This paper appears in
Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing,
J. Dongarra, K. Kennedy, P. Messina, D. C. Sorensen and R. G. Voigt, eds.,
SIAM, Philadelphia, 1992, pp.34-40.

Distributed Multifrontal Factorization Using Clique Trees[1]

Alex Pothen and Chunguang Sun

Abstract

We describe the issues involved in the design and implementation of an efficient parallel multifrontal algorithm for computing the Cholesky factorization of a sparse matrix on distributed memory multiprocessors. The clique tree data structure is used to organize the numerical factorization. A crucial issue is the computation of good mappings of the computational tasks on processors to achieve load balance and low communication costs. Good speedups have been obtained for many practical problems on an iPSC/2. Both the arithmetic complexity and the communication complexity of our algorithm on grid graphs are asymptotically optimal.

[1] This research was supported by NSF grants CCR-8705110 and CCR-9024954, and by U. S. Air Force Office of Scientific Research grant AFOSR-88-0161.

DISTRIBUTED MULTIFRONTAL FACTORIZATION USING CLIQUE TREES †

ALEX POTHEN‡ AND CHUNGUANG SUN§

Abstract. We describe the issues involved in the design and implementation of an efficient parallel multifrontal algorithm for computing the Cholesky factorization of a sparse matrix on distributed memory multiprocessors. The clique tree data structure is used to organize the numerical factorization. A crucial issue is the computation of good mappings of the computational tasks on processors to achieve load balance and low communication costs. Good speedups have been obtained for many practical problems on an iPSC/2. Both the arithmetic complexity and the communication complexity of our algorithm on grid graphs are asymptotically optimal.

1. Introduction. The multifrontal method has proved to be attractive in solving large sparse systems of equations efficiently in many different computational environments (see Liu [6] for a survey). We describe a parallel multifrontal sparse Cholesky factorization algorithm for distributed-memory multiprocessors. We show that the clique tree is a natural data structure in formulating the multifrontal method in that the parallel algorithm is easily described and efficiently implemented in terms of it.

Two groups have described previous implementations of the distributed multifrontal method. The algorithm of Lucas, Blank, and Tiemann [7] requires the use of the nested dissection ordering, and also requires extra storage since a column in a separator is stored on more than one processor. Ashcraft, Eisenstat, Liu, and Sherman [1, 2] have compared the distributed multifrontal method with two variants of column-oriented distributed sparse factorization algorithms, and have shown that the multifrontal approach outperforms the other algorithms on an iPSC/2 for the model grid problem. However, the crucial issue of task partitioning on processors to achieve load balance and to reduce communication costs

† This research was supported by NSF grants CCR-8705110 and CCR-9024954, and by U.S. Air Force Office of Scientific Research grant AFOSR-88-0161.

‡ Department of Computer Science, Whitmore Lab, The Pennsylvania State University, University Park, PA 16802. Electronic address: pothen@cs.psu.edu, na.pothen@na-net.ornl.gov.

§ Department of Computer Science, Whitmore Lab, The Pennsylvania State University, University Park, PA 16802. Electronic address: csun@cs.psu.edu.

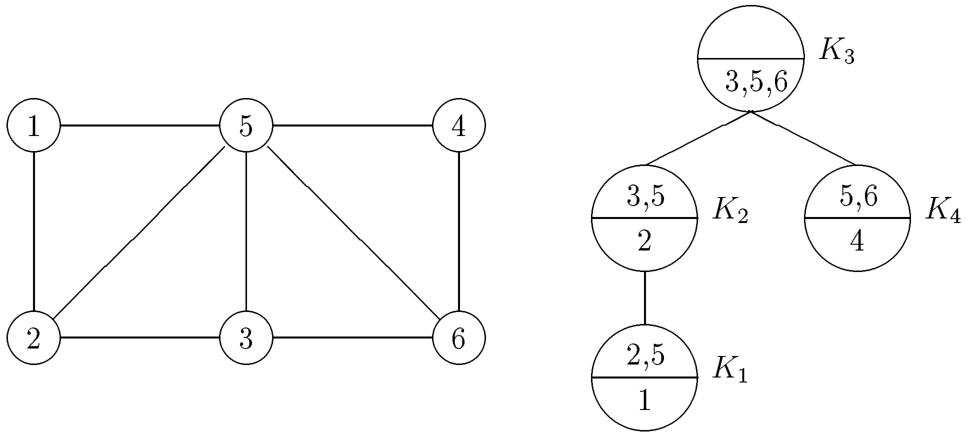

FIG. 1. *A chordal adjacency graph of a* 6×6 *Cholesky factor and its corresponding clique tree.*

for general sparse problems has not been addressed in previous work.

