PROCEEDINGS OF THE SIXTH SIAM CONFERENCE ON

PARALLEL PROCESSING FOR SCIENTIFIC COMPUTING

Volume I

Edited by Richard F. Sincovec
Oak Ridge National Laboratory
David E. Keyes
Yale University
Michael R. Leuze
Oak Ridge National Laboratory
Linda R. Petzold
University of Minnesota
Daniel A. Reed
University of Illinois

siam. Philadelphia

Society for Industrial and Applied Mathematics

Parallel Orthogonal Factorizations of Large Sparse Matrices on Distributed-Memory Multiprocessors

Thomas F. Coleman[*] Chunguang Sun[†]

Abstract. We describe the issues involved in the design and implementation of an efficient parallel multifrontal algorithm for computing the QR factorization of a large sparse matrix on distributed-memory multiprocessors. The proposed algorithm has the following novel features. First, a supernodal tree computed from the sparsity structure of R is used to organize the numerical factorization. Second, a new algorithm has been designed for the most crucial task in this context—the QR factorization of two upper trapezoidal matrices in parallel. Third, the overall factorization is accomplished by a sequence of Householder and Givens transformations. Experimental results on an Intel iPSC/860 are included.

1. Introduction. Let A be a large sparse $m \times n (m \geq n)$ matrix with full column rank. The QR factorization of A is expressed as

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where Q is an $m \times m$ orthogonal matrix and R is an $n \times n$ upper triangular matrix. Usually Q is not formed explicitly. It is well known that the upper triangular matrix R is mathematically equivalent to the Cholesky factor of $A^T A$. George and Heath [4] propose a row merging scheme for reducing A to R by a sequence of Givens rotations. Liu [7] generalizes the George and Heath scheme into a general row merging scheme which leads to substantial reduction in arithmetic operations.

2. A multifrontal sparse QR factorization algorithm. The basic idea of multifrontal sparse QR factorization is described in [7]. The overall sparse QR factorization is accomplished by a sequence of upper trapezoidal submatrix merges organized around a row merge tree. A supernodal tree computed from the structure of R is used as our row merge tree which is slightly different from the row merge tree described in [7]. A multifrontal algorithm for sparse QR factorization is described below.

[*] Computer Science Department and Advanced Computing Research Institute, Cornell University, Ithaca, NY 14853 (coleman@cs.cornell.edu).

[†] Advanced Computing Research Institute, Cornell Theory Center, Cornell University, Ithaca, NY 14853-3801 (csun@cs.cornell.edu).

1. find a column ordering for A such that R is sparse.
2. compute the elimination tree of R^T by using the algorithm described in [6] and number the nodes of the elimination tree in postorder.
3. determine the symbolic structure of R.
4. compute the supernodal tree of R^T from the corresponding elimination tree and number the supernodes in postorder.
5. perform numerical factorization by processing the supernodes in order.

Let R_{i*} denote row i of R and $Stru(R_{i*})$ the row structure of R_i—i.e., the set of nonzero column indices of R_i in increasing order. The nodes in a supernode are numbered in increasing order. Let K be a supernode. $|K|$ denotes number of nodes in K and $First(K)$ the first node in K. The upper trapezoidal matrix associated with K is referred to as the *frontal matrix* of K and is denoted by F_K. Let m_K and n_K be the number of rows and the number of columns of F_K, respectively. Then $m_K \le n_K = |Stru(R_i)|$, where $i = First(K)$. For supernodes close to the root, their frontal matrices are usually upper triangular. The upper trapezoidal matrix obtained from F_K by deleting the first $|K|$ rows of F_K is called the *update matrix* of K and is denoted by U_K. U_K participates in the formation of the frontal matrix of its parent. The computation associated with a supernode is described in Fig. 1, where A_i denotes the submatrix consisting of all rows from A whose first nonzeros are in column i. Reduction of A_j to an upper trapezoidal matrix R_j may be accomplished by either Householder transformations or Givens rotations. The submatrix merges in line 3 and 6 are done by sparse Givens rotations.

```
1  allocate space for $F_K$;
2  assemble $U_C$ into $F_K$ and deallocate $F_C$, where $C$ is a child of $K$;
3  for each remaining child $C$ of $K$ do merge $U_C$ into $F_K$;
4  for $j = First(K)$ to $First(K) + |K| - 1$ do
5      reduce $A_j$ to an upper trapezoidal matrix $R_j$ by orthogonal transformation;
6      merge $R_j$ into $F_K$;
7      row $j - First(K) + 1$ of $F_K$ is the computed row $j$ of $R$;
8  end for
```

