SIAM J. SCI. STAT. COMPUT.
Vol. 13, No. 2, pp. 645–653, March 1992

013

TIMELY COMMUNICATION

Under the "timely communications" policy for the SIAM Journal on Scientific and Statistical Computing, papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of the journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.

A FAST REORDERING ALGORITHM FOR PARALLEL SPARSE TRIANGULAR SOLUTION*

ALEX POTHEN† AND FERNANDO L. ALVARADO‡

Abstract. A space-efficient partitioned representation of the inverse of a unit lower triangular matrix L may be used for efficiently solving sparse triangular systems on massively parallel computers. The number of steps required in the parallel triangular solution is equal to the number of subsets of elementary triangular matrices in the partitioned representation of the inverse. Alvarado and Schreiber have recently described two partitioning algorithms that compute the minimum number of subsets in the partition over all permutations of L which preserve the lower triangular structure of the matrix. Their algorithms require space linear and time nonlinear in the number of nonzeros in L. This paper describes a partitioning algorithm that requires only $\mathcal{O}(n)$ time and space for computing an optimal partition, when L is restricted to be a Cholesky factor. (Here n is the order of L.) The savings result from the observation that instead of working with the structure of L, it is sufficient to work with its transitive reduction, the elimination tree of L. Experimentally the new partitioning algorithm requires negligible time in comparison to the previous partitioning algorithms and to the Multiple-Minimum-Degree ordering algorithm.

Key words. directed acyclic graph, elimination tree, massively parallel computers, reordering algorithm, sparse Cholesky factorization, sparse triangular systems, transitive reduction

AMS(MOS) subject classifications. 65F50, 65F25, 68R10

1. The problem. A unit lower triangular matrix L of order n can be expressed as a product of elementary lower triangular matrices $L = \prod_{i=1}^{n-1} L_i$, where $L_i = I + \underline{m}_i \underline{e}_i^T$, $\underline{m}_i$ has its first i components equal to zero, and $\underline{e}_i$ is the ith coordinate vector. Assume that L is sparse. Consider a representation of L in which the elementary lower triangular matrices are grouped together to form m unit lower triangular factors $L = \prod_{i=1}^{m} P_i$, where each factor P_i has the property that P_i^{-1} can be represented in the same space as P_i. We say that P_i is invertible in place. Here each $P_i = \prod_{k=e_i}^{e_{i+1}-1} L_k$, with $e_1 \equiv 1 < e_2 < \cdots < e_m < e_{m+1} \equiv n$. This leads to a partitioned representation of the inverse of L of the form $L^{-1} = \prod_{i=m}^{1} P_i^{-1}$, that can be stored in just the space required for L. This partitioned inverse representation is advantageous when a triangular system $L\underline{y} = \underline{b}$ must be solved repeatedly with different right-hand-side vectors $\underline{b}$ on a massively parallel computer such as the Connection Machine; on such a machine, the solution $\underline{y}$ can be computed as $\underline{y} := \prod_{i=m}^{1} P_i^{-1} \underline{b}$ in m steps. This representation has been considered by Alvarado et al. in the Power Engineering literature [4], [8], and by Alvarado and Schreiber [3]. It has been observed that a

* Received by the editors April 8, 1991; accepted for publication (in revised form) August 6, 1991.
† Department of Computer Science, Whitmore Laboratory, Pennsylvania State University, University Park, Pennsylvania 16802 (pothen@cs.psu.edu, na.pothen@na-net.ornl.gov). A part of this work was done while the author was visiting the Departments of Computer Science and Mathematics at the University of Wisconsin, Madison. The research of this author was supported by National Science Foundation grant CCR-9024954, by U. S. Department of Energy grant DE-FG02-91ER25095, and by U.S. Air Force Office of Scientific Research grant AFOSR-88-0161.
‡ Electrical and Computer Engineering Department, 1425 Johnson Dr., University of Wisconsin, Madison, Wisconsin 53706 (alvarado@ece.wisc.edu). The research of this author was supported by National Science Foundation grants ECS-8822654 and ECS-8907391.

reduction in the number of factors m is advantageous, and that this number can be reduced by permuting L into a new lower triangular matrix L_Π, and determining the partitioned representation of L_Π^{-1}.