In this paper, we develop a distributed multifrontal algorithm by using the clique tree. A new task partitioning and mapping strategy for general sparse problems is proposed and compared with existing schemes. Previous works on distributed sparse Cholesky factorizations report experimental results on only two classes of problems—grids and L-shaped problems [1, 2, 3]. We have experimented with a collection of problems with no regular sparsity structure from the Boeing-Harwell test set and have obtained good speedups.

2. Application of clique trees to the multifrontal method. We assume that the reader is familiar with the graph model of sparse Cholesky factorization. The adjacency graph G of a sparse Cholesky factor L is a *chordal* graph, i.e, a graph in which every cycle with more than three edges has a *a chord*, an edge joining two vertices which are nonadjacent on the cycle. A *clique* is a subgraph in which every pair of vertices is adjacent. A *maximal clique* is a clique such that no additional vertex in the graph can be added to it while preserving the clique property. The set of maximal cliques of the chordal graph G can be organized into a tree structure called the *clique tree*. A discussion of clique trees and their applications in sparse matrix algorithms may be found in [5, 9]. The adjacency graph of a 6×6 Cholesky factor and a corresponding clique tree are shown in Fig. 1. Note that in the figure, each maximal clique is drawn as a circle, and the vertices of the clique are listed in the circle.

Each maximal clique K (henceforth clique) is partitioned into two sets—$new(K)$, the *new* set of vertices, and $anc(K)$, the *anc* set of vertices. The *new* set is shown below the line in a circle and the *anc* set above the line in a circle in Fig. 1. Every vertex in $anc(K)$ belongs also to some ancestor clique of K, and vertices in $new(K)$ do not belong to any ancestor clique. An important property of clique trees computed from the algorithm in [5] is that $anc(K) \subset P$, where P is the parent of K, and P is the only clique satisfying this relation.

Let $s = |new(K)|$, $t = |anc(K)|$, and $r = s + t$. The clique K corresponds to an $r \times r$ dense submatrix in L called the *frontal matrix* of K and is denoted by M_K. The columns in

$new(K)$ form the first s columns of M_K and the columns in $anc(K)$ form the last t columns of M_K. The computation associated with K is a partial Cholesky factorization(henceforth factorization) of M_K which eliminates its first s columns. Nonzeros in columns in $new(K)$ are initialized with values from the corresponding columns of A, and nonzeros in columns in $anc(K)$ are initialized to zero. After M_K is factored, the first s columns become a group of eliminated columns in L and the last t columns constitute a $t \times t$ dense submatrix representing the accumulated elimination effects on columns in $anc(K)$. This $t \times t$ dense submatrix is called the *update* matrix of clique K and is denoted by U_K. Recall that $anc(K) \subset P$, where P is the parent of K, and therefore, U_K updates M_P only.

A clique and the dense submatrix it represents will be used interchangeably in the following, and by the factorization of a clique K we mean the factorization of M_K. If $new(K) = \{i_1, \cdots, i_s\}$ and $anc(K) = \{j_1, \cdots, j_t\}$ for a clique K, it will be written as $\{i_1, \cdots, i_s; j_1, \cdots, j_t\}$. Now consider the example in Fig. 1. Let $K_1 = \{1; 2, 5\}$, $K_2 = \{2; 3, 5\}$, $K_3 = \{3, 5, 6\}$ and $K_4 = \{4; 5, 6\}$. Let M_i represent the frontal matrix and U_i the update matrix of a clique K_i. The factorization of M_1 produces the matrix

$$\begin{pmatrix} l_{11} & & \\ l_{21} & u_{22} & \\ l_{51} & u_{52} & u_{55} \end{pmatrix}.$$

The first column in this matrix $(l_{11}, l_{21}, l_{51})^T$ is the first column of L, and the last two columns form U_1. Since $anc(K_1) \subset K_2$, U_1 updates M_2 only. The update on matrix M_2 by U_1 is illustrated by means of an 'extend-add' operation as

$$\begin{pmatrix} u_{22} & \\ u_{52} & u_{55} \end{pmatrix} \bigoplus \begin{pmatrix} b_{22} & & \\ b_{32} & b_{33} & \\ b_{52} & b_{53} & b_{55} \end{pmatrix} \Longrightarrow \begin{pmatrix} b_{22} + u_{22} & & \\ b_{32} & b_{33} & \\ b_{52} + u_{52} & b_{53} & b_{55} + u_{55} \end{pmatrix}.$$

The process of updating a matrix M by another matrix U as described here is also referred to as assembly of the matrix U into the matrix M. In this example, the factorization of K_2 can not be started until the factorization of K_1 is finished and U_1 is assembled into M_2. The factorization of K_4 is, however, independent of the factorization of K_1 or K_2. The structure of a clique tree shows clearly the inherent parallelism in the problem.

