LOST PIVOT RECOVERY FOR AN UNSYMMETRIC-PATTERN MULTIFRONTAL METHOD

STEVEN M. HADFIELD* AND TIMOTHY A. DAVIS†

Computer and Information Sciences Department, University of Florida
Technical Report TR-94-029.

Abstract. The unsymmetric-pattern multifrontal method by Davis and Duff relaxes the assumption of a symmetric pattern matrix and has considerable performance advantages for the factorization of sequences of identically structured sparse matrices with unsymmetric patterns. However, when anticipated pivots become no longer numerically acceptable, standard multifrontal recovery techniques are inadequate. This paper develops a robust lost pivot recovery capability for the unsymmetric pattern multifrontal method. The recovery capability is built into a distributed memory, parallel implementation of the method and both its sequential and parallel performance are evaluated.

Key words. LU factorization, unsymmetric sparse matrices, multifrontal methods, parallel algorithms

AMS subject classifications. 65F50, 65F05

1. Introduction. Systems of differential-algebraic equations arise in many application areas including circuit simulation, chemical engineering, magnetic resonance spectroscopy, and air pollution modeling [4, 17, 20, 22]. Methods for solving such systems frequently produce sequences of identically structured sparse matrices whose patterns can be highly unsymmetric. The LU factorization of these matrices allow their corresponding linear systems to be easily solved. Multifrontal factorization methods are useful as they can define a rich computational structure for the factorization based on a single analysis of the matrix's structure and then reuse this computational structure for the numerical factorization of the various matrices in the sequence. Furthermore, the computational structures defined by a multifrontal method together with their use of dense submatrices allows parallelism to be exploited in the factorization process.

However, as the values of matrix entries change across matrices in the sequence, entries that were once numerically acceptable as pivots may become no longer acceptable for subsequent matrices in the sequence. Techniques that recover from the loss of these anticipated pivots without having to abandon the previously defined computational structure can significantly improve overall execution time.

Until recently all multifrontal methods assumed a symmetric-pattern matrix which results in a corresponding computational structure that is a tree (or forest) [1, 5, 6, 7, 8, 18]. This simplified structure allows for a simple lost pivot recovery mechanism. Recently, Davis and Duff have introduced a new multifrontal method that relaxes the assumption of a symmetric pattern [3] and has demonstrated impressive performance advantages over previous methods [19]. In addition to reducing computational requirements, this method exposes considerable parallelism [14, 15]. With this unsymmetric pattern multifrontal method however, the resulting computational structure

* 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 the National Science Foundation (ASC-9111263, DMS-9223088), and by Cray Research, Inc., through the allocation of supercomputer resources.

is no longer a tree but rather a more generalized directed acyclic graph (DAG). The relationships between computational units in this structure are more complicated and require more sophisticated techniques for the recovery of lost pivots.

The focus of this paper is the theoretical development of a lost pivot recovery mechanism for the unsymmetric pattern multifrontal method. The resulting mechanism is implemented within the context of a distributed memory, parallel refactorization code based on the unsymmetric pattern multifrontal method.

Section 2 discusses the basic concepts behind multifrontal methods with the differences between the new unsymmetric pattern method and classical symmetric pattern methods highlighted. Section 3 identifies the lost pivot recovery issues that are unique to the unsymmetric pattern method and summarizes how these issues are addressed. Section 4 provides the theoretical foundation for the lost pivot recovery mechanism and Section 5 summarizes the implementation. Section 6 provides some key performance results in terms of both sequential execution times and the effects of lost pivot recovery on exploitable parallelism.

2. Multifrontal Concepts. Multifrontal methods for sparse matrix factorization decompose the sparse matrix into a set of overlapping dense submatrices called *frontal matrices*. Each frontal matrix is partially factorized by one or more pivots. Entries in the unfactorized portion of the frontal matrix (called the *contribution block*) must be uniquely assembled (added) into subsequent frontal matrices. An *assembly directed acyclic graph (DAG)* is used to define the computational structure with nodes 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, 18]. 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.

In the most general case, there are four types of relationships that can exist between frontal matrices with these relationships defined by the pattern of nonzeros in the matrix. These relationships are shown in Figure 2.1 below. The notation P_i refers to the *pivot block* of the ith frontal matrix. It is a submatrix formed by the pivot rows and columns within a single 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. 2.1. *Possible Relationships between Frontal Matrices*

These relationships imply certain inclusion properties between the frontal matrices' patterns. Specifically, an L relationship implies that the subsequent (parent)

frontal matrix's pattern of columns will include all of the non-pivotal columns of the preceding (child) frontal matrix's contribution block. Likewise, a U relationship implies a similar inclusion of the child's contribution block pattern of rows in the parent's pattern of rows. The LU relationship implies the inclusion properties of both the L and U relationships and allows the entire child's contribution block to be absorbed within the parent. The inclusion properties of a single LU parent, as is always the case with a symmetric pattern multifrontal method, allows any failed pivots in the child frontal matrix to be shifted to the LU parent and simply handled as an extension of the number of pivots associated with that frontal matrix. There is no extension of the parent's contribution block and no impacts on the assembly tree. The situation becomes significantly more complicated when there are L and/or U relationships to other frontal matrices.

3. Lost Pivot Recovery Issues. Multifrontal methods can address lost pivot recovery with three distinct strategies. First, *avoidance* tries to preclude the need for recovering lost pivots by relaxing the criteria used to judge potential pivots as being numerically acceptable. This is done via the concept of *threshold pivoting* [6]. Threshold pivoting requires the magnitude of the pivot entry's value to be greater than a threshold parameter (μ) times the magnitude of the largest entry in the column. The threshold parameter is set between 1 and 0. Lost pivots can be avoided by setting μ closer to one for the initial matrix in the sequence, and then lowering it to improve the potential for acceptable pivots in subsequent matrices in the sequence.

The second strategy, *intra-frontal matrix recovery*, tries to recover from lost pivots using only permutations confined to the pivot block of the frontal matrix. Such permutations have no effects on the assembly DAG or composition of other frontal matrices.

The problem with avoidance and intra-frontal matrix recovery is that they are not guaranteed to work. Thus, there is a need for the third strategy which is *inter-frontal matrix recovery*. Inter-frontal matrix recovery uses symmetric permutations to the entire matrix to shift lost pivot rows and columns to subsequent frontal matrices. In the case of a symmetric pattern multifrontal method, the impacts are limited to the single LU parent. With the unsymmetric pattern multifrontal method, the impacts are more significant and wide spread. Specifically, all L and U parent frontal matrices that occur prior to a specified recovery matrix in the assembly DAG must be expanded by portions of the failed pivot row(s) and column(s). The specified recovery matrix must be an LU ancestor of the failed frontal matrix where an *LU ancestor* is defined as a subsequent frontal matrix to which there is a path of strictly L and/or LU edges (L path) and a path of strictly U and/or LU edges (U path) from the frontal matrix containing the failed pivots. This recovery matrix will absorb any remaining portion of the lost pivot row(s) and column(s), just as a single LU parent would do in the symmetric pattern methods. The intervening frontal matrices (between the failed frontal matrix and its recovery matrix) that lie on an L path are expanded by the lost pivot column(s) and those on a U path are expanded by the lost pivot row(s).

Figures 3.1, 3.2, and 3.3 illustrate inter-frontal matrix recovery within the unsymmetric pattern multifrontal method. In Figure 3.1 the entire matrix is shown with rows and columns labeled with capital letters. Corresponding frontal matrices are identified with the row/column label of their first pivot entry. Frontal matrix A has two pivots with the (A_g, A_g) entry being an acceptable pivot and the (A_l, A_l) entry being a lost pivot that must be symmetrically permuted to expand the pivot block of the recovery frontal matrix D. Figure 3.2 shows the corresponding assembly DAG

with the inter-frontal matrix relationships show as labeled edges and Figure 3.3 illustrates the specific frontal matrices. Notice that the symmetric permutation shown in Figure 3.1 for inter-frontal matrix recovery causes new nonzero entries to be created. This situation is called *fill-in* and is indicated by a '*'. This fill-in is accounted for by the column extension to frontal matrix B that lies on an L path from A and the row extension to frontal matrix C that lies on a U path from A.

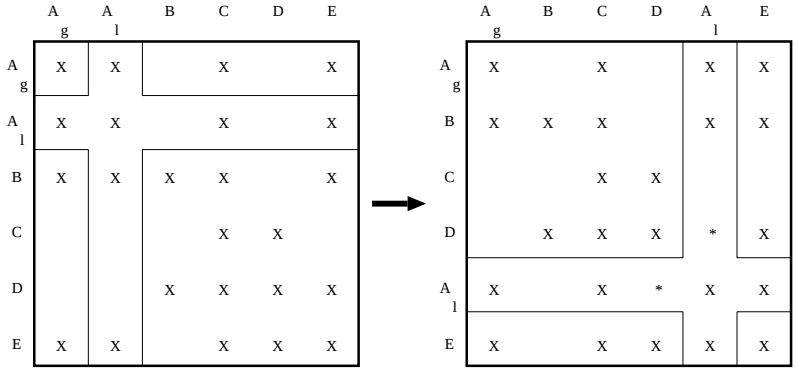

FIG. 3.1. *Sparse Matrix with Lost Pivots*

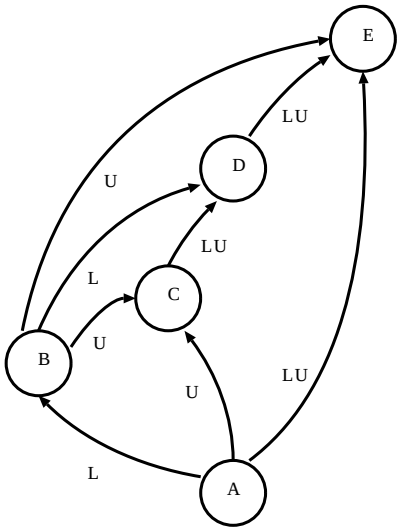

FIG. 3.2. *Sample Assembly DAG*

The theoretical foundation established in the next section will show that the inter-frontal recovery mechanisms just summarized will handle any lost pivot recovery necessary. Furthermore, the extent of fill-in due to lost pivot recovery will be fully defined and the entire process shown to be achievable without modifying the structure of the assembly DAG, which is especially important for distributed memory parallel implementations where static scheduling and allocations are used.

