

- [15] —, *A clique tree algorithm for partitioning a chordal graph into transitive subgraphs*. Work in preparation, 1992.
- [16] A. POTHEN AND F. L. ALVARADO, *A fast reordering algorithm for parallel sparse triangular solution*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 645–653.

multiplication involves as many virtual processors as the number of nonzeros.

There is some flexibility in assigning elementary matrices to factors, and this may be exploited to reduce the disparity between the number of nonzeros in the different factors. If no successor of  $v \in P_i$  is included in  $P_i$  or  $P_{i+1}$ , then it is possible to include  $v$  in the latter factor, while preserving the fact that the column subgraph  $G(P_{i+1})$  is transitively closed.

A second issue is to reduce the number of factors in the partitioned inverse further by permitting some fill during the inversion of the factors, taking care that the method remains numerically stable. A third issue is to partition the matrix  $L$  into a block triangular matrix ensuring only that the diagonal blocks incur no fill upon inversion. Then the subdiagonal blocks of  $L$  need not be inverted, since the solution to the triangular system can be decomposed in the usual way into the solution of several subsystems. The solution vector associated with each subsystem can be computed using the partitioned inverse of its diagonal block, and then by block back-substitution the contribution this solution vector makes to higher numbered subsystems can be eliminated. Fourthly, it may be possible to use the transitive reduction of the DAG  $G(L)$  to design partitioning algorithms which are more efficient in practice than RP1 or RP2.

#### 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, *Optimal parallel solution of sparse triangular systems*. SIAM J. Sci. Stat. Comput., to appear, 1992.
- [3] F. L. ALVARADO, D. C. YU, AND R. BETANCOURT, *Partitioned sparse  $A^{-1}$  methods*, IEEE Trans. Power Systems, 5 (1990), pp. 452–459.
- [4] E. ANDERSON AND Y. SAAD, *Solving sparse triangular systems on parallel computers*, International Journal of High Speed Computing, 1 (1989), pp. 73–95.
- [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. Softw., 15 (1989), pp. 1–14.
- [7] S. C. EISENSTAT, M. T. HEATH, C. S. HENKEL, AND C. H. ROMINE, *Modified cyclic algorithms for solving triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 589–600.
- [8] J. R. GILBERT, *Predicting structure in sparse matrix computations*, Tech. Report 86-750, Computer Science, Cornell University, 1986.
- [9] S. W. HAMMOND AND R. SCHREIBER, *Efficient ICCG on a shared memory multiprocessor*. International Journal of High-Speed Computing, to appear, 1992.
- [10] N. J. HIGHAM AND A. POTHEN, *The stability of the partitioned inverse approach to parallel sparse triangular solution*, Tech. Report CS-92-52, Computer Science, University of Waterloo, Oct 1992. (Also Numerical Analysis Report No. 222, Department of Mathematics, University of Manchester, England.) Submitted to *SIAM J. Sci. Stat. Comput.*
- [11] G. LI AND T. F. COLEMAN, *A new method for solving triangular systems on a distributed memory multiprocessor*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 382–396.
- [12] J. W. H. LIU, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Trans. Math. Softw., 11 (1985), pp. 141–153.
- [13] J. W. H. LIU, *The role of elimination trees in sparse factorization*, SIAM J. Mat. Anal. Appl., 11 (1990), pp. 134–172.
- [14] B. W. PEYTON, A. POTHEN, AND X. YUAN, *Partitioning a chordal graph into transitive subgraphs for parallel sparse triangular solution*. Work in preparation, Oct. 1992.

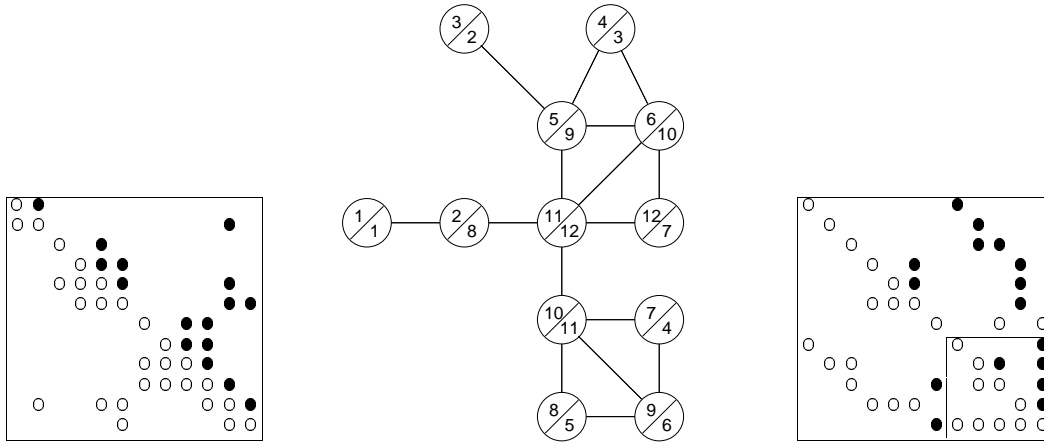


FIG. 8. Partitioning over the class of perfect elimination orderings of the filled graph  $G(F)$ . The original ordering is shown on the left, and an ordering and a partition that solve problem (Pr3) are shown on the right. The figure illustrates that the upper and lower triangles of the matrix on the left are not preserved under the permutation, but the structure of  $F$  is preserved.

is shown on the right in  $G(F)$ .) It is easily seen that the permutation moves some elements of the lower triangle into the upper triangle and vice versa, and that it preserves the structure of  $F$ .

Problem (Pr3) turns out to be much harder than (Pr2), but can be solved by developing several results concerning transitive perfect elimination orderings: i.e., perfect elimination orderings of subgraphs of chordal graphs which make them transitively closed subgraphs as well. Algorithms to solve (Pr3) will be reported in [14, 15].

The numerical stability of the partitioned inverse approach to triangular solution has been investigated in [10]. The method has been shown to be normwise backward stable and normwise forward stable when a certain scalar which can be considered as a ‘growth factor’ is small; this condition is satisfied when  $L$  is well-conditioned. However, the method does not enjoy the componentwise backward and forward stability properties of the substitution algorithm. If the factors are not invertible in place, then the partitioned inverse approach may be unstable even when the ‘growth factor’ is small.

There are several other directions to explore in future work.

