida, Gainesville, FL, June 1993. (UMFPACK is available via Netlib (linalg/umfpack.shar) or via anonymous ftp to ftp.cis.ufl.edu:pub/umfpack).

[3] T. A. Davis and I. S. Duff, *An unsymmetric-pattern multifrontal method for sparse LU factorization*, SIAM J. Matrix Anal. Appl., (submitted March 1993, under revision. Also TR-94-038.).

[4] I. S. Duff, *Sparse Matrices and their Uses*, Academic Press, New York and London, 1981.

[5] I. S. Duff, *Parallel implementation of multifrontal schemes*, Parallel Computing, 3 (1986), pp. 193–204.

[6] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford Science Publications, New York, NY, 1989.

[7] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Transactions on Mathematical Software, 9 (1983), pp. 302–325.

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

[9] S. M. Hadfield, *On the LU Factorization of Sequences of Identically Structured Sparse Matrices within a Distributed Memory Environment*, PhD thesis, University of Florida, Gainesville, FL, April 1994. (also TR-94-019).

[10] S. M. Hadfield and T. A. Davis, *Analysis of potential parallel implementation of the unsymmetric-pattern multifrontal method for sparse LU factorization*, Tech. Report TR-92-017, Department of Computer and Information Systems, University of Florida, Gainesville, FL, 1992.

[11] S. M. Hadfield and T. A. Davis, *A parallel unsymmetric-pattern multifrontal method*, SIAM J. Sci. Computing, (1994). (submitted. Also Univ. of Fl. tech report TR-94-028).

[12] K. S. Kundert, *Sparse matrix techniques and their applications to circuit simulation*, in Circuit Analysis, Simulation and Design, A. E. Ruehli, ed., New York: North-Holland, 1986.

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

[14] E. G.-Y. Ng, *Comparison of some direct methods for solving sparse nonsymmetric linear systems*, in 5th SIAM Conference on Applied Linear Algebra, SIAM, June 1994, p. 140.

[15] S. E. Zitney, *Sparse matrix methods for chemical process separation calculations on supercomputers*, in Proc. Supercomputing '92, Minneapolis, MN, Nov. 1992, IEEE Computer Society Press, pp. 414–423.

[16] S. E. Zitney and M. A. Stadtherr, *Supercomputing strategies for the design and analysis of complex separation systems*, Ind. Eng. Chem. Res., 32 (1993), pp. 604–612.

[17] Z. Zlatev, *Sparse matrix techniques for general matrices with real elements: Pivotal strategies, decompositions and applications in ODE software*, in Sparsity and Its Applications, D. J. Evans, ed., Cambridge, United Kingdom: Cambridge University Press, 1985, pp. 185–228.