4. Theoretical Foundation. The inter-frontal matrix recovery strategy just summarized is fairly simple in concept but complicated in realization. Specifically, the

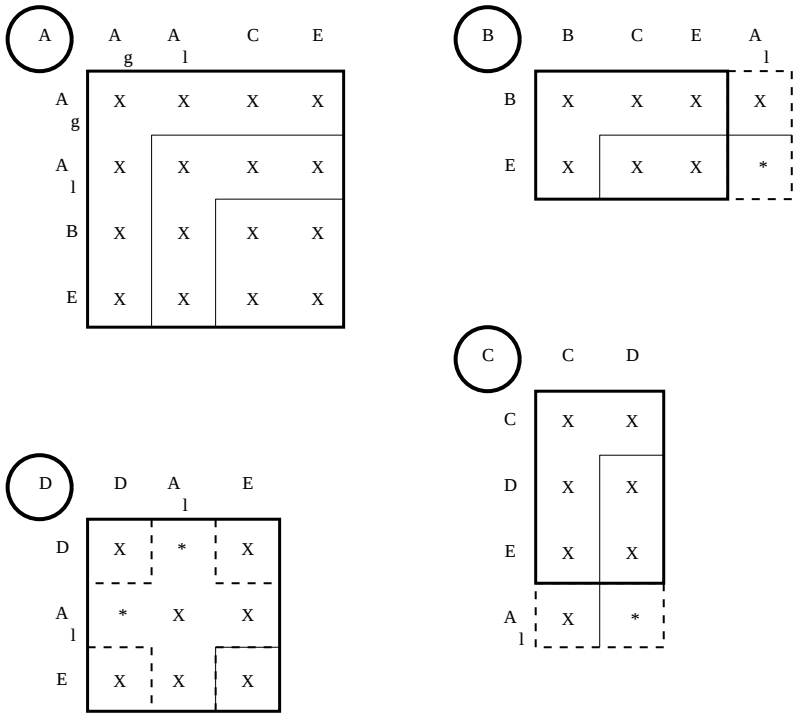

FIG. 3.3. *Frontal Matrices Extended by Lost Pivot Recovery*

scope of the lost pivot recovery effects (to include new fill-in) must be fully defined. Impacts on the assembly DAG's structure must be understood. Furthermore, the adequacy of the strategy to handle multiple interacting recoveries and repeated pivot failures must be established.

The theoretical foundation established in this section does this by first defining some necessary notation and foundational theorems that describe the relationships between frontal matrices' patterns that are imposed by relationships (edges and paths) in the assembly DAG. These results are then used to address and resolve the larger issues of sufficiency and required detailed mechanisms.

4.1. Definitions and Notation. The ordered set, $\mathcal{F}$, is defined as the set of all frontal matrices in a multifrontal formulation for a particular sparse matrix factorization. Each element of this set represents a distinct frontal matrix and an equivalence relation is established between these frontal matrices and the original position of their first pivot in the pivot ordering (assuming the necessary permutations have been applied to put the matrix into the pivot ordering used by the initial analyze-factorize routine [3]). Hence the ordering of the pivot entries provides the ordering for the set $\mathcal{F}$ and a frontal matrix's ID is defined to be the pivot column/row index of the first pivot within the frontal matrix.

When block upper triangular form is in use, the notation $\mathcal{F}_i$ will represent the subset of $\mathcal{F}$ that contains all of the frontal matrices in the ith diagonal block. Within the context of the assembly DAG, $\mathcal{F}_i$ will define the node set of a sub-DAG that is in no way connected to any other nodes in the assembly DAG (nodes not in $\mathcal{F}_i$).

Frontal matrices will typically be represented by capital letters. A special nota-

tion is reserved for frontal matrices that are designated as recovery frontal matrices. Specifically, for $A \in \mathcal{F}$, the notation R_A (where $R_A \in \mathcal{F}$) represents the recovery frontal matrix of A. As mentioned earlier, R_A will be the first LU ancestor of A.

As the set $\mathcal{F}$ is an ordered set, the less than ($<$), equal to ($=$), and greater than ($>$) relations are well defined and trichotomy insures that for any two frontal matrices A and $B \in \mathcal{F}$ only one of these relations applies and that if $A = B$, then A and B represent the same frontal matrix.

Within a particular frontal matrix, the specific components (submatrices) will frequently need to be referenced. This will be done using the same notation found in Davis and Duff's original work [3]. Specifically, $\mathcal{L}_A$ refers to the set of row indices that define the rows from the overall matrix that make up the frontal matrix $A \in \mathcal{F}$. Likewise, $\mathcal{U}_A$ refers to the set of column indices that define the columns of the entire matrix. (Again, the indices of the overall matrix assume the necessary permutations have been applied to establish the initial pivot ordering as determined by the analyze-factorize algorithm). The subset $\mathcal{L}'_A \subseteq \mathcal{L}_A$ is the set of row indices that define the pivot rows of the frontal matrix A. Prior to any lost pivot recovery, $\mathcal{L}'_A$ will be equal to $\{A, A+1, \ldots, A+(k_A-1)\}$ where A is the row and column index of the first pivot of A and k_A is the number of potential pivots in A. The subset $\mathcal{L}''_A \subseteq \mathcal{L}_A$ contains all the row indices of the nonpivotal rows of A. These nonpivotal rows define the rows of the frontal matrix's contribution block and hence define the L edges from the frontal matrix A to L parent frontal matrices. In a similar fashion, $\mathcal{U}'_A \subseteq \mathcal{U}_A$ defines the indices of the pivot columns of A and will initially be equal to $\mathcal{L}'_A$. The set $\mathcal{U}''_A \subseteq \mathcal{U}_A$ defines the indices of the nonpivotal columns of A and hence also defines the columns of the contribution block and the U edges to U parents of A.

Furthermore, sometimes subsets of the $\mathcal{L}''$ and $\mathcal{U}''$ patterns will need to be referenced. Specifically, the notation $\mathcal{L}''_A(B)$ will refer to the subset of frontal matrix A's pattern of nonpivotal rows that are pivotal rows for frontal matrices B and greater. Likewise, $\mathcal{U}''_A(B)$ refers to the nonpivotal columns of frontal matrix A that are pivot columns for frontal matrices B and greater.

When discussing the failed pivots of a frontal matrix, the notation $\mathcal{L}'_{A_{lost}}$ and $\mathcal{U}'_{A_{lost}}$ will be used to define the row and columns indices, respectively, of the lost pivots within frontal matrix A. The notation, $\mathcal{L}'_{A_{good}}$ and $\mathcal{U}'_{A_{good}}$ will be used to represent the row and column indices of the acceptable pivots within the frontal matrix A. These patterns accommodate any unsymmetric permutations that may have taken place within the frontal matrix's pivot block.

Relations between frontal matrices reflect the need to pass the contributions of one frontal matrix to another frontal matrix where the contributions will be applied to entries in the destination's pivot row and/or pivot columns. There are two types of such relations. An L edge from frontal matrix A to B occurs when $\mathcal{L}''_A \cap \mathcal{L}'_B \neq \emptyset$. That is, A produces contributions that need to be assembled into B's pivot rows. This L edge is represented by the notation $A \xrightarrow{L} B$. In such a relation, A is said to be the L child of B and B the L parent of A. Similarly, a U edge from A to B exists if $\mathcal{U}''_A \cap \mathcal{U}'_B \neq \emptyset$ and represents the fact that A produces contributions that need to be assembled into the pivot columns of B. This edge is represented by the notation $A \xrightarrow{U} B$ and A is called the U child of B and B is the U parent of A. When L and U edges both exist from A to B, an LU edge is said to exist and is shown as $A \xrightarrow{LU} B$ with A the LU child of B and B the LU parent of A.

Notice that due to the ordering scheme imposed on $\mathcal{F}$ and the fact that contributions are always passed to frontal matrices that appear later in the pivot ordering,

all edges from a frontal matrix A to B will be such that $A < B$.

An L path exists between frontal matrices A and B if and only if there is a path from A to B consisting of all L and/or LU edges. In this case B is called an L ancestor of A and A is an L predecessor of B. When such a path contains at least one edge, the notation $A \overset{L^+}{\rightsquigarrow} B$ is used to represent the path. Correspondingly, a U path from A to B exists if and only if there is a path from A to B of all U and/or LU edges with B the U ancestor of A and A the U predecessor of B. Such a path is denoted by $A \overset{U^+}{\rightsquigarrow} B$. An LU ancestor of a frontal matrix A is another frontal matrix such that there is both an L path and a U path from A to this other frontal matrix. The L and U paths may or may not share frontal matrices as path nodes. Such relationships are denoted by $A \overset{LU^+}{\rightsquigarrow} B$. Furthermore, in some situations, paths of zero or more edges will need to be considered (where a zero length path exists only if both the source and destination frontal matrices are the same). Such paths are denoted by $A \overset{L^*}{\rightsquigarrow} B$ and $A \overset{U^*}{\rightsquigarrow} B$.