The first issue concerns the uniqueness of the minimum-cardinality partition. Algorithms that we have designed for computing the partitions in all of the above problems belong to the class of greedy algorithms. Further, for each  $i$ , they include as many elementary matrices as possible into a factor  $P_i$  subject to the condition that the factors  $P_1, \dots, P_{i-1}$  have been obtained similarly. It is easily seen, however, that minimum cardinality partitions need not be unique: for instance, for problem (Pr2-Cholesky), it is possible to design an algorithm that processes the vertices of the elimination tree from  $n$  down to 1 to compute the partition in the order  $P_m, \dots, P_1$ . Such a partition would put, for each  $i$ , as many elementary matrices as possible into  $P_i$  subject to the condition that factors  $P_m, \dots, P_{i+1}$  have been obtained similarly. In the context of highly parallel triangular solution, it may be preferable to have each factor contain roughly the same number of nonzeros, since the matrix-vector

by substitution methods. We see that when  $L$  has a fairly rich structure the partitioned inverse approach is much better than the substitution method, but when  $L$  is very sparse there is little gained. The use of an MLMD primary ordering improves both substitution and partitioned methods. However, with the introduction of the additional fill in the exact factor  $L_3$  (compared with  $L_2$ ), the number of *levels* in  $G(L)$  increases sharply (as does the time for substitution) while the number of factors in the best reordered partition drops dramatically. The difference in the solution time, even for this problem of modest size, is about a factor of twenty. Thus we conclude that the method can be quite useful in highly parallel machines when the matrix  $L$  has a rich enough structure, as happens when it is an exact triangular factor.

**5. Concluding remarks.** We have considered the characterization and computation of partitioned inverse representations involving the fewest factors of a given sparse triangular matrix  $L$ . Three variants of this problem have been considered in this paper; for completeness, here we will also describe a fourth variant which will be considered in a future paper. The four problems minimize the number of factors in the in-place partitioned inverse of  $L$

- over a fixed row and column ordering of  $L$ —problem (Pr1),
- over symmetric permutations of  $L$  which preserve its lower triangular structure—problem (Pr2),
- over symmetric permutations which preserve its lower triangular structure, when  $L$  is a Cholesky factor—problem (Pr2-Cholesky), and
- over symmetric permutations of the *filled matrix*  $F = L + D + L^T$  which preserve the structure of  $F$ , again when  $L$  is a Cholesky factor—problem (Pr3).

When  $L$  is obtained from unsymmetric, symmetric indefinite, or incomplete factorization, Algorithms RP1 and RP2 may be used to compute the best reordered partition in  $\mathcal{O}(ne)$  time and space. When  $L$  is a Cholesky factor, Algorithm RPtree is an extremely efficient  $\mathcal{O}(n)$  time and space algorithm for computing the reordered partition corresponding to the third problem listed above. We believe that the results in this paper demonstrate the potential superiority of the partitioned inverse approach over the conventional substitution algorithm for sparse triangular solution on highly parallel computers.

We now discuss the fourth variant of the partitioning problem. Given the factorization  $A = LDL^T$  of a symmetric, positive definite matrix, consider the filled matrix  $F = L + D + L^T$  and the corresponding undirected graph  $G(F)$  which is chordal. In problem (Pr3) 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  $G(F)$  (rather than preserving the structure of the DAG  $G(L)$  as (Pr2) does). Such an ordering would have to be applied to the original matrix  $A$ , *before* the computation of the factorization.

We illustrate this idea by means of the example in Fig. 8. The filled matrix  $F$  shown on the left corresponds to the Cholesky factor  $L$  in Fig. 5, and the filled graph  $G(F)$  is the undirected graph corresponding to the DAG  $G(L)$  in the latter figure. It can be shown that a solution to (Pr3) would partition the vertices of  $G(F)$  into two factors  $P_1 = \{1, 3, 4, 7, 8, 9, 12\}$  and  $P_2 = \{2, 5, 6, 10, 11\}$ . A renumbering of the columns and rows of  $F$  obtained by numbering the vertices in  $P_1$  before  $P_2$  (and topologically within each  $P_i$ ) and the corresponding partition are shown in the matrix on the right of Fig. 8. (The corresponding vertex ordering

$n$	$\tau(L)$	Levels in $G(L)$	Substitution Time	Factors	Partitioned soln. time
256	32,896	256	7.34	1	0.04
512	131,328	512	50.22	1	0.21

TABLE 2

*CM-2 times (seconds) for full matrix substitution and partitioned solution.*

Matrix	Ordering	Factor- ization	$\tau(L)$	Levels in $G(L)$	Substitution Time	Factors	Partitioned soln. time
$L_1$	RCM	ILU	23,526	823	19.22	816	11.66
$L_2$	MLMD	ILU	26,793	78	1.38	66	1.20
$L_3$	MLMD	exact	118,504	311	16.78	16	0.87

TABLE 3

*CM-2 times (seconds) for sparse triangular substitution and partitioned solution.*

for computing a matrix-vector product involving a triangular matrix. Hence the code for triangular solution using the partitioned inverse approach is nearly identical to that for our substitution method in that the inner loop involves the same operations. But the number of executions of the loop in the partitioned inverse approach is equal to the number of factors in the partition of  $L$  rather than the number of *levels* in its graph.

We report the CM performance of these two methods for an  $n \times n$ , dense, unit lower-triangular matrix  $L$ . Our results are obtained with CM Fortran in the ‘slice-wise’ execution model, which treats each Weitek chip of the CM-2 as a processor. For this experiment we used 256 Weitek processors on the Connection Machine at NASA Ames. Results are given in Table 2. Clearly, the partitioned method is superior, by a factor roughly equal to the ratio of the number of *levels* in  $G(L)$  (which is  $n$  in this case) to the number of factors in the partition of  $L$  (one).

Next, we performed an experiment using a sparse matrix  $A$  of order 4037 obtained from a triangular mesh in the region around a three-element airfoil. Three unit lower triangular matrices  $L_1$ ,  $L_2$ , and  $L_3$  were obtained by approximate factorization.  $L_1$  is obtained by an incomplete LU factorization of  $A$ ; we carry out the Gaussian elimination process, but we allow nonzeros in  $L$  (and  $U$ ) only where there is a nonzero in  $A^2$ . The ordering of  $A$  is obtained from a lexicographic sort of the  $(x, y)$  coordinates of the grid which leads to the matrix; this ordering produces a large number of *levels* in  $G(L)$ .  $L_2$  is the incomplete  $LU$  factor obtained when a variant of MLMD is used as the primary ordering of  $A$ .  $L_3$  is the exact lower-triangular factor of  $A$ , with the same primary ordering as for  $L_2$ .

In Table 3 we give the size of these factors, the number of *levels* (this is proportional to the time required for our parallel substitution algorithm), and the number of factors, which is in practice proportional to the time required by the partitioned inverse approach.