In the multifrontal method, the Cholesky factorization of a sparse matrix is computed as a sequence of partial factorizations on a set of dense submatrices. These dense submatrices can be chosen to be the maximal cliques in a clique tree representation of the Cholesky factor. The numerical factorization starts with the leaves of the clique tree. Initially, all leaf cliques can be factored independently, and then the update matrix generated from each leaf clique is assembled into its parent. Then leaf cliques are deleted from the clique tree, and the cliques that newly become leaves are factored. This process is repeated until the root clique is factored. If the factorization of a clique is considered as a computational task, the clique tree captures the precedence relationships among all computational tasks. Since the factorization of the given sparse matrix is formulated in terms of partial factorizations of dense submatrices, vectorization on the processors can be fully exploited.

3. Mapping schemes. George, Liu, and Ng [4] have proposed the *subtree-to-subcube mapping* for the model grid problem ordered by nested dissection. They proposed their scheme with respect to the elimination tree, but we apply this scheme here to the clique tree of the grid. The root clique is partitioned among all processors. Each subtree of the root clique is allocated equal numbers of processors. This process is recursively applied to each subtree. Since the distribution of the arithmetic work in a clique tree is ignored, this scheme is only effective for those problems with balanced tree structure and balanced workload distribution such as the grids.

Geist and Ng [3] have proposed a bin-pack mapping scheme for mapping arbitrary trees. This is the only mapping strategy reported in the literature that is applicable to general problems. We have adapted their scheme, which was described with respect to the fan-out algorithm, to the multifrontal context. An important difference is that they wrap-map the columns which are outside the subtrees packed in the bins, but for the purpose of keeping communication costs low, we map a clique K outside the subtrees packed in the bins to the processors involved in computing subtrees of clique K. We omit other details of the bin-pack mapping scheme.

Now we propose a strategy for mapping arbitrary trees, called the *proportional mapping scheme*. The root of the clique tree is partitioned among all processors. Each subtree of a clique that has already been mapped to a set of processors is allocated a subset of processors whose size is proportional to the workload associated with that subtree. More precisely, the number of processors assigned to a subtree is obtained by rounding the number calculated above to the nearest integer. This process is then recursively applied to each subtree. Several safeguards need to be incorporated to ensure the correctness of the algorithm because of rounding. A complete description of the proportional mapping will be given in [10]. This scheme may be viewed as a generalization of the subtree-to-subcube mapping in which both the structure of the tree and the distribution of work in the tree are considered in mapping tasks. Proportional mapping has also been considered in the context of distributed sparse orthogonal factorization [8].

4. Distributed numerical factorization. All three mapping schemes discussed above partition the numerical factorization into a *local* phase and a *distributed* phase. In the local phase, processors work in parallel on independent subtrees and no communication among processors is necessary. In the distributed phase, a clique is factored by a group of processors. The cliques are ordered such that each processor performs its local phase computations first, and then in the distributed phase, processes the remaining cliques mapped to it in topological order on the clique tree. The sizes of the cliques on a clique tree are not uniform. There are fewer and bigger cliques near the 'top' of the tree, and there are more and smaller cliques near the 'bottom' of the tree. A bigger clique near the top is partitioned among a group of processors to achieve load balance. This is accomplished by mapping the columns of the clique onto the group of processors in a wrap-around fashion. Let me be a processor under consideration. The set of columns in $new(K)$ mapped to me is represented by $my\_new(K)$, and the set of columns in $anc(K)$ mapped to me is represented by $my\_anc(K)$. Let $my\_col(K) = my\_new(K) \cup my\_anc(K)$ and let $P(K)$ denote the group of processors on which a clique K is partitioned.

```
for each of my cliques K do
    if K is entirely mapped to me then
        allocate space for columns in anc(K) and initialize them to zero;
        for each child clique C of K do
            assemble the update matrix of C into M_K;
            discard the update matrix of C;
        end for
        perform the partial Cholesky factorization on M_K;
    else
        { clique K is partitioned among a set of processors }
        if my_col(K) ≠ ∅ then allocate space for columns in new(K) ∪ my_anc(K);
        { assemble the children of K into K if it is not a leaf clique }
        if K is not a leaf clique then multi_assem(K);
        { partial Cholesky factorization of M_K }
        if my_col(K) ≠ ∅ then multi_fact(K);
    end if
end for
```