The definitions of L and U edges represent the true relationships between frontal matrices and thus the set of all of these edges is called the true edge set and denoted: $\mathcal{E}_T$. As $\mathcal{E}_T$ really consists of two types of edges (L edges and U edges), it can be decomposed into two disjoint subsets: $\mathcal{E}_T^L$, which contains only the L edges, and $\mathcal{E}_T^U$, which contains only the U edges. This is similar to a formulation of elimination DAGs done by Gilbert and Liu [12] and Eisenstat and Liu [9, 10].

The edge set used to define the assembly DAG is called the assembly edge set and denoted by $\mathcal{E}_A$. It differs from the true edge set, $\mathcal{E}_T$. The $\mathcal{E}_A$ edge set is generated by the combined analyze-factorize algorithm (UMFPACK [2, 3]) for the first matrix in the sequence. There is an edge from A to B in $\mathcal{E}_A$ if any contributions were assembled from frontal matrix A into frontal matrix B. An assembly of one or more rows from A is done in UMFPACK whenever $\mathcal{U}_A''(B) \subseteq \mathcal{U}_B$ and $\mathcal{L}_A''(B) \cup \mathcal{L}_B$ is not empty. Similarly, an assembly of one or more columns from A is done in UMFPACK whenever $\mathcal{L}_A''(B) \subseteq \mathcal{L}_B$ and $\mathcal{U}_A''(B) \cup \mathcal{U}_B$ is not empty. The entire (remaining) contribution block of A is assembled into B if both $\mathcal{U}_A''(B) \subseteq \mathcal{U}_B$ and $\mathcal{L}_A''(B) \subseteq \mathcal{L}_B$ hold. In general, $\mathcal{E}_A$ is a transitive reduction of $\mathcal{E}_T$ and it is frequently useful to think of it that way. However, unless the matrix is in block triangular form, there may exist edges in $\mathcal{E}_T$ that are not in $\mathcal{E}_A$.

With the preceding notation and definitions established, development of the necessary theoretical results may commence. The theory simplifies if an assumption of block triangular form can be made and such an assumption is made in this article. The development starts by stating a key result of block triangular form and then proceeds to some foundational theorems and the rest of the lost pivot recovery results.

4.2. Block Triangular Form. Block Triangular Form is significant because it can expose a high level parallelism between assembly sub-DAGS, reduce overall computations (as off-diagonal blocks need not be factorized), and simplify lost pivot recovery [13]. The advantages in terms of lost pivot recovery occur as $\mathcal{L}'' \neq \emptyset$ and $\mathcal{U}'' \neq \emptyset$ for all frontal matrices except the last frontal matrix in each diagonal block $\mathcal{F}_i$. A proof can be found in Hadfield's dissertation [13].

4.3. Foundational Theorems. Three theorems and two corollaries form the foundation upon which most of the lost pivot recovery results are built. These are provided below:

THEOREM 4.1 (L ANCESTOR FILL-IN). *If* $A \overset{L^+}{\rightsquigarrow} B$ *then* $\mathcal{U}_A''(B) \subseteq \mathcal{U}_B$.

Proof: The proof of this theorem follows an induction argument based on L edges of the L path from A to B.

BASIS: For $A \xrightarrow{L} B$ (direct L edge), the entries a_{i_1,j_1}, a_{i_2,j_1}, and a_{i_1,j_2} must be assumed nonzero for some $i_1 \in \mathcal{L}'_A$, $i_2 \in \mathcal{L}'_B \cap \mathcal{L}''_A$, $j_1 \in \mathcal{U}'_A$, and for all $j_2 \in \mathcal{U}''_A$. Then, as frontal matrix A's contribution block must be assumed nonzero, all of the a_{i_2,j_2} entries are nonzero, which implies $\mathcal{U}''_A(B) \subseteq \mathcal{U}_B$.

INDUCTION: If $A \overset{L^+}{\rightsquigarrow} C \xrightarrow{L} B$, then by inductive hypothesis $\mathcal{U}''_A(C) \subseteq \mathcal{U}_C$. Furthermore, as $C \xrightarrow{L} B$, the argument of the induction basis provides $\mathcal{U}''_C(B) \subseteq \mathcal{U}_B$ and an edge from C to B implies that $C < B$. Thus, the following set of inclusions are established

$$\mathcal{U}''_A(B) \subseteq \mathcal{U}''_C(B) \subseteq \mathcal{U}_B.\square$$

COROLLARY 4.2. *If $A \overset{L^*}{\rightsquigarrow} B$ then $\mathcal{U}''_A(B) \subseteq \mathcal{U}_B$.*

This corollary simply adds the null edge case, which is true via the definition $\mathcal{U}''_A \subset \mathcal{U}_A$.

THEOREM 4.3 (U ANCESTOR FILL-IN). *If $A \overset{U^+}{\rightsquigarrow} B$ then $\mathcal{L}''_A(B) \subseteq \mathcal{L}_B$.*

Proof: Follows directly as the transpose of the proof of Theorem 4.1.

COROLLARY 4.4. *If $A \overset{U^*}{\rightsquigarrow} B$ then $\mathcal{L}''_A(B) \subseteq \mathcal{L}_B$.*

This corollary simply adds the null edge case, which is true via the definition $\mathcal{L}''_A \subset \mathcal{L}_A$.

THEOREM 4.5 (PATHS DUE TO FILL-IN). *Assume that $A \overset{L^+}{\rightsquigarrow} B$ and $A \overset{U^+}{\rightsquigarrow} C$, then (a) if $B < C$ there is an U path $B \overset{U^+}{\rightsquigarrow} C$ or (b) if $B > C$ there is a L path $C \overset{L^+}{\rightsquigarrow} B$.*

Proof: The proof of this theorem is an induction argument based on the L and U paths. Furthermore, the basis is strengthened to include a path of zero edges which is needed to facilitate the proof.

BASIS: Assume $A \xrightarrow{L} B$ and $A \overset{U^*}{\rightsquigarrow} C$. Then consider the case of $B < C$ and all C_1 and C_2 such that

$$A \overset{U^*}{\rightsquigarrow} C_1 \xrightarrow{U} C_2 \overset{U^*}{\rightsquigarrow} C \text{ and } C_1 < B < C_2.$$

(Note that C_1 could be A and C_2 could be C and as $B < C$ there must be at least one edge in $A \overset{U^*}{\rightsquigarrow} C$). As $A \overset{U^*}{\rightsquigarrow} C_1$, the inclusion: $\mathcal{L}''_A(C_1) \subseteq \mathcal{L}_{C_1}$, holds per Corollary 4.4 and together with $C_1 < B$ and $A \xrightarrow{L} B$ implies that $B \in \mathcal{L}''_{C_1}$. This implies the existence of the edge $C_1 \xrightarrow{L} B$, which together with $C_1 < B$ and Corollary 4.2 implies $\mathcal{U}''_{C_1}(B) \subseteq \mathcal{U}_B$. Due to $C_1 \xrightarrow{U} C_2$ and $B < C_2$, $C_2 \in \mathcal{U}''_{C_1}(B) \subseteq \mathcal{U}_B$, which establishes the existence of the edge $B \xrightarrow{U} C_2$ and as $C_2 \overset{U^*}{\rightsquigarrow} C$, the result $B \overset{U^+}{\rightsquigarrow} C$ follows.

If $B > C$ then $A \overset{U^*}{\rightsquigarrow} C$ implies $\mathcal{L}''_A(C) \subseteq \mathcal{L}_C$ by Corollary 4.4. (Note that A could be equal to C here). Since $A \xrightarrow{L} B$ and $B > C$, the results $B \in \mathcal{L}''_C$ and $C \xrightarrow{L} B$ follow.

Furthermore, the transpose of this argument establishes that the results also hold for $A \overset{L^*}{\rightsquigarrow} B$ and $A \xrightarrow{U} C$.

INDUCTION: For the inductive step, the argument assumes paths of length one or more as follows: $A \overset{L^+}{\rightsquigarrow} B$ and $A \overset{U^+}{\rightsquigarrow} C$. First the case of $B < C$ is considered with any

three frontal matrices: C_1, C_2, and B_1 such $C_1 < B_1 < C_2$ and $A \overset{U^*}{\rightsquigarrow} C_1 \overset{U}{\rightarrow} C_2 \overset{U^*}{\rightsquigarrow} C$ and $A \overset{L^+}{\rightsquigarrow} B_1 \overset{L^*}{\rightsquigarrow} B$. By inductive hypothesis, the edge $C_1 \overset{L}{\rightarrow} B_1$ exists. This result, together with Theorem 4.1, implies that $\mathcal{U}''_{C_1}(B_1) \subseteq \mathcal{U}_{B_1}$. Hence, as $C_1 \overset{U}{\rightarrow} C_2$ and $B_1 < C_2$, one finds that $C_2 \in \mathcal{U}''_{B_1}$ and the edge $B_1 \overset{U}{\rightarrow} C_2$ exists, which establishes the path $B_1 \overset{U^+}{\rightsquigarrow} C$. As none of the selection criteria on C_1, B_1, or C_2 preclude $B_1 = B$, these results include $B \overset{U^+}{\rightsquigarrow} C$.