These results confirm that the time required to solve a triangular system by partitioning of the inverse is quite well predicted by the number of factors in the partition. It also shows that the number of *levels* in  $G(L)$  is a good predictor of the time required for solution

MLMD ordering is more effective in almost all cases than MMD in reducing the number of factors in the partition of both  $L$  and  $L_Q$ . In some cases, the initial number of factors obtained when MLMD is used as the primary ordering is lower than the final number of *levels* obtained with MMD after the secondary reordering ( $Q$  in problem (Pr2)). However, because of the increased fill, choosing between MMD and MLMD as the primary ordering is not straightforward.

**4.2. Triangular solution on a CM-2.** Now we compare the performance of the partitioned inverse approach with the conventional substitution algorithm for triangular solution on a CM-2.

An efficient parallel substitution algorithm was implemented in CM Fortran, a dialect of Fortran 90. The data structure consists of several arrays of length equal to  $\tau(L)$ . We associate a set of  $\tau(L)$  virtual processors with the nonzeros, one with each position in these arrays. We store the factors  $\hat{L}$  and  $D$  where  $L = \hat{L}D$  and  $\hat{L}$  is unit triangular, and solve  $L\underline{x} = \underline{b}$  via  $\hat{L}\underline{x} = D^{-1}\underline{b}$ , since this removes a multiplication from the inner loop. The matrix  $\hat{L}$  is stored as a one-dimensional array containing its nonzeros in column-major order. In addition, the *level* of vertex  $j$  in  $G(L)$  is stored along with  $\hat{L}_{ij}$ . The elements of  $\underline{b}$  are stored at the processors containing the corresponding diagonal elements of  $\hat{L}$ . The solution  $\underline{x}$  overwrites  $\underline{b}$ . Finally, a Boolean vector indicates the location of the diagonal elements in the arrays. This vector thus segments the arrays into columns of differing lengths—in other words, the matrix is stored as a ragged array of columns of nonzeros. The Connection Machine software provides some operations for such data structures. It allows broadcast of values from diagonal elements to all elements of the corresponding column (called a segmented copy scan) and summation of the values in a column (with a segmented add scan). Also, the Connection Machine router allows processors to send data to any other processor or read from any other processor; this is expressed using a vector-valued subscript in CM Fortran. Calls to utility library routines were used for the scan operations, which are not part of CM Fortran.

The substitution algorithm for triangular solution loops sequentially over *levels* of  $G(L)$  starting with the source vertices (*level zero*). At the beginning of step  $\ell$ , those elements  $x_j$  for which  $level(j) = \ell$  are known. Recall that  $x_j$  is stored in the processor holding  $\hat{L}_{jj}$ . These known values of  $\underline{x}$  are sent to the processors holding the corresponding column of  $\hat{L}$  by a segmented copy scan. These processors then multiply their  $\hat{L}$  value by the element of  $\underline{x}$  they receive. The router is then used to permute all these products into row-major order, so that the elements of each set  $R_i \equiv \{\hat{L}_{ij}x_j \mid L_{ij} \neq 0 \text{ and } level(j) = \ell\}$  are stored in consecutive locations. The vector-valued subscript used to accomplish this permutation is computed in a preprocessing step. The rows  $R_i$  now form a ragged array. A segmented add scan forms the sums of these partial results within rows. Finally, the router is used to send the sum of the elements of  $R_i$  to the processor holding  $\hat{L}_{ii}$  and  $b_i$  where it can be subtracted from  $b_i$ . (In fact, our code avoids this last subtraction entirely, doing it as part of the add scan above.)

Thus an iteration of the loop involves one parallel multiplication, one copy scan and one add scan, and two uses of the router for permutation of data. The time required to set up and load the data structure, including the computation of the permutation used, the loading of  $\underline{b}$  and extraction of  $\underline{x}$ , was not timed.

The sequential algorithm for solving a triangular system is quite similar to the algorithm

TABLE 1

Comparison of execution times on a Sun SPARCstation 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.

Problem	Original Data		MMD		etree height	CPU time (sec)			Factors	
	$n$	$\tau(A)$	time	$\tau(L)$		RP1	RP2	RPtree	(Pr1)	(Pr2)
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

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 factors in the partitioned inverse is several fold smaller (by a factor of sixteen on the average) than the elimination tree height. Hence the use of the partitioned inverse potentially leads to much faster parallel triangular system solution on massively parallel computers.

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 factors (in (Pr1) and (Pr2)) is  $2 \log_2 k + \Theta(1)$ . The results in Table 1 show that the number of factors for these irregular problems is only weakly dependent on the order of  $A$ , compatible with logarithmic growth.

The RPtree algorithm has  $\mathcal{O}(n)$  time complexity while RP1 and RP2 are both  $\mathcal{O}(n\tau(L))$  algorithms (recall that  $\tau(L) \equiv e$ ). 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 the RPtree algorithm is quite small when compared to the cost of computing the initial MMD ordering. An equally important advantage of the RPtree algorithm is that it requires only  $\mathcal{O}(n)$  additional space, whereas both RP1 and RP2 require  $\mathcal{O}(\tau(L))$  additional space. However, Algorithms RP1 and RP2 can be used to partition triangular factors arising from approximate or incomplete Cholesky factorizations as well as unsymmetric and symmetric indefinite factorizations.

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 here. 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 fill. In spite of the greater fill, the

of the elimination tree [13] is that if  $v$  is the parent of a vertex  $u$  in the elimination tree, then  $hadj(u) \subseteq \{v\} \cup hadj(v)$ . Hence  $hd(u) \leq 1 + hd(v)$ . On the other hand, if  $u$  and  $v$  can be included in the same transitively closed column subgraph, then  $hadj(u) \supseteq \{v\} \cup hadj(v)$ . It then follows that  $u$  and  $v$  can be possibly included in the same column subgraph only if  $hadj(u) = \{v\} \cup hadj(v)$ , or equivalently,  $hd(u) = 1 + hd(v)$ . Furthermore, if  $v$  has a child  $u$  not satisfying the degree condition, then  $v$  but not  $u$  is adjacent to some higher numbered vertex  $x$ , and hence  $v$  cannot belong to the same *member* as  $u$ . Thus we partition the children of  $v$  into two subsets:  $C_1$  consists of children  $u$  such that  $u$  and  $v$  can be included in the same column subgraph;  $C_2$  includes the rest of its children. It follows that if  $m_i$  is the maximum *member* value among vertices in  $C_i$ , then the inclusion of  $v$  into a column subgraph containing a child preserves transitivity only if  $m_1 > m_2$ .

It can be established by induction that Algorithm RPtree solves (Pr2) by partitioning  $G(L)$  into the minimum number of factors over all topological orderings [16].