FIG. 2. *The distributed multifrontal algorithm using clique trees.*

Initially the numerical values of a matrix A have to be distributed among processors according to the mapping of the clique tree. If a clique K is entirely mapped to a processor p, then the numerical values of A for the columns in $new(K)$ are distributed to p. If a clique K is partitioned, then the numerical values of A for the columns in $new(K)$ are distributed among processors in $P(K)$ according to the wrapping method used.

In the local phase, each processor executes a sequential version of the multifrontal algorithm using clique trees. The use of the multifrontal method is not really necessary in this phase, since any efficient sparse factorization algorithm can be used. However, the dense submatrix operations of the multifrontal method offer potential for vectorized processing. In the distributed phase, the assembly and factorization of a clique K need the cooperation of the processors in $P(K)$. A clique K is partially mapped to a processor p if K is partitioned and at least one column of K is mapped to p. A clique K is associated with a processor p if K is entirely mapped to p, or K is partially mapped to p, or the clique K is partitioned and at least one child of the clique K is mapped to p either entirely or partially. The reason for including the last type of cliques is that processor p needs to participate in the assembly of clique K, but not in the factorization of clique K. The overall algorithm for the numerical factorization is described in Fig. 2.

Let S_K denote the set of all columns in the update matrices of all child cliques of a clique K. Assembling a partitioned clique K takes two stages. In the first stage, each processor p in $P(K)$ processes the columns in S_K that are stored locally. In the second stage, each processor p in $P(K)$ processes the columns in S_K that are stored in other processors. This algorithm is shown in Fig. 3. Factorization of a partitioned clique K also takes two stages.

```
multi_assem(K)
for each column j in S_K stored locally do
    if column j updates a column of K stored locally then
        carry out the update and discard column j;
    else
        send column j to the processor storing the corresponding column of K;
        discard column j;
    end if
end for
for each column j in S_K not locally stored do
    if column j updates a column of K stored locally then
        receive column j, carry out the update and discard column j;
end for
```

FIG. 3. *The algorithm for distributed assembly of a partitioned clique.*

In the first stage, each processor p in $P(K)$ processes the *new* columns of K in increasing order. In the second stage, each processor p updates all of its columns in $anc(K)$ by the columns in $new(K)$. After K is factored, the update matrix of the clique K is scattered among the set of processors in $P(K)$ according to the wrapping scheme used. This algorithm is shown in Fig. 4.

5. Experimental results on an iPSC/2. We experimented with grid problems and a set of eight problems from the Boeing-Harwell collection on an iPSC/2. The eight problems are BCSPWR10, BCSSTK13, BCSSTK21, BLCKHOLE, NASA1824, NASA2146, NASA4704, and LSHP3466. All programs are written in C, and floating point numbers were represented in single precision. Comprehensive experimental results will be described in [10].

The optimal nested dissection ordering is used as the fill-reducing ordering for the grid problems. The clique tree of a square grid with nested dissection ordering is a balanced binary tree which is suited for the hypercube architecture since a subtree can be naturally mapped to a subcube. Experimental results on the 127×127 grid using all three mapping schemes are shown in Table 1. There is not enough space for computing the Cholesky factorization associated with the 127×127 grid on one processor. Hence speedups are computed with respect to the running time on two processors; this is justified since we have observed efficiencies close to one for the model problem on two processors.

A user-supplied tolerance is needed by the bin-pack mapping algorithm. If the ratio of the weights of the lightest bin to the heaviest bin is greater than or equal to the tolerance, then the bin packing process terminates. 'Time_P' denotes the running time for proportional mapping in seconds. 'Prop', 'Subcube', 'Bin40', 'Bin60' and 'Bin80' denote the speedups for the proportional mapping, subtree-to-subcube mapping and bin-pack mapping with tolerance values 40%, 60% and 80%, respectively. Higher values of the tolerance cause the bin-pack algorithm to pack several small and unrelated subtrees into a bin to satisfy load balance. In turn, this causes loss of parallelism and increased communication costs, greatly

multi_fact(K)
for each column j in $new(K)$ in increasing order **do**
    if column j is my column **then**
        scale column j and send it to all other processors in $P(K)$;
    else
        receive column j;
    end if
    update all unprocessed columns in $my\_new(K)$ by column j;
end for
update columns in $my\_anc(K)$ by columns in $new(K)$;
discard the columns in $new(K) \setminus my\_new(K)$;

FIG. 4. *The algorithm for distributed factorization of a partitioned clique.*

slowing down the factorization. Since clique trees of square grids are well balanced, the proportional mapping and the subtree-to-subcube mapping produce similar mappings and they have essentially the same performance for the grids.