The transpose of the argument just provided establishes that if $B > C$ the results $C \overset{L^+}{\rightsquigarrow} B$ also hold. □

THEOREM 4.6 (ALWAYS AN LU ANCESTOR). *Assuming block upper triangular form, every frontal matrix will have either an LU ancestor or no ancestors at all.*

Proof: Assume that there exists a frontal matrix A with ancestors but no LU ancestor. The assumption of block triangular form insures that $\mathcal{L}'' \neq \emptyset$ and $\mathcal{U}'' \neq \emptyset$; hence, the frontal matrix must have edges to both L and U parents in $\mathcal{E}_T$. Likewise, each L parent must have L edges to later frontal matrices in $\mathcal{F}_i$ unless the parent is the last frontal matrix in the block. As each $\mathcal{F}_i$ has only a finite number of frontal matrices in it, there will be an L path from A to the last frontal matrix in the block. In the same fashion, an argument based on the U edges that follows the same logic will establish the existence of a U path from A to the last frontal matrix in the block. Hence, there are both L and U paths from A to the last frontal matrix in the block and thus this frontal matrix is an LU ancestor of A. □

With this foundational theory established, construction may commence on the specific lost pivot recovery theory.

4.4. Fill-In due to Recovery. In this section, theoretical results are developed to describe how and where new fill-in is produced due to the recovery of a frontal matrix's lost pivot rows and columns. Specifically, new fill-in resulting from the recovery of lost pivots occurs in the lost pivot rows and columns. New fill-in the lost pivot columns is limited in scope to the row patterns of L ancestors. New fill-in in the lost pivot rows is limited in scope to the column patterns of U ancestors. Furthermore, fill-in in a particular row of the lost pivot columns (or column of the lost pivot rows) can result from several frontal matrices. This will have significant impacts on the distributed memory implementation to be described in the next section as the fill-in contributions become decentralized and must be combined in a fashion that eliminates duplicates and insures all distinct contributions are included.

A final, yet very significant, result is that the first LU ancestor will be able to absorb the remaining nonzero rows of the lost pivot columns and nonzero columns of the lost pivot rows by simply extending its pivot block (defined by its $\mathcal{L}'$ and $\mathcal{U}'$ patterns) to include the lost pivot rows and columns. Hence, the failed frontal matrix can be completely recovered by its first LU ancestor and will have no additional direct effects on the remaining frontal matrices and the assembly DAG. If any pivots still fail in the LU ancestor, they can be treated as pivot failures of the LU ancestor (as discussed in Section 4.7, below).

Throughout the theorems and corollaries of this section, the reader should recall that the recovery frontal matrix for any particular failed frontal matrix is defined as the first LU ancestor, which exists for all frontal matrices except for those that occur last in a diagonal block. The last frontal matrices are always square with $\mathcal{L} = \mathcal{L}'$ and $\mathcal{U} = \mathcal{U}'$. Thus any lost pivots in these last frontal matrices indicate the overall matrix is singular.

THEOREM 4.7 (L ANCESTOR RECOVERY FILL-IN). *For $A, B, R_A \in \mathcal{F}_i$ for some i such that A contains failed pivots, $A \overset{L^+}{\rightsquigarrow} B$, R_A is the recovery frontal matrix of A, and $B < R_A$, there will be fill-in in the intersection of the rows of $\mathcal{L}_B$ and columns of $\mathcal{U}'_{A_{lost}}$.*

Proof: The proof follows an induction argument on the L edges of the L path from A to B.

BASIS: If $A \overset{L}{\rightarrow} B$ (direct L edge), then $a_{i,j}$ is assumed nonzero for all $i \in \mathcal{L}'_B \bigcap \mathcal{L}''_A$ and all $j \in \mathcal{U}'_A$ including $j \in \mathcal{U}'_{A_{lost}}$. Thus, when the lost pivot columns ($\mathcal{U}'_{A_{lost}}$) are permuted to extend the pivot block of R_A, these assumed nonzero entries will cause an extension of the $\mathcal{U}''_B$ by the columns of $\mathcal{U}'_{A_{lost}}$ and the resulting fill-in will cause the intersection of the $\mathcal{L}_B$ rows and the $\mathcal{U}'_{A_{lost}}$ columns to become nonzero.

INDUCTION: If $A \overset{L^+}{\rightsquigarrow} B$ (L path with more than one edge) with $B < R_A$, then for $C \in \mathcal{F}_i$ such that $A \overset{L^+}{\rightsquigarrow} C \overset{L}{\rightarrow} B$, the inductive hypothesis provides that $\mathcal{U}''_C$ has been extended by $\mathcal{U}'_{A_{lost}}$. Since $C \overset{L}{\rightarrow} B$ and $B < R_A$, Theorem 4.1 provides that $\mathcal{U}'_{A_{lost}} \subseteq \mathcal{U}''_C(B) \subseteq \mathcal{U}_B$, which means frontal matrix B must be extended by the lost pivot columns. Hence, there will be new fill-in in the intersection of the $\mathcal{L}_B$ rows and the columns of $\mathcal{U}'_{A_{lost}}$. □

COROLLARY 4.8. *The nonzero rows of the $\mathcal{U}'_{A_{lost}}$ columns that are contained in $\mathcal{L}'_B$ pattern for B as defined in Theorem 4.7 will be fully absorbed by the frontal matrix B and will play no further role in the recovery of A.*

This is true as these entries go into the pivot rows of B and will be now considered as part of the frontal matrix B. If good pivots are found in these rows, they will become part of the U factor associated with frontal matrix B, else they will be handled as B's failed pivot rows.

COROLLARY 4.9. *For $B, C \in \mathcal{F}_i$ such that $A \overset{L^+}{\rightsquigarrow} B$ and $A \overset{L^+}{\rightsquigarrow} C$ with $B < R_A$ and $C < R_A$ and R_A the recovery matrix for A, there is the possibility that either $\mathcal{L}_B \bigcap \mathcal{L}''_C \neq \emptyset$ and/or $\mathcal{L}''_B \bigcap \mathcal{L}_C \neq \emptyset$. Thus, there may be multiple contributions made to a particular row of the $\mathcal{U}'_{A_{lost}}$ columns as a result of the recovery of A.*

THEOREM 4.10 (U ANCESTOR RECOVERY FILL-IN). *For $A, B, R_A \in \mathcal{F}_i$ such that A contains failed pivots, $A \overset{U^+}{\rightsquigarrow} B$, R_A is the recovery frontal matrix of A, and $B < R_A$, there will be fill-in in the intersection of the columns of $\mathcal{U}_B$ and the rows of $\mathcal{L}'_{A_{lost}}$.*

Proof: Follows the transpose of the argument for Theorem 4.7. Furthermore, the transpose of Corollaries 4.8 and 4.9 also follow.

COROLLARY 4.11. *The nonzero columns of the $\mathcal{L}'_{A_{lost}}$ rows that are contained in $\mathcal{U}'_B$ pattern for B as defined in Theorem 4.10 will be fully absorbed by the frontal matrix B and will play no further role in the recovery of A.*

COROLLARY 4.12. *For $B, C \in \mathcal{F}_i$ such that $A \overset{U^+}{\rightsquigarrow} B$ and $A \overset{U^+}{\rightsquigarrow} C$ with $B < R_A$ and $C < R_A$ and R_A the recovery matrix for A, there is the possibility that either $\mathcal{U}_B \bigcap \mathcal{U}''_C \neq \emptyset$ and/or $\mathcal{U}''_B \bigcap \mathcal{U}_C \neq \emptyset$. Thus, there may be multiple contributions made to a particular column of the $\mathcal{L}'_{A_{lost}}$ rows as a result of the recovery of A.*

THEOREM 4.13 (LIMITS OF LOST PIVOT COLUMN FILL-IN). *The only fill-in in the lost pivot columns of a frontal matrix A is due to L ancestors of A.*

Proof: Assume $a_{i,k}$ is new fill-in created due to the recovery of A where $k \in \mathcal{U}'_{A_{lost}}$ but $i \notin \mathcal{L}_B$ for any $B \in \mathcal{F}_i$ such that $A \overset{L^+}{\rightsquigarrow} B$. Then, there must exist a j such that $A < j < R_A$ such that $a_{i,j} \neq 0$, and $a_{j,j} \neq 0$, and $a_{j,k} \neq 0$ otherwise the

fill-in would not have occurred. If $a_{j,k} \neq 0$ regardless of the pivot failure in A then the edge $A \xrightarrow{L} J$ with $j \in \mathcal{L}'_J$ must exist, which raises a contradiction. Otherwise, if $a_{j,k}$ became nonzero due to fill-in from the recovery of A then this argument is inductively applied to the $a_{j,k}$ entry. The induction is terminated by either finding an entry $a_{m,k}$ in column k that is nonzero regardless of the recovery (which results in the contradiction previously described) or if no such entry is found the fill-in could not have occurred (another contradiction of the assumptions). □

THEOREM 4.14 (LIMITS OF LOST PIVOT ROW FILL-IN). *The only fill-in in the lost pivot rows of a frontal matrix A is due to U ancestors of A.*

Proof: Follows the transpose of the argument for Theorem 4.13.

THEOREM 4.15 (RECOVERY MATRIX EXTENSIONS). *The recovery frontal matrix R_A for a failed frontal matrix A needs only to have its pivot block (defined by $\mathcal{L}'_{R_A}$ and $\mathcal{U}'_{R_A}$) extended to accommodate the recovery of A.*

Proof: For any $B \in \mathcal{F}_i$ such that $A \overset{L^*}{\rightsquigarrow} B \overset{L^+}{\rightsquigarrow} R_A$, Theorem 4.1 provides that $\mathcal{U}''_B(R_A) \subseteq \mathcal{U}_{R_A}$. Similarly, for any $C \in \mathcal{F}_i$ such that $A \overset{U^*}{\rightsquigarrow} C \overset{U^+}{\rightsquigarrow} R_A$, Theorem 4.3 provides that $\mathcal{L}''_C(R_A) \subseteq \mathcal{L}_{R_A}$. As $\mathcal{U}''_B$ and $\mathcal{L}''_C$ have only been extended by $\mathcal{U}'_{A_{lost}}$ and $\mathcal{L}'_{A_{lost}}$, respectively, these are the only extensions needed to the recovery frontal matrix R_A. □

COROLLARY 4.16. *Any edges $A \xrightarrow{L} B$ and/or $A \xrightarrow{U} B$ such that $B > C$ and $A \overset{LU^+}{\rightsquigarrow} C$ are redundant due to the existence of corresponding L and U paths from C to B, respectively.*

The next theorem establishes that by using the first LU ancestor as the recovery matrix, all contributions to the lost pivot block by earlier ancestors are avoided. This result allows the implementation to preclude the multiple contribution handling for the lost pivot block portion of the lost pivot rows and columns. Corollaries 4.9 and 4.12 established the need for this processing for the rest of the lost pivot row and column entries.