**4. Experimental Results.** In this section we provide experimental results to demonstrate the superiority of the partitioned inverse approach over the conventional substitution algorithm for highly parallel triangular solution. First we describe the performance of the various partitioning algorithms, and then we report results for triangular solution on a CM-2.

**4.1. Partitioning algorithms.** We implemented Algorithms RP1, RP2, and RPtree and compared their performances 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 factors 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 the SunOS 4.1 version of the Unix operating system. The unoptimized standard system compiler was used to compile the code. Let  $\tau(A)$  denote the number of nonzeros in the strict lower triangle of  $A$ ;  $\tau(L)$  is then  $e$ , the number of edges in  $G(L)$ . 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.

Table 1 also reports the number of factors in the partitioned inverse of  $L$ . The number in the column 'Factors(Pr1)' corresponds to the number of factors in the solution of problem (Pr1), i.e., the best no-fill partition problem. The number in the column 'Factors(Pr2)' indicates the number of factors in the solution of problem (Pr2), i.e., the best reordered partitioning problem. Note the substantial decrease in the number of factors obtained by the permutation.

Later results in this section will show that when the partitioned inverse is employed on a highly parallel computer, the number of factors in the partitioned inverse determines the

---

**Input:** The elimination tree of a DAG  $G(L)$  and the higher degrees of the vertices.

**Output:** A mapping of the vertices such that  $member(v) = \ell$  implies that  $v \in P_\ell$ .

**for**  $v := 1$  to  $n \rightarrow$

**if**  $child(v) = 0$  **then** { $v$  is a leaf}

$member(v) := 1$ ;

**else** { $v$  is not a leaf}

$u := child(v)$ ;  $m_1 := 0$ ;  $m_2 := 0$ ;

**while**  $u \neq 0$  **do**

**if**  $hd(u) = 1 + hd(v)$  **then**

$m_1 := \max\{m_1, member(u)\}$ ;

**else** { $hd(u) < 1 + hd(v)$ }

$m_2 := \max\{m_2, member(u)\}$ ;

**fi**

$u := sibling(u)$ ;

**od**

**if**  $m_1 \leq m_2$  **then** { $v$  begins a new factor}

$member(v) := m_2 + 1$ ;

**else** { $m_1 > m_2$ ,  $v$  can be included in a factor which includes a child}

$member(v) := m_1$ ;

**fi**

**fi**

**rof**

FIG. 7. Algorithm RPtrree.

---

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. Algorithm RPtrree is shown in Fig. 7.

The reader can verify that  $P_1 = \{1, 3, 4, 7, 8, 9\}$ ,  $P_2 = \{2, 5, 10\}$ , and  $P_3 = \{6, 11, 12\}$  for the graph in Fig. 5. The time and space complexities of the algorithm are easily shown to be  $\mathcal{O}(n)$ . We turn to a discussion of the correctness of the algorithm.

Condition 1 of problem (Pr2) requires that if a vertex  $v$  belongs to  $P_\ell$ , then all predecessors of  $v$  must belong to  $P_1, \dots, P_\ell$ . The elimination tree  $T$ , being the transitive reduction of the DAG  $G(L)$ , preserves path structure: i.e., there exists a directed path from  $v$  to  $w$  in  $G(L)$  if and only if there is a (possibly some other) directed path from  $v$  to  $w$  in the elimination tree  $T$ . Hence the predecessors of a vertex in  $G(L)$  remain its predecessors in the elimination tree. Further, since we assign  $member$  values in a topological ordering of the vertices in the elimination tree, to satisfy Condition 1 we need consider only the children of a vertex  $v$  among its predecessors. Now since Algorithm RPtrree assigns  $member$  values such that  $member(v)$  is greater than or equal to  $member(u)$  for any child  $u$ , the condition is satisfied.

Condition 2 requires that each factor  $P_\ell$  be transitively closed. An important property

P1 partitions  $L$  into six factors as shown on the left; Algorithms RP1, RP2, or RPtree will partition it into three factors as shown on the right.

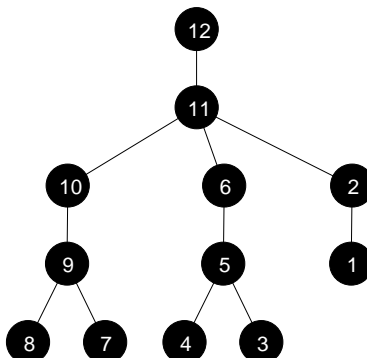


FIG. 6. The elimination tree of the Cholesky factor in Fig. 5.

The *elimination tree* of  $L$  (equivalently  $G(L)$ ) 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  $j$ -th 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  $i$ . If  $(j, i)$  is an edge in the elimination tree, the lowest-numbered vertex in  $\text{hadj}(j)$  is  $i$ . The elimination tree of the graph  $G(L)$  in Fig. 5 is shown in Fig. 6. (The vertex numbering corresponds to the original ordering shown on the left in Fig. 5.) A comprehensive survey of the role of elimination trees in sparse Cholesky factorization has been provided by Liu [13].

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  $v$  of  $L$ , stored in an array  $hd(v)$  (the higher degree of  $v$ ). The algorithm uses a variable *member* to partition the vertices;  $\text{member}(v) = \ell$  implies that  $v$  belongs to the set  $P_\ell$ .

Unlike Algorithms RP1 and RP2 which compute the factors  $P_1, \dots, P_m$  in that sequence, Algorithm RPtree examines the vertices of the elimination tree in increasing order of their numbers. If a vertex  $v$  is a leaf of the tree, then it is included in the first *member* (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 column subgraph of  $G(L)$  induced by  $u$  and  $v$  is transitively closed, and  $C_2$  denotes the subset of the remaining children. Let  $m_1$  denote the maximum *member* value of a child in  $C_1$  and  $m_2$  denote the maximum *member* value of a child in  $C_2$ . Set  $m_i = 0$  if  $C_i = \emptyset$ . If  $C_1$  is empty, or if  $m_1 \leq m_2$ , then we will show that  $v$  cannot be included in the same *member* as any of its children, and hence  $v$  begins a new *member* ( $m_2 + 1$ ). Otherwise,  $m_1 > m_2$ , and  $v$  can be included together with some child  $u \in C_1$  such that  $\text{member}(u) = m_1$ .

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*. The array  $\text{child}(v)$  represents the first child of  $v$ , and  $\text{sibling}(v)$  represents the right sibling of  $v$ , where

vertices, we need to maintain only the sets  $\mathcal{E}$  and  $\mathcal{E}^+$  in the algorithm. Thus we can dispense with the processing of vertices by *level* values.