TABLE 1
Speedups obtained from proportional mapping, subtree-to-subcube mapping and bin-pack mapping for the 127×127 *grid.*

No. Proc	Time_P	Prop	Subcube	Bin40	Bin60	Bin80
2	148.18					
4	75.404	3.93	3.93	3.93	3.93	3.93
8	44.246	6.70	6.74	6.68	6.68	6.70
16	25.374	11.7	11.7	11.7	9.51	6.32
32	13.843	21.4	20.4	20.3	16.0	8.55

No performance results are available in the literature on the distributed Cholesky factorization of general sparse matrices, and we report results on the Boeing-Harwell problems for the first time. The multiple minimum degree ordering is used as the fill-reducing ordering. Our algorithm performs well on BCSSTK13, BCSSTK21, BLCKHOLE, NASA1824, NASA2146, NASA4704, and LSHP3466, but poorly on BCSPWR10. Two extreme examples representing good and low performance are shown in Fig. 5. The y-axis represents the efficiency which is the ratio of speedup to the number of processors. Since the subtree-to-subcube mapping is clearly not suitable for mapping general problems, we compare the performance of the proportional mapping with that of the bin-pack mapping. The curve for proportional mapping is indicated by 'prop'. We have experimented with three tolerances 40%, 60% and 80% (indicated by 'b40', 'b60' and 'b80', respectively) for the bin-pack mapping.

The primary factor which determines the efficiency of the distributed multifrontal algorithm is the number of floating point operations that need to be performed. If this number is large, arithmetic operations dominate communication costs and other data structure over-

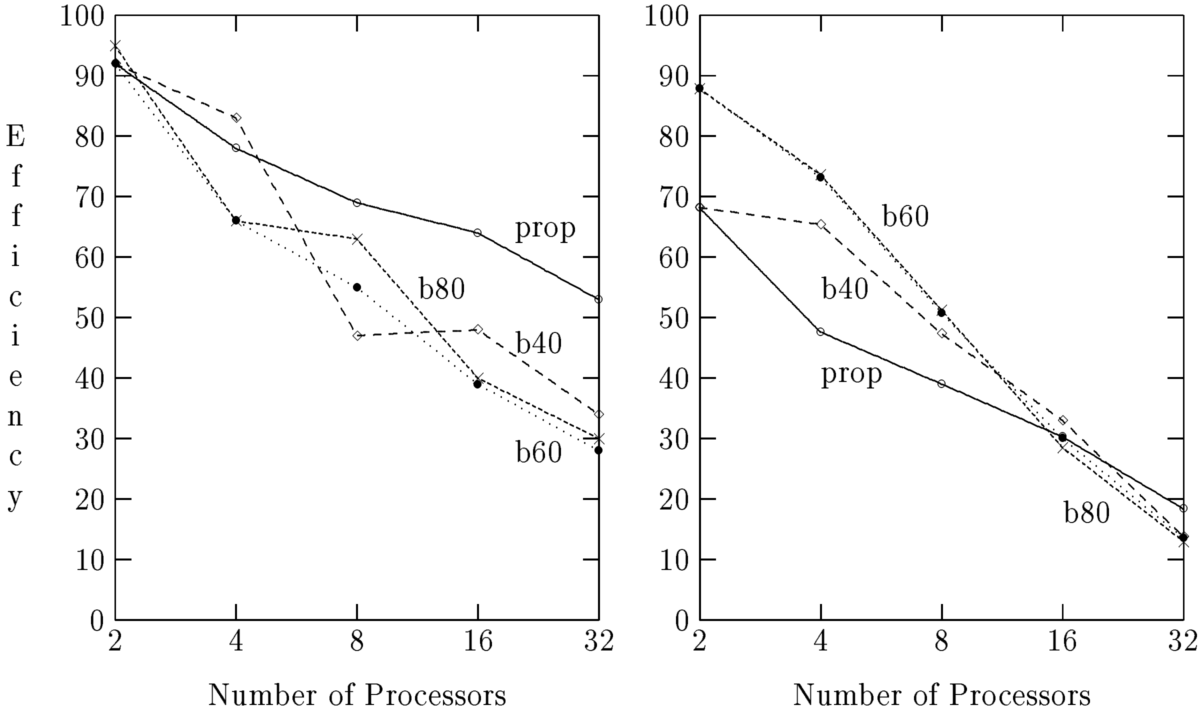

FIG. 5. *Comparison of mapping schemes on* BCSSTK13*(left) and* BCSPWR10*(right)*