THEOREM 4.17 (NO CONTRIBUTIONS TO LOST PIVOT BLOCK). *If the first LU ancestor B of A (that is, $A \overset{LU^+}{\rightsquigarrow} B$ with $A \overset{LU^+}{\rightsquigarrow} C$ for no $C < B$) is used for the recovery matrix for A (designated R_A), then there will be no contributions made to the lost pivot block of A by L or U ancestors of A.*

Proof: The lost pivot block of the failed frontal matrix A is defined by the rows in $\mathcal{L}'_{A_{lost}}$ and the columns in $\mathcal{U}'_{A_{lost}}$. By Theorems 4.13 the columns of $\mathcal{U}'_{A_{lost}}$ are only updated via L ancestors. By Theorem 4.14 the rows of $\mathcal{L}'_{A_{lost}}$ are only updated by U ancestors. Thus, for updates to the lost pivot block of A to occur, the updating frontal matrix must be both an L and a U ancestor of the failed frontal matrix. As the recovery matrix, R_A, is the first such LU ancestor and as the patterns $\mathcal{L}'_{R_A}$ and $\mathcal{U}'_{R_A}$ are extended by $\mathcal{L}'_{A_{lost}}$ and $\mathcal{U}'_{A_{lost}}$, respectively, Corollaries 4.8 and 4.11 provide that no further contributions to A's lost pivot block will be made as a result of the recovery of A. □

The practical results of the theory just developed are that the recovery of a failed frontal matrix can be facilitated by (a) extending all L ancestors that occur before the first LU ancestor by the lost pivot rows, (b) extending all U ancestors that occur before the first LU ancestor by the lost pivot columns, and (c) extending the pivot block of the recovery frontal matrix by the lost pivot rows and columns. Furthermore, L ancestors absorb the entries of the lost pivot columns that occur in the ancestor's pivot rows and, likewise, U ancestors absorb the entries of the lost pivot rows that

occur in the ancestor's pivot columns.

4.5. Impacts on Assembly DAG. One of the most significant features of these lost pivot recovery mechanisms is the minimal impact on the relationships between frontal matrices as defined by the edge set: $\mathcal{E}_T$. Specifically, since LU ancestors are available to serve as recovery matrices, no additional edges need be added to $\mathcal{E}_T$ as a result of the recovery.

THEOREM 4.18 (NO ADDITIONAL EDGES FOR LU ANCESTOR RECOVERIES). *Only redundant edges are created due to recovery fill-in when the recovery frontal matrix R_A is an LU ancestor of the failed frontal matrix A.*

Proof: Since $A \overset{LU^+}{\rightsquigarrow} R_A$, Theorem 4.3 provides that $\mathcal{L}''_A(R_A) \subseteq \mathcal{L}_{R_A}$. Furthermore, for each $B \in \mathcal{F}_i$ such that $A \overset{L^+}{\rightsquigarrow} B$ and $B < R_A$, Theorem 4.5 insures the existence of a U path: $B \overset{U^+}{\rightsquigarrow} R_A$. This path, together with Theorem 4.3, provides that $\mathcal{L}''_B(R_A) \subseteq \mathcal{L}_{R_A}$. By Theorem 4.13, this accounts for all the recovery fill-in in the pattern $\mathcal{L}_{R_A}$, so no new L edges are created from R_A. Furthermore, the transpose of this argument insures that no new U edges are created from R_A.

Next, the new edges to R_A are considered. If $A \overset{L^+}{\rightsquigarrow} B$ with $B < R_A$, then there will be potentially new fill-in in the $\mathcal{L}''_B$ rows of the $\mathcal{U}'_{A_{lost}}$ columns. But for each row $c \in \mathcal{L}''_B$, by definition, there exists an edge $B \overset{L}{\rightarrow} C$ where $c \in \mathcal{L}'_C$. Hence, the path $A \overset{L^+}{\rightsquigarrow} C$ exists and if $C < R_A$ then by Theorem 4.5 the path $C \overset{U^+}{\rightsquigarrow} R_A$ exists. Thus, if the new fill-in of row $c \in \mathcal{L}''_B$ creates a new edge $C \overset{U}{\rightarrow} R_A$, this edge will be redundant to the path $C \overset{U^+}{\rightsquigarrow} R_A$ that already exists. In a similar fashion and by a transpose of this argument, any new L edge will be redundant to an existing L path.□

Hence only redundant edges are created due to the recovery to the LU ancestor R_A and the edge set $\mathcal{E}_T$ still correctly defines the relationships between frontal matrices.

4.6. Effects of Multiple Recoveries. Up to this point, all the results established have focused on the effects of a single failed frontal matrix. Nothing, however, precludes the possibility of multiple failed frontal matrices. Furthermore, several failed frontal matrices may all affect a particular frontal matrix. If the failed frontal matrices are all L predecessors or all U predecessors of the affected frontal matrix then the results thus far provide for the appropriate recovery actions. However, if the affected frontal matrix is an L ancestor of one failed frontal matrix and the U ancestor of another, it will be expanded in both row and column patterns. The theory developed thus far fails to provide direction on how to handle the new contributions in the overlap of the new recovery columns and new recovery rows.

Furthermore, the row and column patterns of a frontal matrix's pivot block define its relationship to the preceding frontal matrices. Since this pivot block can be expanded due to the recovery of LU predecessors, it is necessary to complete all pivot block expansions before determining how to handle the recovery of other failed frontal matrices. Ordering the handling of recoveries by increasing topological order of their recovery matrices will facilitate both overlap resolution and the necessary prerequisite pivot block expansions.

THEOREM 4.19 (PATHS BETWEEN RECOVERY MATRICES). *For A, B, C, R_A, and $R_B \in \mathcal{F}_i$ with A and B failed frontal matrices, R_A and R_B their respective recovery matrices, $C < R_A$, $C < R_B$, and the $\mathcal{E}_T$ edge set, if $A \overset{L^+}{\rightsquigarrow} C$ and $B \overset{U^+}{\rightsquigarrow} C$, then one and only one of $R_A = R_B$, $R_A \overset{L^+}{\rightsquigarrow} R_B$, or $R_B \overset{U^+}{\rightsquigarrow} R_A$ will be true.*

Proof: Due to the definition of the $\mathcal{E}_T$ edge set and that of the recovery matrices, the paths $A \overset{LU^+}{\rightsquigarrow} R_A$ and $B \overset{LU^+}{\rightsquigarrow} R_B$ exist. Thus, as $B \overset{U^+}{\rightsquigarrow} C$, $B \overset{L^+}{\rightsquigarrow} R_B$, and $C < R_B$, Theorem 4.5 provides the existence of $C \overset{L^+}{\rightsquigarrow} R_B$. Combined with the assumption of $A \overset{L^+}{\rightsquigarrow} C$, the path $A \overset{L^+}{\rightsquigarrow} R_B$ is established. But, the existence of $A \overset{U^+}{\rightsquigarrow} R_A$ is also known, so by Theorem 4.5 and the trichotomy of the ordered set $\mathcal{F}$ either (a) $R_A < R_B$ and $R_A \overset{L^+}{\rightsquigarrow} R_B$, (b) $R_B < R_A$ and $R_B \overset{U^+}{\rightsquigarrow} R_A$, or (c) $R_A = R_B$. □

Now that a relationship between the recovery matrices of the failed frontal matrices has been established by Theorem 4.19, the handling of the overlap area contributions of the multiply affected frontal matrix can be defined.

THEOREM 4.20 (OVERLAP AREA CONTRIBUTION HANDLING). *If A, $B \in \mathcal{F}_i$ are failed frontal matrices with R_A and R_B their respective recovery matrices and $A \overset{L^+}{\rightsquigarrow} C$ and $B \overset{U^+}{\rightsquigarrow} C$ (but no U path from A to C or L path from B to C), then the contributions made by C in the portion of the extended contribution block defined by the rows of $\mathcal{L}'_{B_{lost}}$ and the columns of $\mathcal{U}'_{A_{lost}}$ need to be incorporated into the pivot rows/columns of either R_A or R_B whichever occurs first in the assembly DAG.*

Proof: If A and B have the same recovery matrix then there is no issue to resolve. If $R_A < R_B$, then Theorem 4.19 establishes the existence of a path from R_A to R_B and Corollaries 4.8 and 4.11 require that the overlap area contributions be absorbed by R_A. Likewise, if $R_B < R_A$, then the overlap area contributions will be absorbed by R_B. □

COROLLARY 4.21. *Resolution of the overlap area contribution handling described in Theorem 4.20 can also be achieved by absorbing the contributions in whichever recovery frontal matrix (R_A or R_B) has the lower topological level.*

This latest corollary provides a way of organizing multiple recoveries that will simplify the implementation (to be discussed in detail later).

The second major issue of this section is to define the need to complete all the pivot block expansions due to failed LU predecessors prior to accommodating any additional effects due to other failed frontal matrices.