Henceforth, without loss of generality, we will assume that L is irreducible. Alvarado and Schreiber [3] considered the following problem:

(P1) Given a unit lower triangular matrix $L = \prod_{i=1}^{n-1} L_i$, find a permutation $L_\Pi = \Pi L \Pi^T$ and a representation $L_\Pi = \prod_{i=1}^{m} P_i$, where

(1) each $P_i = \prod_{k=e_i}^{e_{i+1}-1} L_k$, with $e_1 = 1 < e_2 < \cdots e_m < e_{m+1} = n$,

(2) each P_i is invertible in place, and

(3) m is minimum over all permutations Π such that L_Π is lower triangular.

They designed two algorithms to solve (P1), which they called RP1 and RP2 in their paper. Both algorithms require time nonlinear in the number of nonzeros in L, and space proportional to the number of nonzeros in L.

In this paper, we consider the restriction of (P1) to unit lower triangular matrices, which arise in the LDL^T factorization of symmetric, positive definite matrices. (Henceforth we call this the Cholesky factorization.) Several applications of this problem in Power Engineering are described in [8]. We describe an $\mathcal{O}(n)$-time algorithm to compute the minimum number of factors in the partitioned inverse representation of L. The algorithm requires only the elimination tree and the number of nonzeros in each column of L as input, and *not* the nonzero structure of L. Thus the space requirement of the proposed algorithm is $\mathcal{O}(n)$. Further, since the elimination tree and the nonzero counts of the columns of L may be computed directly from the original matrix A, the space requirement of the overall algorithm to compute the factors in the partitioned inverse representation from A is $\mathcal{O}(n + \tau(A))$, where $\tau(A)$ is the number of nonzeros in the strict lower triangle of A.

In (P1), the action of the permutation Π on L is to reorder the elementary matrices whose product is L; however, these elementary matrices cannot be arbitrarily reordered, since we require the resulting matrix L_Π to be lower triangular. From the equation $L_i = I + \underline{m}_i \underline{e}_i^T$, it can be verified that the elementary matrices L_i and L_{i+1} can be permuted if and only if $l_{i+1,i} = 0$. These precedence constraints on the order in which the elementary matrices may appear is captured in a graph model of the problem.

2. A graph model. Let $G(L) = (V, E)$ denote the directed graph with vertex set V equal to the set of columns of L, and an edge $(j, i) \in E$ (with $i > j$) if and only if $l_{i,j} \neq 0$. The edge (j, i) is directed from the lower-numbered vertex j to the higher-numbered vertex i. Hence $G(L)$ is a directed acyclic graph (DAG). Since we have assumed that L is irreducible, $G(L)$ is weakly connected, i.e., there is a path joining every pair of vertices in $G(L)$ when $G(L)$ is viewed as an undirected graph. If there is a directed path from a vertex j to a vertex i in $G(L)$, we will say that j is a *predecessor* of i, and that i is a *successor* of j. In particular, if $(j, i) \in E$, then j is a predecessor of i and i is a successor of j.

Given a subset P of the columns of L, the concept of the subgraph corresponding to the nonzeros in P will be useful in the remainder of this paper. Accordingly, we define the *subgraph of $G(L)$ induced by a set of vertices* P as the graph that contains all edges directed from vertices in P to all vertices in $G(L)$, and all the vertices that are the endpoints of such edges.