heads, and efficiencies are high. The serial multifrontal algorithm performs about 30 million flops on BCSSTK13 and takes 376 seconds on a single processor, while BCSPWR10 requires about 160 thousand flops and takes only 4.16 seconds on a single processor; thus efficiencies are good for the former and poor for the latter problem, when the number of processors is large.

Another feature that influences efficiencies is the structure of the clique tree. The more balanced the clique tree, the better the performance. The clique tree of BCSSTK13 is reasonably well-balanced, while the clique tree of BCSPWR10 is highly unbalanced.

The performance of the bin-pack mapping is sensitive to the tolerance used, and one tolerance is not uniformly better than another over varying number of processors for a given problem. This is a serious drawback since it is difficult to determine an appropriate tolerance *a priori*. In general, the bin-pack mapping performs better than the proportional mapping for very few (2 or 4) processors, and the proportional mapping is better for larger numbers of processors. The performance of the bin-pack mapping deteriorates much faster than that of proportional mapping as the number of processors increases.

6. Concluding remarks. We have theoretically analyzed the performance of our parallel clique tree-based multifrontal algorithm with the subtree-to-subcube mapping on nine-point $k \times k$ grids whose vertices are numbered by nested dissection. We have shown that the arithmetic complexity is $\mathcal{O}(k^3/p)$ and the communication complexity is $\mathcal{O}(k^2)$, when p processors are used. Furthermore, we have proved that both the arithmetic complexity and the communication complexity are asymptotically optimal.

We have shown that the clique tree data structure can be used to obtain an efficient distributed multifrontal Cholesky factorization algorithm. The crucial issue of task mapping has been studied and experimental results for general sparse problems have been obtained. Several enhancements necessary to improve performance remain to be explored; among these are the design of improved mapping algorithms, the computation of good orderings, and the restructuring of clique trees. Our algorithm offers great potential for distributed-memory multiprocessors with vector processing capabilities. We are currently investigating these issues.

REFERENCES

[1] C. C. Ashcraft, *The domain/segment partition for the factorization of sparse symmetric positive definite matrices*, Tech. Report ECA-TR-148, Boeing computer Services, Seattle, Washington, Nov 1990.

[2] C. C. Ashcraft, S. C. Eisenstat, J. W. Liu, and A. H. Sherman, *A comparison of three column-based distributed sparse factorization schemes*, Tech. Report CS-90-09, Department of Computer Science, York University, North York, Ontario, Canada, Aug 1990.

[3] G. A. Geist and E. G. Ng, *Task scheduling for parallel sparse Cholesky factorization*, International Journal of Parallel Programming, 18 (1989), pp. 291–314.

[4] J. A. George, J. W. Liu, and E. G. Ng, *Communication results for parallel sparse Cholesky factorization on a hypercube*, Parallel Computing, 10 (1989), pp. 287–298.

[5] J. G. Lewis, B. W. Peyton, and A. Pothen, *A fast algorithm for reordering sparse matrices for parallel factorization*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 1146–1173.

[6] J. W. Liu, *The multifrontal method for sparse matrix solution: theory and practice*, Tech. Report 90-04, Computer Science, York University, May 1990.

[7] R. Lucas, T. Blank, and J. Tiemann, *A parallel solution method for large sparse systems of equations*, IEEE Transactions on Computer-Aided Design, CAD-6 (1987), pp. 981–991.

[8] A. Pothen and P. Raghavan, *Sparse orthogonal factorization on a distributed-memory multiprocessor.* In preparation, 1991.

[9] A. Pothen and C. Sun, *Compact clique tree data structures in sparse matrix factorizations*, in Large-Scale Numerical Optimization, Proceedings in Applied Mathematics, vol. 46, Society for Industrial and Applied Mathematics, Dec 1990, pp. 180–204.

[10] ——, *An efficient distributed multifrontal algorithm.* In preparation, 1991.