THEOREM 4.22 (PIVOT BLOCK RECOVERIES FIRST). *When multiple failed frontal matrices affect a particular frontal matrix, it is necessary that the effects of recoveries for which the current frontal matrix is the recovery frontal matrix be resolved before any other recoveries are addressed.*

Proof: Corollaries 4.8 and 4.11 establish the importance of an L ancestor's pivot block pattern of rows ($\mathcal{L}'$) and a U ancestor's pivot block pattern of columns ($\mathcal{U}'$) in determining how a failed frontal matrix's recovery will affect the ancestor frontal matrix. When a frontal matrix is the LU ancestor recovery matrix for a failed frontal matrix, its $\mathcal{L}'$ and $\mathcal{U}'$ patterns will be expanded to accommodate the lost pivots. Since the recoveries of other failed frontal matrices will depend on these patterns, all of the effects of failed frontal matrices that the current frontal matrix recovers must be first resolved before other recovery effects can be addressed. Furthermore, as the relationship between the failed frontal matrix and its recovery matrix is well defined a priori, use of the recovery frontal matrix's pivot block pattern is not necessary in recovering these failed frontal matrices. □

This theorem will be combined with Corollary 4.21 and the results of the next section to prescribe the ordering for resolving the effects of multiple failed frontal matrices.

Besides the possibility of distinct frontal matrices having their original anticipated pivots fail, there is the possibility that these same pivots will fail once they have been

absorbed into their recovery matrix's pivot block.

4.7. Repeated Failures. Once a failed frontal matrix's pivot block as been absorbed into a recovery matrix, there are no guarantees that the previously lost pivots will be acceptable in their new frontal matrix. Hence, the question of whether the recovery is now complete or not needs to be asked.

The answer is simple and elegant and aids significantly in both the theoretical development and implementation of lost pivot recovery. The following theorem provides this answer:

THEOREM 4.23 (RECOVERY COMPLETION). *Once the lost pivot block of a failed frontal matrix (A) has been absorbed into the specified recovery frontal matrix (R_A), the recovery of the failed frontal matrix is complete. Any subsequent failures of the lost pivot block of A can be handled as failures in the recovery frontal matrix.*

Proof: For all $B \in \mathcal{F}_i$ such that $A \overset{L^+}{\rightsquigarrow} B$ and $B < R_A$, the rows of $\mathcal{L}'_B$ in the $\mathcal{U}'_{A_{lost}}$ columns have become part of the pivot rows of B by Corollary 4.8. Furthermore, any other nonzero rows of the $\mathcal{U}'_{A_{lost}}$ columns that are not contained in $\mathcal{L}'_B$ for one of the B's defined above are absorbed in $\mathcal{L}_{R_A}$ by Theorem 4.15.

In a similar fashion, all $C \in \mathcal{F}_i$ such that $A \overset{U^+}{\rightsquigarrow} C$ and $C < R_A$, the columns of $\mathcal{U}'_C$ in the $\mathcal{L}'_{A_{lost}}$ rows have become part of the pivot columns of C by Corollary 4.11. Furthermore, any other nonzero columns of the $\mathcal{L}'_{A_{lost}}$ rows that are not contained in $\mathcal{U}'_C$ for one of the C's defined above are absorbed in $\mathcal{U}_{R_A}$ by Theorem 4.15.

Thus, all the effects of A's lost pivot recovery have been absorbed into other frontal matrices and subsequent failures of the pivots in R_A can be handled as failures in R_A since no new edges from R_A have been added to $\mathcal{E}_T$ based on the results of Theorem 4.16. □

COROLLARY 4.24. *The recovery of a particular failed frontal matrix need not progress past the topological level of its recovery matrix.*

The significance of this result is that the scope of a failed frontal matrix's recovery is now limited to a well defined subset of the frontal matrices. This result can also be used to specify the ordering of recovery resolutions necessary when multiple failed frontal matrices affect a particular frontal matrix.

4.8. Ordering Recovery Resolutions. When multiple failed frontal matrices affect a particular frontal matrix, the ordering with which the effects of each recovery upon the current frontal matrix are determined is important. In Theorem 4.20, the ordering of failed L and U ancestors' effects is based on the positions of the ancestors' recovery matrices. Furthermore, Theorem 4.22 requires that all recovery operations for failed frontal matrices that are recovered by the current frontal matrix be resolved before any other failed frontal matrices' recoveries. With Theorem 4.23 now also established, a composite ordering of the recovery resolutions is defined by the following theorem.

THEOREM 4.25 (ORDERING RECOVERY RESOLUTIONS). *If resolution of the effects of multiple failed frontal matrices on a particular frontal matrix are ordered by ascending topological level of the failed frontal matrices' recovery matrices, the ordering criteria of Theorems 4.20 and 4.22 will be met.*

Proof: By Theorem 4.23 all recovery effects are terminated at the recovery matrix. Theorem 4.5 insures paths from all affected frontal matrices to an LU ancestor recovery matrix. Thus, any frontal matrix affected by a failed frontal matrix must be at a topological level lower than the recovery matrix of the failed frontal matrix.

Combining this result with the results of Corollary 4.21 and Theorem 4.22 completes the proof. □

With this result established, all the necessary mechanisms can be built to accommodate any number of failed frontal matrices. Yet, the communication paths upon which frontal matrices will share the information necessary for these recovery mechanisms have not been fully defined. The next section completes the theoretical development by establishing that either $\mathcal{E}_T$, a transitive reduction of $\mathcal{E}_T$, or $\mathcal{E}_A$ can be used for the control edge set $\mathcal{E}_C$ used for lost pivot recovery.

4.9. Communication Paths. During the recovery of a failed frontal matrix, the lost pivot block, the rows $\mathcal{L}''$ of the lost pivot columns, and the columns $\mathcal{U}''$ of the lost pivot rows must be passed to and absorbed by other frontal matrices. In addition, new fill-in can occur in the lost pivot rows and columns and this fill-in must also be passed to and absorbed by other frontal matrices. This section will show that the communication paths necessary for this data passing can be either $\mathcal{E}_T$, a transitive reduction of $\mathcal{E}_T$, or the assembly edge set $\mathcal{E}_A$. The resulting edge set is called the *control edge set* and is designated as $\mathcal{E}_C$.

THEOREM 4.26 ($\mathcal{E}_C$ BUILT DIRECTLY FROM $\mathcal{E}_T$). *$\mathcal{E}_T$ is sufficient to accommodate all necessary communication for lost pivot recovery.*

Proof: The definition of $\mathcal{E}_T$ insures that there are paths from each frontal matrix to its recovery frontal matrix. Theorems 4.13, 4.14, and 4.18 insure that all recovery effects are then limited to frontal matrices on paths in $\mathcal{E}_T$. □

This definition of $\mathcal{E}_C$ does, however, provide one serious drawback. Specifically, the $\mathcal{E}_T$ edge set is very large and thus will impose serious performance impacts. The next theorem establishes the sufficiency of a much smaller edge set to be used as $\mathcal{E}_C$.

THEOREM 4.27 ($\mathcal{E}_C$ BASED ON A TRANSITIVE REDUCTION OF $\mathcal{E}_T$). *If a transitive reduction of $\mathcal{E}_T$ is used as $\mathcal{E}_C$ then the resulting edge set will still be sufficient for all necessary lost pivot recovery.*

Proof: Follows directly from the definition of a transitive reduction. That is, if there is an edge $A \rightarrow B$ in $\mathcal{E}_T$, then there is also a path from A to B in the transitive reduction of $\mathcal{E}_T$. When combined with the results of Theorem 4.26, this completes the proof.□

Furthermore, if an assumption of block triangular form can be made, the original edge set from the assembly DAG, designated $\mathcal{E}_A$, can also be used as the control edge set $\mathcal{E}_C$. While use of $\mathcal{E}_A$ for this purpose precludes the need to compute the transitive reduction of $\mathcal{E}_T$, there are typically about 10% more edges in $\mathcal{E}_A$ than in the reduced $\mathcal{E}_T$ which results in additional run time overhead.

4.10. Absence of Block Triangular Form. If block triangular form can not be assumed, lost pivot recovery is still possible but more difficult. Specifically, empty contribution blocks are possible and LU ancestors are not guaranteed. Furthermore, $\mathcal{E}_T$ is insufficient for lost pivot recovery and must be augmented. Likewise, $\mathcal{E}_A$ is also insufficient for lost pivot recovery. While these results complicate lost pivot recovery, it is still possible. For details on this more generalized case, see [13].

5. Implementation. The implementation of lost pivot recovery is based on a parallel distributed memory version of the unsymmetric pattern multifrontal method called PRF (for Parallel_ReFactor). The PRF software accepts an assembly DAG and factorizes sequences of matrices that have a nonzero structure identical to the matrix for which the assembly DAG was created. Initially, the PRF code strictly uses the initial pivot ordering. With the inclusion of lost pivot recovery, the PRF

code is enhanced to include an avoidance strategy (relaxing of the pivot threshold), intra-frontal matrix recovery, and inter-frontal matrix recovery.