A *topological ordering* of $G(L)$ is an ordering of its vertices in which predecessors are numbered lower than successors; i.e., for every edge $(j, i) \in E$, $i > j$. By construction, the original vertex numbering of $G(L)$ is a topological ordering. A permutation

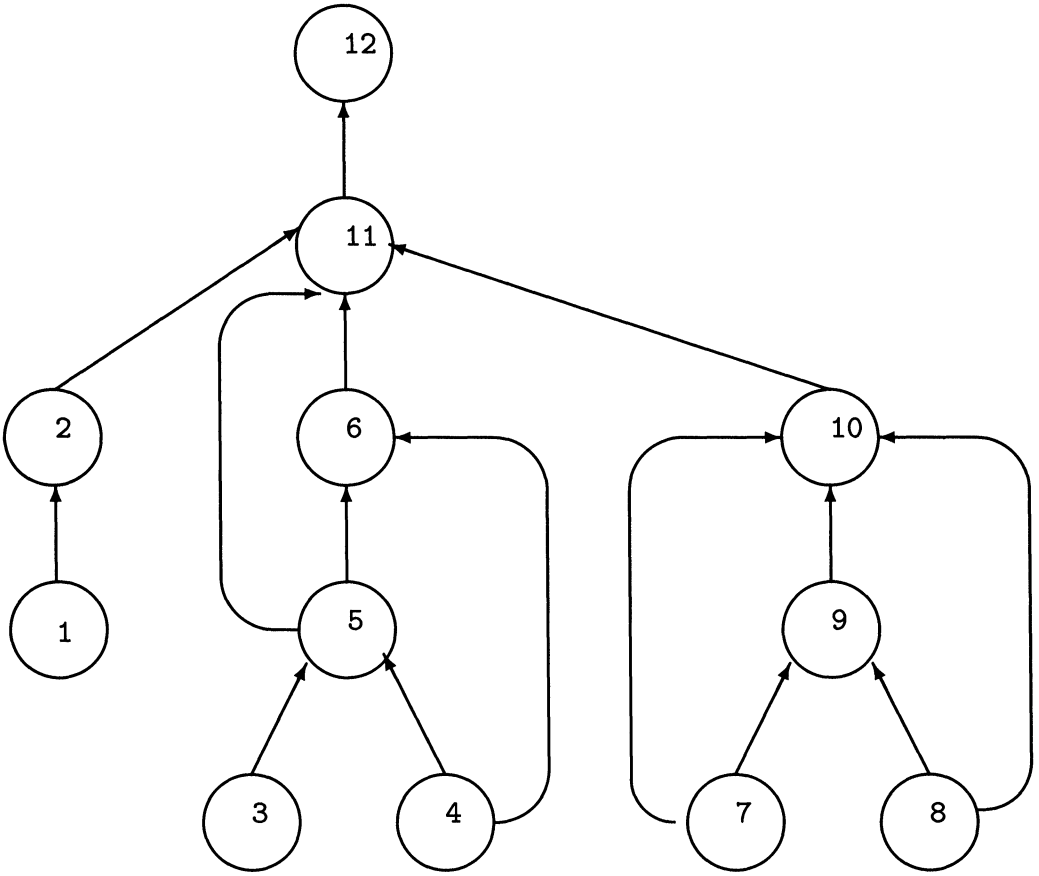

FIG. 1. *A directed acyclic graph $G(L)$ corresponding to a Cholesky factor L.*

Π that leaves L_Π lower triangular corresponds to a topological reordering of the vertices of $G(L)$. A topologically ordered DAG corresponding to a Cholesky factor L is shown in Fig. 1.

In what follows, we identify a subset of columns P with the factor formed by multiplying, in order of increasing column number, the elementary matrices corresponding to columns in P. The condition that the nonzero structure of a factor P should be the same as the structure of its inverse corresponds in the graph model to the requirement that the subgraph induced by P should be transitively closed [3], [9]. (A DAG G is *transitively closed* if for every pair of vertices j and i such that there is a directed path in G from j to i, the edge (j, i) is present in G.)

A graph model of (P1) is provided in the following problem:

(P2) Find an ordered partition $P_1 \prec P_2 \prec \cdots \prec P_m$ of the vertices $\{1, 2, \cdots, n-1\}$ of a topological ordering of $G(L)$ such that
  (1) for every $v \in \{1, 2, \cdots, n-1\}$, if $v \in P_i$ then all predecessors of v belong to $P_1, \cdots, P_i$,
  (2) the subgraph induced by each P_i is transitively closed, and
  (3) m is minimum subject to the two conditions above.