The second improvement is to reduce the cost of checking condition 2 in Algorithm RP1. If  $u$  and  $v$  are both numbered vertices which have been included in the current factor  $P_k$ , and  $v$  is a successor of  $u$ , then  $\text{hadj}(v) \subseteq \text{hadj}(u)$ , otherwise  $v$  would have failed condition 2. Thus we need not consider vertex  $u$  when applying the requirements of condition 2 to a vertex that is also a successor of  $v$ . We make use of this in the faster implementation by deleting from  $\text{pred}(w)$  some of the predecessors of  $w$  that need not be examined in checking condition 2. In the situation above, when  $v$  is included in  $P_k$  we remove  $u$  from the predecessor sets of  $v$ 's successors, thus avoiding some of the unnecessary checking.

In condition 2', the test whether  $u \in P_k \cap \text{pred}(v)$  can be done efficiently by maintaining an array that maps each vertex to the factor to which it has been assigned. Fig. 4 contains a description of Algorithm RP2.

The worst-case time complexity of Algorithm RP2 is  $\mathcal{O}(ne)$  as well (and there are DAGs which attain this bound), though practically the above improvements should reduce the running times in many cases.

**3. Cholesky factorization.** Now we consider the restriction of (Pr2) to Cholesky factors. Then the graph  $G(L)$  viewed as an undirected graph is *chordal*; i.e., every cycle with more than three edges has a *chord*, an edge joining two nonconsecutive vertices on the cycle. The chordality of  $G(L)$  simplifies the problem a great deal since it suffices to consider the transitive reduction of  $G(L)$ , the elimination tree, instead of  $G(L)$ . This simplification enables the design of an  $\mathcal{O}(n)$ -time and space algorithm (Algorithm RPtree) for computing the partition.

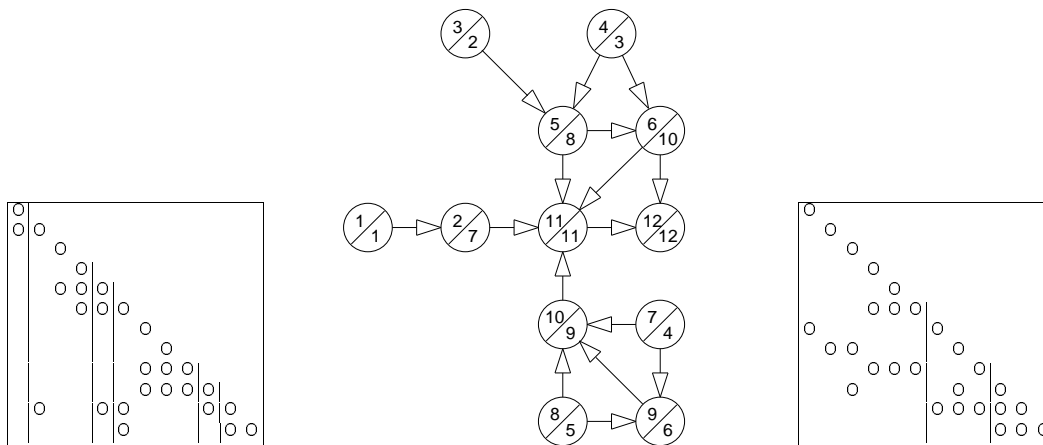


FIG. 5. A Cholesky factor  $L$ , its DAG, and its partitions. The original ordering and the partition found by Algorithm P1 are shown on the left, and the ordering and partition found by Algorithm RPtree are shown on the right.

In Fig. 5, we display the structure of a Cholesky factor  $L$  and the associated chordal graph  $G(L)$ . The vertex numberings on the left in  $G(L)$  correspond to the matrix on the left, and those on the right correspond to the reordered matrix shown on the right. Algorithm

the worst-case, this cost is  $\sum_{v \in V} d_I(v)d_O(v)$ , where  $d_I(v)$  is the indegree and  $d_O(v)$  is the outdegree of  $v$ . Since  $d_I(v) \leq n - 1$ , and  $\sum_{v \in V} d_O(v) = e$ , the time complexity of the algorithm is  $\mathcal{O}(ne)$ . If we assume that the indegrees and outdegrees are bounded by  $d$ , then the complexity is  $\mathcal{O}(d^2n)$ . The space complexity is  $\mathcal{O}(e)$ .

---

Input: A lower triangular matrix  $L = L_1 \cdots L_n$  and its DAG  $G(L)$ .

Output: permutation  $Q : V \longrightarrow \{1, \dots, n\}$  and a partition of  $L$  into factors.

**forall**  $v \in V$  **do**

$pred(v) \leftarrow \{u : L_{vu} \neq 0\}; \quad count(v) \leftarrow indegree(v);$

**od**

$i \leftarrow 0; \quad \{i \text{ elementary matrices have been included in factors}\}$

$k \leftarrow 1; \quad P_k \leftarrow \emptyset; \quad \{P_k \text{ is the factor being computed}\}$

$\mathcal{E} \leftarrow \{v \in V : count(v) = 0\}; \quad \{\text{vertices eligible to be tested for inclusion in current factor}\}$

$\mathcal{E}^+ \leftarrow \emptyset; \quad \{\text{vertices eligible to be tested for inclusion in the next factor}\}$

**while**  $i < n$  **do**

**while**  $\mathcal{E} \neq \emptyset$  **do**

choose  $v \in \mathcal{E}$  and delete it from  $\mathcal{E}$ ;

**if** ( [Condition 2'] Every successor of  $v$  is a successor of all

$u \in P_k \cap pred(v)$  ) **then**

$i \leftarrow i + 1; \quad Q(v) \leftarrow i; \quad P_k \leftarrow P_k \cup \{v\};$

**for** every successor  $w$  of  $v$  **do**

$pred(w) \leftarrow pred(w) \setminus pred(v); \quad count(w) \leftarrow count(w) - 1;$

**if**  $count(w) = 0$  **then**  $\mathcal{E} \leftarrow \mathcal{E} \cup \{w\}; \quad \mathbf{fi}$

**od**

**else**

include  $v$  in  $\mathcal{E}^+$ ;

**fi**

**od**

$k \leftarrow k + 1; \quad P_k \leftarrow \emptyset; \quad \mathcal{E} \leftarrow \mathcal{E}^+; \quad \mathcal{E}^+ \leftarrow \emptyset;$

**od**

FIG. 4. *Algorithm RP2.*

---

At the expense of additional space, in most cases we can reduce the running time required by Algorithm RP1 by incorporating two enhancements.