The PRF software uses host-based preprocessing to statically schedule, allocate, and assign frontal matrix tasks. The schedules, frontal matrix descriptions, and matrix entry values are then passed to the appropriate parallel processing nodes, which are part of an nCUBE 2 multiprocessor. The factorization of subsequent matrices in the sequence reuses the schedules and frontal matrix descriptions, and thus only new matrix entry values are required. The actual factorization of a matrix is accomplished by each parallel processing node executing the loop described in Figure 5.1 with multiple processors cooperatively handling the larger frontal matrix tasks. In this fixed pivot ordering version of the PRF code, synchronization between frontal matrices is accomplished using the schedules and the blocking message receive primitives for contribution messages and pivot column broadcasts.

```
for (each frontal matrix in this processor's schedule) do
        allocate storage for local portion of current frontal matrix
        assemble original entries and contributions into frontal matrix
        partially factorize the frontal matrix according to its pivots
        send contribution block to subsequent frontal matrices
        retain the LU factors for this frontal matrix
endfor
```

FIG. 5.1. *Original PRF Factorization Loop (Fixed Pivot Order)*

With the introduction of lost pivot recovery into the PRF code additional synchronization is required. Specifically, the effects of failed predecessor frontal matrices can cause frontal matrices to expand by new rows and/or columns. Such expansions must be assigned to processors and will require larger memory allocations for frontal matrix storage, updating of frontal matrix row and columns patterns, and possible remapping of the assembly of original entries and contributions. Furthermore, if multiple processors have been allocated to a particular frontal matrix, all of the cooperating processors must agree upon the resolution and allocation of the lost pivot recoveries. Hence two new levels of synchronization are required for lost pivot recovery. First, synchronization is required between frontal matrices as defined by the control edge set $\mathcal{E}_C$. This synchronization is done between the root processors of the frontal matrices' subcubes (the relative node 0's within the subcubes). Use of the transitively reduced $\mathcal{E}_T$ edge set as $\mathcal{E}_C$ reduces the number of required messages, but requires that specific frontal matrix recoveries be forwarded along paths in $\mathcal{E}_C$ to all affected frontal matrices. The $\mathcal{E}_C$ edge set is determined during host preprocessing and distributed to the parallel processing nodes. The recovery matrices (first LU ancestors) and their topological levels are also determined during host preprocessing for each frontal matrix. Theorem 4.24 specifies how this information can be used to discontinue the forwarding of individual frontal matrix recoveries. The second synchronization level is between the subcube processors for a distinct frontal matrix. That is, after the subcube's root processor has received and resolved all lost pivot recovery effects on the current frontal matrix, the resulting updates are broadcast to each of the other subcube processors. This insures agreement on how the lost pivot recovery effects have been resolved and allocated.

The two new levels of synchronization required for lost pivot recovery must be integrated into the frontal matrix task processing as defined by the PRF factorization loop of Figure 5.1. This integration and the other additional lost pivot recovery

processing is illustrated in Figure 5.2. Within this description, recovery structures are created for each frontal matrix with failed pivots. These structures include scalar data identifying the number of lost pivot rows and columns and the recovery matrix's ID and level. There is also vector data defining the patterns of lost pivot rows and columns, as well as the actual values of entries in the lost pivot rows and columns. For entries outside of the lost pivot block, the values must be accompanied by row, column, and source frontal matrix ID to facilitate duplicate elimination as multiple copies of the recovery structure may be created and passed along partially disjoint paths.

```
      for (each frontal matrix in this processor's schedule) do
            if (this is the root processor of the frontal matrix's subcube) then
1:                accept, combine, and resolve incoming recovery structures
2:                broadcast resulting updates to rest of frontal matrix's subcube
            else
3:                receive broadcast of updates due to lost pivot recovery
            endif
4:          allocate storage for local portion of current frontal matrix
5:          assemble original entries and contributions into frontal matrix
6:          partially factorize the frontal matrix according to it's pivots
            if (this is not the root processor of the frontal matrix's subcube) then
7:                forward local portions of lost pivot rows/columns and new
                  contributions due to earlier recoveries to root processor
            else
8:                receive the lost pivot recovery data from other subcube processors
9:                associate the lost pivot recovery data with appropriate recovery
                  structures updating corresponding patterns as necessary
10:               forward recovery structures to control parents
            endif
11:         forward contribution block to subsequent frontal matrices
12:         retain the LU factors for this frontal matrix
      endfor
```

FIG. 5.2. *PRF Factorization Loop Enhanced for Lost Pivot Recovery*

Details of the processing outlined in Figure 5.2 follow from the theoretical development of the previous section. Detailed descriptions of this processing can be found in [13]. A major complicating factor in this implementation is that the recovery structure for an individual failed frontal matrix is duplicated along disjoint control paths. Rows from this structure are absorbed by L ancestors and columns by U ancestors per Corollaries 4.8 and 4.11. Furthermore, new rows are introduced by L ancestors and new columns by U ancestors per Theorems 4.13 and 4.14. As both these absorptions and introductions can occur in distinct instances of the recovery structure, careful integration of the patterns and contributions of the various instances is necessary and accomplished by each frontal matrix task for all received recovery structures. A shared memory parallel implementation could maintain a single copy of each recovery structure with access controlled by locking mechanisms. This would eliminate much of the complexity associated with lost pivot recovery.

Intra-frontal matrix lost pivot recovery did not need to be addressed in the theoretical development but is handled within the partial dense factorization routine of step (6) in Figure 5.2. This partial dense factorization routine is based on a pipelined, fan-out algorithm developed by Geist and Heath [11] which uses a column-oriented

data allocation. In this algorithm, the cooperating processor that holds the next pivot's column will receive the multipliers (column of the L factor) and then update just the next pivot's column with these multipliers. The multipliers for this next pivot can then be computed and sent out after which the rest of columns can be updated by the current pivot's multipliers. In this way the next pivot multipliers can be on their way to the other processors before those processors need them.

To adapt this algorithm for intra-frontal matrix recovery, the processor owning the next pivot column will first check for a valid pivot in the pivot block portion of the column. If a valid pivot cannot be found in this column, the other pivot columns held by this processor are updated (using the current pivot's multipliers) and checked for valid pivots. If one of these other columns has a valid pivot, its multipliers are computed and sent out with the necessary permutation information appended. If the processor owning the next pivot has no columns with valid pivots, it sends out a failure status instead of the multiplier broadcast. When the other processors attempt to read these multipliers and see the failure status, each will check its own potential pivot columns for a valid pivot. If a valid pivot is found, the processor will contribute the forward distance from the original pivot owner's processor ID to its own processor ID (with these IDs being relative to the frontal matrix's assigned subcube). Otherwise, if no valid pivot was found, the number of total processors in the assigned subcube is contributed. These contributions are made to a global synchronization operation that uses prefix fan-in and fan-out computations to return the minimum of all contributions to each participating processor. As a result of this operation, each processor will know the nearest processor to the current pivot owner that has a valid pivot. This processor will become the new owner of this pivot and compute and send the corresponding multipliers. If the global synchronization operation returns the size of the assigned subcube, then there are no valid pivots held by any processor and the frontal matrix has lost pivots that must be handled by inter-frontal matrix recovery mechanisms.

6. Performance Results. With the theoretical development of a robust lost pivot recovery mechanism complete and its implementation summarized, we now present the performance of the parallel factorize-only algorithm, when pivot failures are present. 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 [20, 21]. 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 are done with the pivot threshold value set to 0.1 (which is the value used during the analyze-factorize UMFPACK run that built the assembly DAG and thus specified the initial pivot ordering) and to 0.001 (which corresponds to the use of a lost pivot avoidance strategy). To put these times into perspective, they will be compared to an estimation of the analyze-factorize

time. Only an estimated analyze-factorize time is possible, as the UMFPACK analyze-factorize routine required too much memory to be run on an single nCUBE processing node. The estimate of the analyze-factorize time was achieved by taking the ratio of the UMFPACK analyze-factorize time to the UMFPACK refactor time as run on a single processor of a KSR-1 system and multiplying the ratio by the single nCUBE 2 processing node execution time of the fixed pivot order refactorization.

The sequential execution time results for the three test matrix sequences are shown in Figures 6.1, 6.2, and 6.3. The RDIST1 matrix experienced few lost pivots even if an avoidance strategy was not in use. For all the matrices in this sequence, the factor-only code with lost pivot recovery ran faster than the estimated time to perform an analyze-factor operation. For the RDIST2 and RDIST3A sequences, the use of an avoidance strategy was necessary for the factor-only time to be better than the estimated analyze-factor time, yet this combination did consistently produce the fastest execution times. Without an avoidance strategy, the execution times soon exceeded the estimated analyze-factor times. The conclusion reached from these sequential execution tests was that a factor-only algorithm that incorporates lost pivot recovery is a viable alternative to the analyze-factor operation for sequences of identically structured matrices. This is especially true if an avoidance strategy is employed.

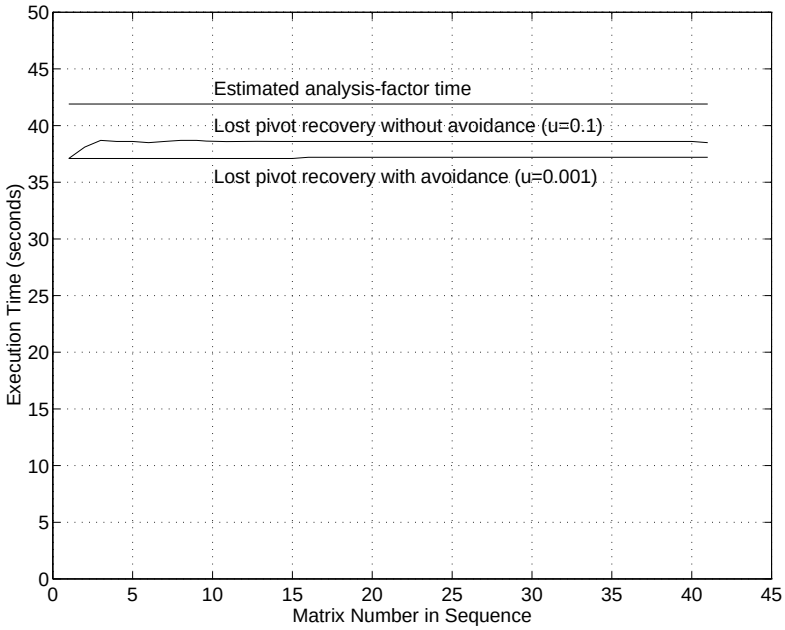

FIG. 6.1. *Sequential Execution Time for RDIST1 Sequence*

The sequential execution time results indicate that lost pivot recovery is viable even only a single processor is in use. The implementation, however, also allows for the exploitation of parallelism. Many parallel executions were done on selected matrices from each of the three sequences on hypercubes of dimensions 1 to 6 (2 to 64 processors). Matrix 37 of the RDIST3A sequence experienced the most lost pivots and, hence, the worst parallel performance. With avoidance this matrix experienced 52 lost pivots and without avoidance there were 221 lost pivots. Figure 6.4 presents