The permutation Π in (P1) can be obtained by renumbering the vertices in the ordered

partition P_1 to P_m in increasing order. The first condition follows from the fact that the factors are formed by grouping the elementary matrices of L_Π in order from lowest-numbered to highest-numbered. The other conditions follow from the discussion in the preceding paragraphs. The graph model (P2) is not explicitly stated in Alvarado and Schreiber [3], although it is implicit in the description of their algorithm RP1.

3. Cholesky factorization. Now we consider the restriction of (P1) to Cholesky factors. Then the graph $G(L)$, viewed as an undirected graph, is a chordal graph. The gist of this section is that the chordality of $G(L)$ simplifies the problem a great deal, and enables the design of an $\mathcal{O}(n)$ algorithm for computing the partition, whereas previous algorithms [3] required time nonlinear in the number of edges of $G(L)$. The savings result from the fact that it suffices to consider the transitive reduction of $G(L)$, the elimination tree of L, instead of all the edges in $G(L)$.

The *elimination tree* of L is a directed tree $T = (V, E_T)$, whose vertices are the columns of L, with a directed edge $(j, i) \in E_T$ if and only if the lowest-numbered row index of a subdiagonal nonzero in the jth column of L is i. (The edge is directed from j to i.) The vertex i is the *parent* of j, and j is a *child* of i.

We define the *higher adjacency set* $hadj(j)$ to be the set of all vertices k adjacent to j in $G(L)$ such that k is numbered higher than j. If (j, i) is an edge in the elimination tree, the lowest-numbered vertex in $hadj(j)$ is i. The reader can verify that the elimination tree of the graph $G(L)$ in Fig. 1 is obtained by omitting the edges $(4, 6)$, $(5, 11)$, $(7, 10)$, and $(8, 10)$ from the graph.

A comprehensive survey of the role of elimination trees in sparse Cholesky factorization has been provided by Liu [13]. We will assume some knowledge of the properties of elimination trees, and in particular, the following result will be useful.

LEMMA 3.1. *If v is the parent of a vertex u in the elimination tree T, then $hadj(u) \subseteq \{v\} \cup hadj(v)$.* □

Our partitioning algorithm will require as input the elimination tree with vertices numbered in a topological ordering. It also requires the subdiagonal nonzero counts of each column of L, stored in an array $hd(v)$. The algorithm uses a variable *level* to partition the vertices; $level(v) = l$ implies that v belongs to the set P_l.

The idea of the algorithm is as follows. It examines the vertices of the elimination tree in increasing order. If a vertex v is a leaf of the tree, then it is included in the first *level*, which constitutes the vertices in P_1. Otherwise, it divides the children of v into two sets: C_1 is the subset of the children u such that the subgraph of $G(L)$ induced by u and v is transitively closed, and C_2 denotes the subset of the remaining children. Let l_1 denote the maximum *level* of a child in C_1 and l_2 denote the maximum *level* of a child in C_2. Set $l_i = 0$ if $C_i = \emptyset$. If C_1 is empty, or if $l_1 \leq l_2$, then v cannot be included in the same *level* as any of its children, and hence begins a new *level* $(l_2 + 1)$. Otherwise, $l_1 > l_2$, and v can be included together with some child $u \in C_1$ such that $level(u) = l_1$.

We now describe the details of an implementation. The vertices of the elimination tree are numbered in a topological ordering from 1 to n. The descendant relationships in the elimination tree are represented by two arrays of length n, *child* and *sibling*. $child(v)$ represents the first child of v, and $sibling(v)$ represents the right sibling of v, where the children of each vertex are ordered arbitrarily. If $child(v) = 0$, then v has no child and is a leaf of the elimination tree; if $sibling(v) = 0$, then v has no right sibling. The array $hd(.)$, also of length n, contains the *higher degree* of a vertex v (equal to $|hadj(v)|$). Our partitioning algorithm, Algorithm RPtree, is shown in Fig. 2. The reader can verify that $P_1 = \{1, 3, 4, 7, 8, 9\}$, $P_2 = \{2, 5, 6, 10\}$, and $P_3 = \{11\}$ for the

```
for v := 1 to n →
    if child(v) = 0 then {v is a leaf}
        level(v) := 1;
    else {v is not a leaf}
        u := child(v); l1 := 0; l2 := 0;
        while u ≠ 0 do
            if hd(u) = 1 + hd(v) then
                l1 := max{l1, level(u)};
            else {hd(u) < 1 + hd(v)}
                l2 := max{l2, level(u)};
            fi
            u := sibling(u);
        od
        if l1 ≤ l2 then {v begins a new level}
            level(v) := l2 + 1;
        else {l1 > l2, v can be included in level l1 }
            level(v) := l1;
        fi
    fi
rof
```