The first improvement is that a vertex need not be tested for inclusion into a factor  $P_k$  until all of its predecessors have been numbered. To accomplish this, in  $count(v)$  we count the number of unnumbered predecessors of each vertex  $v$ ; initially, this is its indegree. When this count becomes zero, we include  $v$  in a set  $\mathcal{E}$  of vertices eligible to be tested for inclusion in a factor  $P_k$ .

If an eligible vertex  $v$  satisfies condition 2', then it is deleted from  $\mathcal{E}$  and included in the factor  $P_k$ . Otherwise, it is included in  $\mathcal{E}^+$ , the set of vertices eligible to be tested for inclusion in the next factor  $P_{k+1}$ . Further, since newly eligible vertices are adjacent to currently eligible

Alvarado, Yu, and Betancourt did not consider the issue of optimality, but later it was proved by Alvarado and Schreiber [2] that Algorithm P1 solves problem (Pr1).

**Best reordered partitions.** Now we describe Algorithm RP1 that solves the reordered partitioning problem (Pr2). A vertex  $v$  in the DAG  $G(L)$  is a *source* if there are no edges directed into  $v$ : i.e., there are no edges  $(u, v)$ . The *level* of a vertex  $v$  is the length of a longest directed path into  $v$ . It follows that if  $v$  is a source, then  $level(v) = 0$ ; furthermore, if  $v$  is not a source, then  $level(v)$  is the length of a longest path from a source to  $v$ . The *level* values of all the vertices of  $G(L)$  can be computed in  $\mathcal{O}(e)$  time. We define the set  $hadj(v)$  to be the set of all vertices adjacent to  $v$  and numbered higher than  $v$ .

---

Input: A lower triangular matrix  $L = L_1 \cdots L_n$  and its DAG  $G(L)$ .

Output: A permutation  $Q : V \longrightarrow \{1, \dots, n\}$  and a partition of the permuted matrix  $L_Q$  into factors.

Compute  $level(v)$  for all  $v \in V$ ;

$max\_level \leftarrow \max_{v \in V}(level(v))$ ;

$i \leftarrow 0$ ;     $\{i$  elementary matrices have been included in factors $\}$

$k \leftarrow 1$ ;     $\{P_k$  is the factor being computed $\}$

**while**  $i < n$  **do**

$P_k \leftarrow \emptyset$ ;

$\ell \leftarrow \min\{j \mid \text{there is an unnumbered vertex at level } j\}$ ;

**repeat**

**for every vertex**  $v$  **at level**  $\ell$  **do**

**if** ( $[\text{Condition 1a}]$   $v$  is unnumbered) **and**

$[\text{Condition 1b}]$  Every predecessor of  $v$  has been numbered) **and**

$[\text{Condition 2}]$  Every successor of  $v$  is a successor of all

$u \in P_k$  such that  $u$  is a predecessor of  $v$ ) **then**

$i \leftarrow i + 1$ ;     $Q(v) \leftarrow i$ ;     $P_k \leftarrow P_k \cup \{v\}$ ;

**fi**

**od**

$\ell \leftarrow \ell + 1$ ;

**until**  $\ell > max\_level$  or no vertices at level  $\ell - 1$  were included in  $P_k$ ;

$k \leftarrow k + 1$ ;

**od**

FIG. 3. Algorithm RP1.

---

Algorithm RP1, shown in Fig. 3, rennumbers the elementary matrices during the course of its execution since it computes an appropriate symmetric permutation  $Q$  to minimize the number of factors. Conditions 1a and 1b in the algorithm ensure that the first condition of problem (Pr2) is satisfied; similarly condition 2 ensures that the column subgraphs of the factors are transitively closed.

Alvarado and Schreiber [2] proved that Algorithm RP1 finds a best reordered partition. The time complexity of the algorithm is dominated by the checking of condition 2: in

As noted above, the action of the permutation  $Q$  on  $L$  is to reorder the elementary matrices whose product is  $L$ ; however, these elementary matrices cannot be arbitrarily re-ordered, since we require the resulting matrix  $L_Q$  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 nicely captured in a graph model of (Pr2).

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  $Q$  that leaves  $L_Q$  lower triangular corresponds to a topological reordering of the vertices of  $G(L)$ .

The graph model of (Pr2) is:

(Pr2') Find an ordered partition  $P_1 \prec P_2 \prec \dots \prec P_m$  of the vertices of  $G(L)$  numbered in a topological ordering such that

1. for every  $v \in V$ , if  $v \in P_i$  then all predecessors of  $v$  belong to  $P_1, \dots, P_i$ ,
2. the column subgraph of each  $P_i$  is transitively closed, and
3.  $m$  is minimum subject to these conditions.

The permutation  $Q$  in (Pr2) can be obtained by renumbering the vertices in the ordered partition  $P_1$  to  $P_m$  in increasing order, and in topological order within each subset  $P_i$ .

**2.2. Partitioning algorithms.** We now describe “greedy” algorithms for solving the best no-fill and best reordered partitioning problems.

Input: A unit lower triangular matrix  $L = L_1 L_2 \dots L_n$  and its DAG  $G(L)$ .

Output: A best no-fill partition of  $L$ .

$i \leftarrow 1$ ;     $\{L_i$  is the lowest-numbered elementary matrix not included in a factor yet  $\}$

$k \leftarrow 1$ ;     $\{P_k$  is the factor being computed  $\}$

**while** ( $i \leq n$ ) **do**

$\{\text{Find the largest integer } r \geq i \text{ such that } L_i \dots L_r \text{ is invertible in place}\}$

$r \leftarrow i$ ;

**while**  $r < n$  **and** in  $G(L)$  every successor of the vertex  $r$  is a successor of all predecessors  $v$  of  $r$  such that  $i \leq v < r$  **do**     $r \leftarrow r + 1$ ;    **od**

$P_k \leftarrow \{i, \dots, r\}$ ;     $k \leftarrow k + 1$ ;     $i \leftarrow r + 1$ ;

**od**

FIG. 2. *Algorithm P1.*

**Best no-fill partitions.** Algorithm P1, shown in Fig. 2, was proposed by Alvarado, Yu and Betancourt [3]. This algorithm greedily tries to include as many elementary matrices in the current factor as possible, while maintaining the two properties that a factor should invert in place, and that the ‘left-to-right’ precedence constraint in problem (Pr1) should be obeyed. The condition that in the graph  $G(L)$  every successor of a vertex  $r$  is also a successor of every predecessor of  $r$  ensures that inclusion of the vertex  $r$  in the current factor  $P_k$  will continue to make  $G(P_k)$  transitively closed, and thus  $P_k$  will be invertible in place.

of such edges. Thus the column subgraph of  $P$  is the subgraph induced by the edges that correspond to nonzeros in the column set  $P$ .