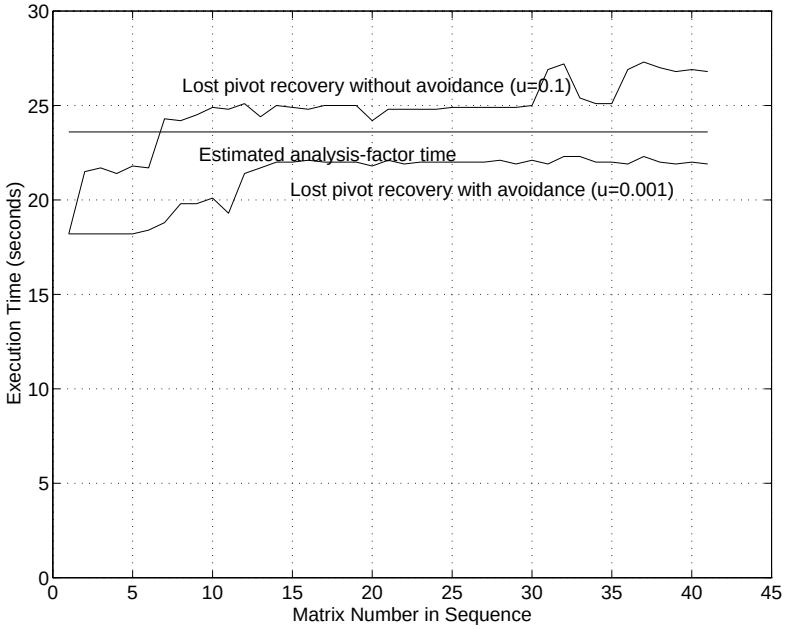

FIG. 6.2. *Sequential Execution Time for RDIST2 Sequence*

the parallel speedup results for this matrix. In this figure, the perfect (linear) speedup and speedup without using lost pivot recovery (LPR) curves are shown for reference. The curve for LPR with no lost pivots was obtained by running the factor-only PRF code with LPR enabled on the first matrix in the sequence in which all anticipated pivots were acceptable. Comparison of this curve with the curve that did not employ LPR shows that the overhead required for the LPR code has minimal adverse impact on speedup. The LPR with avoidance and LPR without avoidance curves illustrate performance achieved using matrix 37 of the RDIST3A sequence. These curves indicate that the benefits of parallelism are significantly reduced when more pivots are lost. However, gains due to parallelism are still very much achievable. Furthermore, these results show the worst of all test cases. Typically significantly fewer pivots were lost and the corresponding degradations in achievable parallelism were significantly less severe.

7. Conclusion. The unsymmetric-pattern multifrontal method of Davis and Duff has proven to have extremely competitive sequential performance [19] and significant parallel potential [16]. When the method is applied to sequences of identically structured matrices, the assembly DAG from the analyze-factor operation on the first matrix can be reused for factor-only operations on the subsequent matrices in the sequence. However, if entries anticipated as pivots by the assembly DAG become no longer numerically acceptable, recovery mechanisms are necessary to maintain the viability of a factor-only version of the method.

This paper describes such a lost pivot recovery mechanism for the unsymmetric pattern multifrontal method. A thorough theoretical development of the lost pivot recovery mechanisms established correctness and robustness. Furthermore, these mechanisms preserve the node and edge sets of the original assembly DAG. Preservation of the assembly DAG is especially important for distributed memory implementations

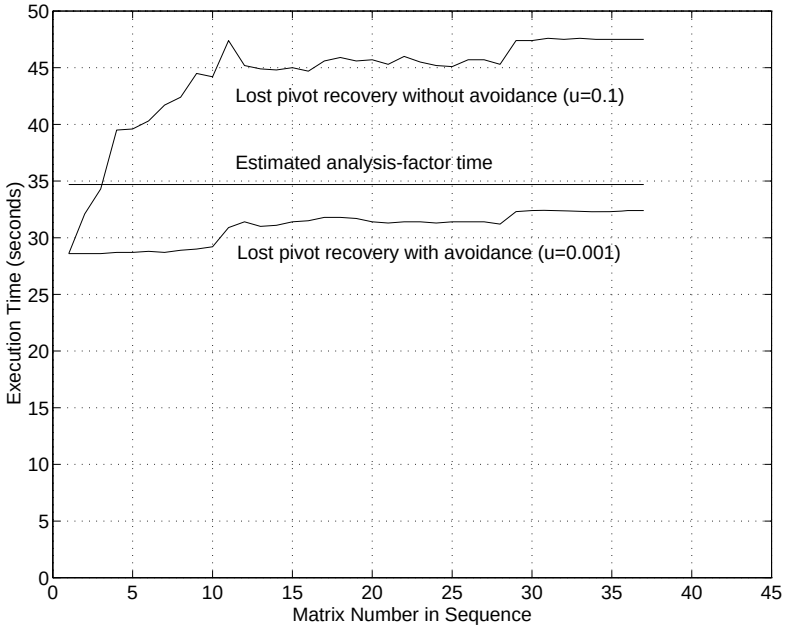

FIG. 6.3. *Sequential Execution Time for RDIST3A Sequence*

so that static scheduling and allocations can be utilized. The implementation of these mechanisms into a parallel distributed memory factor-only code on the nCUBE 2 was also described and its performance evaluated. Analysis of these performance results justifies the following conclusions:

- A fully robust LPR capability is possible for the unsymmetric pattern multifrontal method and is applicable to both sequential and parallel implementations.
- The LPR capability can effectively reduce overall execution time, especially if combined with an avoidance strategy.
- Performance benefits due to parallelism are definitely achievable when lost pivot recovery mechanisms are incorporated into the PRF code. However, the gains due to parallelism are limited when more anticipated pivots are lost during the factorization process.

Future efforts will likely focus on a parallel shared memory factor-only algorithm based on the unsymmetric-pattern multifrontal method. Many of the performance-limiting aspects of the parallel distributed memory lost pivot recovery implementation were directly related to the decentralization of critical data structures. A shared memory implementation should experience significantly better parallel performance.

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).

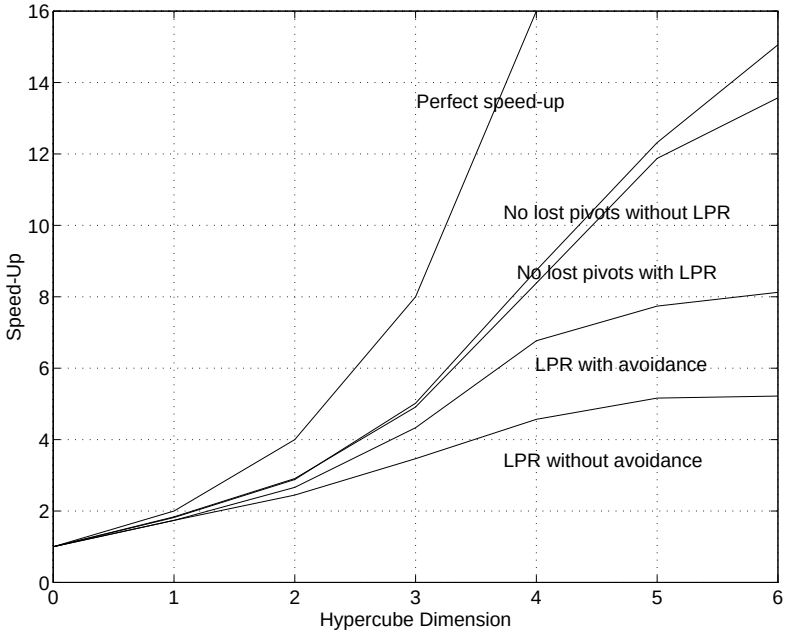

FIG. 6.4. *RDIST3A (matrix 37) Speed-Ups With and Without Lost Pivot Recovery (LPR)*

[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. C. EISENSTAT AND J. W. H. LIU, *Exploiting structural symmetry in unsymmetric sparse symbolic factorization*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 202–211.

[10] ———, *Exploiting structural symmetry in a sparse partial pivoting code*, SIAM J. Sci. Statist. Comput., 14 (1993), pp. 253–257.

[11] G. A. GEIST AND M. HEATH, *Matrix factorization on a hypercube*, in Hypercube Multiprocessors 1986, M. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986, pp. 161–180.

[12] J. R. GILBERT AND J. W. H. LIU, *Elimination structures for unsymmetric sparse LU factors*, SIAM Journal on Matrix Analysis and Applications, 14 (1993), pp. 334–354.

[13] 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. (also TR-94-019).

[14] 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.

[15] 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).

[16] S. M. HADFIELD AND T. A. DAVIS, *Potential and achievable parallelism in the unsymmetric-pattern multifrontal LU factorization method for sparse matrices*, in Fifth SIAM Conference on Applied Linear Algebra, 1994. (also TR-94-006).

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

Note: all University of Florida technical reports in this list of references are available in postscript form via anonymous ftp to `ftp.cis.ufl.edu` in the directory `cis/tech-reports`.