FIG. 2. *Algorithm RPtree.*

graph in Fig. 1.

The complexity of the algorithm is easily analyzed. For a given vertex v, we examine all of its children, and the operations associated with examining a child u can be performed in constant time. If we charge the cost of examining a child u of v to u, then each vertex in the elimination tree is charged at most once, since each child has a unique parent. Thus the time complexity of the algorithm is $\mathcal{O}(n)$. The space complexity is also $\mathcal{O}(n)$, since the elimination tree, the higher degrees, and the *level* information are all stored using arrays of length n.

4. Correctness of the algorithm. We now prove that Algorithm RPtree correctly solves (P1).

THEOREM 4.1. *Algorithm RPtree correctly solves* (P1) *when L is the unit lower triangular matrix in the LDL^T factorization of a symmetric, positive definite matrix.*

Proof. We consider problem (P2), the graph model of (P1), and the DAG $G(L)$, which viewed as an undirected graph is chordal. We will show that the ordered partition obtained by the algorithm satisfies the three conditions in (P2).

First we show that a vertex $v \in P_i$ only if all predecessors of v belong to $P_1, \cdots, P_i$. Since the elimination tree T is the transitive reduction of $G(L)$, any predecessor of v in the latter graph is a predecessor of v in T as well. Thus it suffices to consider the descendants of v in the elimination tree. Further, since the vertices in the elimination tree are topologically ordered, and the algorithm assigns *level* values to the vertices in increasing order, it suffices to consider the children of v in T. Finally, since the algorithm assigns *level* values to a vertex v such that

$$level(v) \geq \max\{level(u) : u \text{ is a child of } v\},$$

the result is true.

Second, we show that the subgraph induced by the vertices in a *level* is transitively closed. In the preceding paragraph, it was seen that *level* values are nondecreasing along any path from a leaf to the root on the elimination tree. Hence it follows that the vertices included in a *level* l by Algorithm RPtree can be expressed as the union (not necessarily disjoint) of certain paths in the elimination tree, where vertices on a path are constrained to have *level* values equal to l. Let $u_0, u_1, \cdots, u_p$ be such a path, which is maximal with respect to the property of having the same *level* value; then since the algorithm includes all these vertices in a single *level*,

$$hd(u_0) = 1 + hd(u_1) = 2 + hd(u_2) = \cdots = p + hd(u_p).$$

It follows from Lemma 3.1 that

$$hadj(u_0) = \{u_1\} \cup hadj(u_1) = \cdots = \{u_1, \ldots, u_p\} \cup hadj(u_p).$$

Hence the subgraph induced by the vertices in this path is transitively closed. Since the set of vertices in a *level* is the union of such paths, the result now follows.

It remains to show that m is the minimum number of ordered sets in the partition of $G(L)$ subject to the above conditions. We prove the result by induction on $(n-1)$, the number of vertices to be partitioned. The base case when this number is one is trivial. Assume inductively that Algorithm RPtree optimally partitions all chordal graphs with at most $n-1$ vertices (hence there are at most $n-2$ vertices to be partitioned).

Consider the partition $P_1 \prec P_2 \prec \cdots \prec P_m$ with m levels obtained by the algorithm on a chordal graph $G(L)$ with n vertices. Let T denote the elimination tree of $G(L)$ with vertices in a topological ordering. As shown in the proof of the second condition, P_1 consists of the union of vertices on certain paths on the elimination tree T, each path beginning from a leaf and ending at a vertex w such that *level*(w) is one, and *parent*(w) has *level* greater than one. (Thus these paths are maximal with respect to the condition that their *level* values are equal to one.) Let $u_0 < u_1 \ldots < u_p$ be such a path, and denote by u_{p+1} the *parent* of u_p in T.