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 column subgraph of  $P$  should be transitively closed [8]. (A DAG  $G$  is *transitively closed* if and only 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  $G$ .)

Hence the graph model of (Pr1) is as follows:

- (Pr1') Find an ordered partition  $P_1 \prec P_2 \prec \dots \prec P_m$  of the vertices of  $G(L)$  such that
1. for every  $v \in V$ , if  $v \in P_i$  then all vertices numbered less than  $v$  belong to  $P_1, \dots, P_i$ ,
  2. the column subgraph of each  $P_i$  is transitively closed, and
  3.  $m$  is minimum over all partitions satisfying the given conditions.

The reader will note that we have identified a factor  $P_i$  in (Pr1) with a subset in the vertex partition in its graph model (Pr1'), since there is a one-to-one correspondence between them. We will not distinguish between them, and it should be clear from the context whether the factor or the vertex subset is being discussed.

We illustrate these concepts by means of an example. Consider the matrix  $L$  with graph  $G(L)$  illustrated in Fig. 1. The original ordering of its rows and columns is shown on the left; the corresponding vertex ordering of  $G(L)$  is indicated by the vertex numbers on the left. As shown in the partition of the matrix on the left in Fig. 1,  $L$  has a best no-fill partition with four factors:

$$L = (L_1)(L_2L_3L_4)(L_5 \cdots L_9)(L_{10}L_{11}L_{12}).$$

It is possible to symmetrically permute the rows and columns of  $L$  such that  $L$  remains lower triangular; this corresponds to reordering the elementary matrices while preserving the lower triangular structure of  $L$ . The permutation corresponds to a reordering of the vertices of  $G(L)$ . If in Fig. 1 we reorder the vertices with the numbers shown in the right, then a best no-fill partition of the permuted  $L$  is

$$L = (L_1 \cdots L_6)(L_7 \cdots L_{12}),$$

which has only two factors. The matrix on the right in Fig. 1 corresponds to the reordered matrix, and its partition into two factors is also shown.

A formal statement of the best reordered partitioning problem is as follows:

- (Pr2) Given a unit lower triangular matrix  $L = \prod_{i=1}^n L_i$ , find an admissible permutation  $Q$  and a partition  $L_Q \equiv QLQ^T = \prod_{i=1}^m P_i$ , where
1. each  $P_i = \prod_{k=e_i}^{e_{i+1}-1} L_k$ , with  $e_1 = 1 < e_2 < \dots < e_m < e_{m+1} = n + 1$ ,
  2. each  $P_i$  is invertible in place, and
  3.  $m$  is minimum over all permutations  $Q$  such that  $L_Q$  is lower triangular.

We have taken the opportunity for writing this survey article to provide a unified discussion of the algorithms that have appeared in two different papers, to improve the description of the algorithms (especially Algorithm RP2 in section 2), to illustrate the differences between the algorithms by means of examples, and to correct minor errors.

**2. Two partitioning problems.** We begin by providing formal statements and graph models of the best no-fill and best reordered partitioning problems, and then describe algorithms for computing the partitions when  $L$  is obtained from unsymmetric, symmetric indefinite, or incomplete factorizations.

**2.1. Graph models.** A formal statement of the best no-fill partitioning problem is as follows:

- (Pr1) Given a unit lower triangular matrix  $L = \prod_{i=1}^n L_i$ , find a partition into factors  $L = \prod_{i=1}^m P_i$ , where
1. each  $P_i = \prod_{k=e_i}^{e_{i+1}-1} L_k$ , with  $e_1 = 1 < e_2 < \dots < e_m < e_{m+1} = n + 1$ ,
  2. each  $P_i$  inverts in place, and
  3.  $m$  is minimum over all partitions satisfying the given conditions.

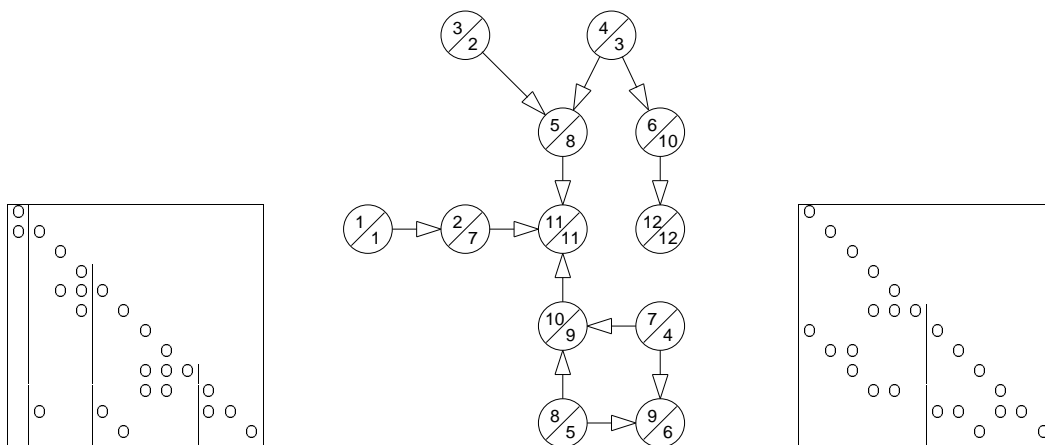


FIG. 1. A lower triangular matrix, its DAG, and its partitions. The original ordering and the partition found by Algorithm P1 are shown on the left, and the ordering and partition found by Algorithms RP1 or RP2 are shown on the right.

It is helpful to consider a graph model of (Pr1) and the other partitioning problems. Let  $G(L)$  denote a directed graph with vertices  $V = \{1, \dots, n\}$  corresponding to the columns of  $L$  and edges  $E = \{(j, i) : i > j \text{ and } l_{ij} \neq 0\}$ . The edge  $(j, i)$  is directed from the lower-numbered vertex  $j$  to the higher-numbered vertex  $i$ . It follows that  $G(L)$  is a directed acyclic graph (DAG). 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 *column subgraph* of  $G(L)$  induced by  $P$  is the graph whose edge set is the subset of edges in  $E$  that are directed from vertices in  $P$  to all vertices in  $V$ , and whose vertex set is the subset of vertices which are the endpoints

To begin we review the partitioned inverse approach to parallel triangular solution. Any unit lower triangular matrix  $L$  can be expressed as a product of elementary matrices:  $L = \prod_{i=1}^n L_i$ , where the elementary matrix  $L_i$  is unit lower triangular and nonzero below the diagonal only in column  $i$ . Hence it has the representation  $L_i = I + \underline{m}_i \underline{e}_i^T$ , where  $\underline{m}_i$  has its first  $i$  components zero, and  $\underline{e}_i$  is the  $i$ -th coordinate vector. (Here it will be convenient to include  $L_n \equiv I$  among the elementary matrices.) The elementary lower triangular matrices can be grouped together to form  $m$  unit lower triangular factors  $L = \prod_{i=1}^m P_i$ , where each factor  $P_i$  is chosen to have the property that  $P_i^{-1}$  can be represented in the same space as  $P_i$ . (Here  $m \leq n$  is a number to be determined.) Each factor  $P_i = \prod_{k=e_i}^{e_{i+1}-1} L_k$ , with  $e_1 \equiv 1 < e_2 < \dots < e_m < e_{m+1} \equiv n + 1$ . The factor  $P_i$  is lower triangular and is zero below its diagonal in all columns except columns  $e_i$  through  $e_{i+1} - 1$ . This leads to a partitioned representation of the inverse of  $L$  of the form  $L^{-1} = \prod_{i=m}^1 P_i^{-1}$  (each  $P_i^{-1}$  is explicitly stored) that can be stored in just the space required for  $L$ .