The fact that the algorithm did not include u_{p+1} in P_1 together with u_p could be due to one of two reasons. If u_{p+1} has some child x with *level*$(x) >$ *level*$(u_p) = 1$, then the inclusion of u_{p+1} in P_1 would destroy the first condition of problem (P2). Otherwise, all the children of u_{p+1} belong to P_1. Now the algorithm would not include u_{p+1} in P_1 only if some child x satisfied $hd(x) < 1 + hd(u_{p+1})$. If this were the case, there is some vertex $w \in hadj(u_{p+1}) \setminus hadj(x)$. Thus the inclusion of u_{p+1} into P_1 would destroy the property that the subgraph induced by P_1 is transitively closed. Thus P_1 contains the maximum number of vertices possible from the vertices in $G(L)$.

Now consider the set of vertices Q which consists of those vertices of $G(L)$ not included in P_1. Then the induced subgraph $G(Q)$ has fewer than $n-1$ vertices, and by the inductive hypothesis, is partitioned optimally by Algorithm RPtree into $m-1$ ordered sets. Let T_Q denote the elimination tree of the subgraph $G(Q)$ obtained by removing vertices in P_1 from the elimination tree T of $G(L)$. From the preceding paragraph, since vertices in P_1 cannot be included in a *level* which contains the leaf vertices of T_Q, it follows that any partition of $G(L)$ requires at least $(m-1)+1 = m$ levels. This completes the proof. □

5. Experimental results. We implemented Algorithm RPtree and compared its performance with the RP1 and RP2 algorithms of Alvarado and Schreiber on eleven problems from the Boeing–Harwell collection [6]. All the algorithms were

implemented in C, within Alvarado's Sparse Matrix Manipulation System [1]. Each problem was initially ordered using the Multiple-Minimum-Degree ordering of Liu [12], and the structure of the resulting lower triangular factor L was computed. We call this the primary ordering step. Then Algorithms RP1, RP2, or RPtree were used in a secondary ordering step to reorder the structure of L to obtain the minimum number of partitions over reorderings that preserve the DAG $G(L)$. All three algorithms lead to the same number of levels in the partition since they solve the same problem.

The experiments were performed on a Sun SPARCstation IPC with 24 Mbytes of main memory and a 100 Mbyte swap space running SunOS 4.1 version of the Unix operating system. The unoptimized standard system compiler was used to compile the code. Recall that $\tau(A)$ is the number of nonzeros in the strict lower triangle of A; $\tau(L)$ is similarly defined. We scale these numbers by a thousand for convenience. In Table 1, we report the scaled values of $\tau(A)$ and $\tau(L)$, the CPU times taken by the primary and secondary ordering algorithms (in seconds), and the height of the elimination tree obtained from the primary ordering. (The fill and the etree height reported here are somewhat different from previously published values for the MMD ordering because of our use of SMMS. In SMMS, the problem datum is first converted to an element list from the Boeing–Harwell format before it is stored using sparse matrix data structures. This changes the initial matrix ordering which is input to the MMD algorithm, with the consequent change in the fill and etree height.)

Table 1 also reports the number of factors in the partitioned inverse of L. The number in the column "levels(new)" corresponds to the number of factors in the solution of the problem (P1), i.e., in the partition of the permuted matrix L_Π. The number in the column "levels(orig)" indicates the number of factors obtained when the unpermuted Cholesky factor L is partitioned. In the graph model (P2), this corresponds to replacing the first condition with the stricter condition:

> For every $v \in \{1, \cdots, n-1\}$, if $v \in P_i$, then all vertices numbered lower than v belong to $P_1, \cdots, P_i$.

TABLE 1

Comparison of execution times on a Sun SPARC*station* IPC *for three secondary reordering schemes with the* MMD *primary ordering. The parameters $\tau(A)$ and $\tau(L)$ have been scaled by a thousand for convenience.*

	Original data		MMD		Etree	CPU time (sec)			Levels	
Problem	n	$\tau(A)$	Time	$\tau(L)$	Height	RP1	RP2	RPtree	Orig	New
BCSPWR10	5,300	8.27	1.72	23.2	128	1.07	1.26	0.10	70	32
BCSSTK13	2,003	40.9	4.74	264	654	61.1	22.1	0.05	53	24
BCSSTM13	2,003	9.97	1.12	42.6	261	5.08	2.63	0.03	25	16
BLCKHOLE	2,132	6.37	0.73	53.8	224	3.15	2.58	0.05	24	15
CAN1072	1,072	5.69	0.72	19.4	151	0.78	0.92	0.02	21	16
DWT2680	2,680	11.2	1.82	49.9	371	2.43	2.45	0.05	50	36
LSHP3466	3,466	10.2	1.03	81.2	341	4.48	4.14	0.07	37	25
NASA1824	1,824	18.7	1.42	72.2	259	6.01	3.88	0.03	34	16
NASA4704	4,704	50.0	3.92	275	553	33.8	16.1	0.12	41	17
39x39 9pt	1,521	10.9	0.50	31.6	185	1.35	1.50	0.02	19	15
79x79 9pt	6,241	45.9	2.17	190	429	12.7	11.4	0.12	30	23

Alvarado and Schreiber [2] have shown that when the partitioned inverse is employed on a massively parallel computer such as the CM-2, the number of levels in the partitioned inverse representation determines the complexity of parallel triangular solution. On the other hand, the complexity of a conventional triangular solution algorithm is governed by the height of the elimination tree. Table 1 shows both these

quantities, and it is seen that the number of levels in the partitioned inverse is many times smaller (by a factor of sixteen on the average) than the elimination tree height. Hence the use of the partitioned inverse leads to much faster parallel triangular system solution on massively parallel computers.

An interesting feature in these results is that the number of levels seems to be only weakly dependent on the problem, the order of A, or the number of nonzeros in A or L. This number is between ten and forty and is significantly smaller than the order of the matrix A for most of these problems. If this observation holds true for larger instances of a wide collection of problems, then it will have a significant impact on the application of the ideas in this paper to parallel computing. For the $k \times k$ model grid problem ordered by the optimal nested dissection ordering, the height of the elimination tree is $3k + \Theta(1)$, while the number of levels is $2\log_2 k + \Theta(1)$.

Algorithm RPtree has $\mathcal{O}(n)$ time complexity while RP1 and RP2 are both nonlinear in the number of nonzeros in L. This is confirmed by the experiments: on the average problem in this test set, RPtree is more than *a hundred* times faster than RP1 or RP2, and the advantage increases with increasing problem size. From a practical perspective, the time needed by Algorithm RPtree is quite small when compared to the cost of computing the initial MMD ordering. This is not true of either the RP1 or the RP2 algorithm. An equally important advantage of Algorithm RPtree is that it requires only $\mathcal{O}(n)$ additional space, whereas both RP1 and RP2 require $\mathcal{O}(\tau(L))$ additional space.

We have also experimented with a variant of the Minimum-Length-Minimum-Degree (MLMD) ordering [5] as the primary ordering, but we do not report detailed results because Timely Communications are by definition brief. The MLMD ordering incurs a great deal more fill in L than the MMD algorithm, and its current, fairly straightforward implementation is quite slow compared to the MMD algorithm. We believe an implementation comparable in sophistication to the MMD algorithm should not be significantly slower than MMD, and may also reduce the number of fills. In spite of the larger number of fills, the MLMD ordering is more effective in almost all cases than MMD in reducing the number of levels in the partition of both L and L_Π. In some cases, the initial number of levels obtained when MLMD is used as the primary ordering is lower than the final number of levels obtained with MMD after the secondary reordering.