It follows that the solution to  $L\underline{x} = \underline{b}$  can be computed by means of  $m$  matrix-vector products

$$\underline{x} = L^{-1}\underline{b} = \prod_{i=m}^1 P_i^{-1}\underline{b}.$$

By using as many virtual processors as there are nonzeros in  $P_i$  and summing the products  $\{(P_i^{-1})_{k\ell} b_\ell \mid (P_i^{-1})_{k\ell} \neq 0\}$  in logarithmic time, we may exploit parallelism fully in computing the matrix-vector products.

We consider the problem of computing partitioned inverses with the fewest factors in this paper, since in practice the complexity of highly parallel triangular solution is determined by the number of factors. There are two variations of this problem, and we describe them next after introducing some notation.

A matrix  $X$  is *invertible in place* if and only if  $(X^{-1})_{i,j} = 0$  whenever  $x_{i,j} = 0$ , for any assignment of (nonzero) numerical values to the nonzeros in  $X$ . Since the elementary lower triangular matrices are invertible in place, there is always at least one partition of  $L$  with factors that invert in place. A partition in which the factors  $P_i$  are invertible in place is called a *no-fill partition*. A no-fill partition of  $L$  with the fewest factors is a *best no-fill partition*. An *admissible permutation*  $Q$  of  $L$  is a symmetric permutation of the rows and columns of  $L$  such that the permuted matrix  $QLQ^T$  is lower triangular. A *best reordered partition* of  $L$  is a best no-fill partition of  $QLQ^T$  with the fewest factors over all admissible permutations  $Q$  of  $L$ .

An overview of this survey is as follows. We shall describe efficient algorithms for computing best no-fill and best reordered partitions of lower triangular matrices in section 2. Then we shall show in section 3 that if  $L$  is restricted to be the unit lower triangular matrix from an  $LDL^T$  (Cholesky) factorization, there is an even more efficient algorithm for computing these partitions that makes use of the elimination tree. In section 4 we demonstrate the usefulness of these ideas in practice by comparing the partitioned inverse approach with a conventional triangular solution algorithm on a Connection Machine CM-2. We conclude by summarizing our findings and describing both ongoing and future extensions of this work in section 5.

# HIGHLY PARALLEL SPARSE TRIANGULAR SOLUTION \*

FERNANDO L. ALVARADO<sup>†</sup>, ALEX POTHEN<sup>‡</sup>, AND ROBERT SCHREIBER<sup>§</sup>

**Abstract.** In this paper we survey a recent approach for solving sparse triangular systems of equations on highly parallel computers. This approach employs a partitioned representation of the inverse of the triangular matrix so that the solution can be computed by matrix-vector multiplication. The number of factors in the partitioned inverse is proportional to the number of general communication steps (router steps on a CM-2) required in a highly parallel algorithm. We describe partitioning algorithms that minimize the number of factors in the partitioned inverse over all symmetric permutations of the triangular matrix such that the permuted matrix continues to be triangular. For a Cholesky factor we describe an  $\mathcal{O}(n)$  time and space algorithm to solve the partitioning problem above, where  $n$  is the order of the matrix. Our computational results on a CM-2 demonstrate the potential superiority of the partitioned inverse approach over the conventional substitution algorithm for highly parallel sparse triangular solution. Finally we describe current and future extensions of these results.

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

**Keywords.** chordal graph, directed acyclic graph, elimination tree, graph partitioning, massively parallel computers, partitioned inverse, sparse triangular systems, transitive closure.

**1. Introduction.** We survey some recent developments in the solution of sparse triangular linear systems of equations on a highly parallel computer. For concreteness, we consider a unit lower triangular system  $Lx = \underline{b}$ , but the results in the paper apply in a straightforward manner to upper triangular systems as well. We discuss the situation when there are multiple right-hand side vectors  $\underline{b}$ , and all these vectors are not necessarily available at once. Such situations occur in finite element applications, preconditioned iterative solvers for linear systems, solution of initial value problems by implicit methods, variants of Newton's method for the solution of nonlinear equations, and in numerical optimization.

There are two possible approaches to the parallel solution of triangular systems of equations. One approach is to exploit whatever limited parallelism is available in the usual substitution algorithm [4, 7, 9, 11]. The second approach requires preprocessing, and works with a partitioned representation of  $L^{-1}$ .

---

\* To appear in *Sparse Matrix Computations: Graph Theory Issues and Algorithms*, J. A. George, J. R. Gilbert and J. W. H. Liu, eds, IMA Volumes in Mathematics and its Applications, Springer-Verlag. A part of this work was done while the authors were visiting the Institute for Mathematics and its Applications (IMA) at the University of Minnesota. We thank the IMA for its support.

<sup>†</sup> Electrical and Computer Engineering Department, 1425 Johnson Drive, The University of Wisconsin, Madison, WI 53706 (alvarado@ece.wisc.edu). This author was supported under NSF Contracts ECS-8822654 and ECS-8907391.

<sup>‡</sup> Department of Computer Science, University of Waterloo, Waterloo, Ontario Canada N2L 3G1 (apothen@narnia.uwaterloo.ca, na.pothen@na-net.ornl.gov). This author was supported by NSF grant CCR-9024954 and by U. S. Department of Energy grant DE-FG02-91ER25095 at the Pennsylvania State University and by the Canadian Natural Sciences and Engineering Research Council under grant OGP0008111 at the University of Waterloo.

<sup>§</sup> RIACS, MS T045-1, NASA Ames Research Center, Moffett Field, CA 94035 (schreiber@riacs.edu). This author was supported by the NAS Systems Division under Cooperative Agreement NCC 2-387 between NASA and the University Space Research Association (USRA).