When the MLMD ordering is used, Algorithm RPtree has an even greater time advantage over the RP1 and RP2 algorithms, since the former works with the elimination tree, while the latter algorithms require the structure of the Cholesky factor. For instance, on BCSSTK13, RP2 takes 461 seconds, while RPtree requires only 0.07 seconds.

6. Extensions. There are two directions in which the results in this paper may be extended.

Given the factorization $A = LDL^T$ of a symmetric, positive definite matrix, consider the filled matrix $F = L + L^T$ and the corresponding chordal undirected graph $G_u(F)$. In problem (P3) we ask for the minimum number of factors m in the partitioned inverse representation of L over all vertex orderings that preserve the structure of the filled graph $G_u(F)$ (rather than the DAG $G(L)$ and the corresponding elimination tree of L as (P2) does). A solution to this problem would reduce the number of factors in the partitioned inverse over the number required in (P2). Such an ordering would have to be applied to the original matrix A, *before* the computation of the factorization. This problem turns out to be much harder than (P2), and

many subtle issues are involved in the solution of this problem. It can be solved by a *generalization* of the Jess and Kees ordering [11]. (However, the Jess and Kees ordering by itself will not work.) An efficient algorithm to solve (P3) that makes use of the clique tree data structure will be reported in [14]. It is heartening that the above ordering is also appropriate for efficiently computing the factorization in parallel on massively parallel machines.

The ideas in this paper can also be applied to the general unsymmetric problem to obtain a more efficient partitioning algorithm than RP2. It turns out that the transitive reduction of the directed graph $G(L)$ could be used instead of $G(L)$ at several places in the RP2 algorithm. It is necessary to implement this idea to see the net computational savings the use of transitive reduction may bring in this context. Other applications of transitive reduction in unsymmetric sparse factorizations have been recently considered by Eisenstat, Gilbert, and Liu [7], [10].

REFERENCES

[1] F. L. ALVARADO, *Manipulation and visualization of sparse matrices*, ORSA J. Comput., 2 (1990), pp. 180–207.

[2] F. L. ALVARADO AND R. SCHREIBER, *Fast parallel solution of sparse triangular systems*, 13th IMACS World Congress on Computation and Applied Mathematics, Dublin, July 1991.

[3] ———, *Optimal parallel solution of sparse triangular systems*, SIAM J. Sci. Statist. Comput., 13 (1992), to appear.

[4] F. L. ALVARADO, D. C. YU, AND R. BETANCOURT, *Partitioned sparse A^{-1} methods*, IEEE Trans. Power Systems, 5 (1990), pp. 452–459.

[5] R. BETANCOURT, *An efficient heuristic ordering algorithm for partial matrix refactorization*, IEEE Trans. Power Systems, 3 (1988), pp. 1181–1187.

[6] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[7] S. C. EISENSTAT AND J. W.-H. LIU, *Exploiting structural symmetry in unsymmetric sparse symbolic factorizations*, Tech. Report 90-12, Computer Science, York University, North York, Ontario, Canada, 1990.

[8] M. K. ENNS, W. F. TINNEY, AND F. L. ALVARADO, *Sparse matrix inverse factors*, IEEE Trans. Power Systems, 5 (1990), pp. 466–472.

[9] J. R. GILBERT, *Predicting structure in sparse matrix computations*, Tech. Report 86-750, Computer Science, Cornell University, Ithaca, NY, 1986.

[10] J. R. GILBERT AND J. W-H. LIU, *Elimination structures for unsymmetric sparse LU factors*, Tech. Report 90-11, Computer Science, York University, North York, Ontario, Canada, 1990.

[11] J. G. LEWIS, B. W. PEYTON, AND A. POTHEN, *A fast algorithm for reordering sparse matrices for parallel factorization*, SIAM J. Sci. Statist. Comput., 6 (1989), pp. 1146–1173.

[12] J. W.-H. LIU, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.

[13] ———, *The role of elimination trees in sparse factorization*, SIAM J. Matrix. Anal. Appl., 11 (1990), pp. 134–172.

[14] A. POTHEN AND X. YUAN, *A clique tree algorithm for optimally reordering sparse Cholesky factors for parallel triangular solution*, work in preparation, 1991.