FIG. 1. *A sequential algorithm for forming the frontal matrix of a supernode K from the update matrices of its children and corresponding rows from the original matrix A*

3. A parallel multifrontal sparse QR factorization algorithm.

Several parallel algorithms for the numeric phase of the sparse QR factorization on distributed-memory multiprocessors have been described in the literature [1, 8]. We propose a new distributed algorithm for the numeric phase of the multifrontal sparse QR factorization described in Section 2. The supernodal tree is mapped onto the multiprocessors by a proportional mapping algorithm [9]. The root of the supernodal tree is partitioned among all processors. If a supernode has already been mapped to a set of processors, each subtree rooted at a child of that supernode is allocated a subset of processors whose size is proportional to the workload associated with that subtree. Initially each processor is working on its own subtrees. Later on processors cooperate to compute the frontal matrix of a partitioned supernode. When a supernode is partitioned among a set of q processors $\{s, s+1, \cdots, s+q-1\}$, the rows of F_K

are partitioned into q blocks of approximately equal number of rows. Block b is assigned to processor $s+b-1$ for $1 \le b \le q$. The processor $s+q-1$ having the last block of F_K is denoted by $LP(F_K)$.

Parallel algorithms are needed for assembling the update matrix of a child of K into F_K, merging the update matrix of a child of K into F_K and rotating relevant rows from A into F_K. Because of space limitation, we omit the descriptions of the parallel algorithms for assembling an update matrix into a frontal matrix and rotating rows from A into a frontal matrix. We briefly describe the parallel algorithm for merging an update matrix into a frontal matrix. Assume that $C = \{16, 17, 18\}$ is a child of $K = \{19, 20, 21\}$ in the supernodal tree. Assume that $Stru(R_{16,*}) = \{16, 17, 18, 19, 20, 21, 46, 47, 48, 49\}$ and $Stru(R_{19,*}) = \{19, 20, 21, 43, 44, 45, 46, 47, 48, 49\}$. Let $m_K = 10$ and $m_C = 10$. F_C is partitioned into two blocks and F_K is partitioned into five blocks as illustrated in Fig. 2. The first block of F_C consisting of the first five rows of F_C is assigned to processor p_0 and the second block of F_C consisting of the last five rows of F_C is assigned to processor p_1. The block i of F_K consisting of rows $2i-1$ and $2i$ of F_K is assigned to processor p_{i-1} for $1 \le i \le 5$. The first three rows of F_C are the computed rows 16, 17 and 18 of R after F_C is formed.

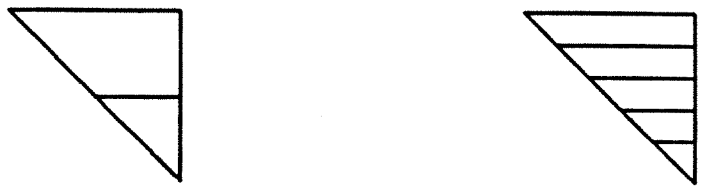

FIG. 2. *Two partitioned frontal matrices F_C(left) and F_K(right)*

Let B denote the second block of F_C. B is partitioned into three segments since there are three processors p_1, p_3 and p_4 to which the rows in B should be initially sent. These three processors are referred to as the *target processors* of B. The first segment consisting of the first row of B is rotated into the second, third, fourth and fifth block of F_K, successively. The second segment consisting of row 2 and row 3 of B is rotated into fourth and fifth block of F_K, successively. The third segment consisting of row 4 and row 5 of B is rotated into fifth block of F_K. The distributed submatrix merging algorithm is described in Fig. 3.

4. Experimental results. Our algorithm has been tested on a 32-node iPSC/860 for a set of problems with regular and irregular sparsity structure. The problems with regular sparsity structure include two-dimensional(2D) $k \times k$ nine-point grid and three-dimensional(3D) $k \times k \times k$ twenty-seven point grid. The matrix associated with a 2D grid is constructed as follows. There is a column associated with each vertex of the grid and a row with each square element. The row has nonzeros for each of the four vertices that define the element. This row is repeated r times to obtain a $(k-1)^2r$ by k^2 sparse matrix. A matrix associated with a 3D grid can be similarly constructed. The grid problems are ordered by the nested dissection ordering [3]. The problems with irregular sparsity structure are generated randomly. The number of nonzero entries in a row is limited. However, the locations of the nonzero entries in a row are randomly distributed. Random problems are ordered by the minimum degree ordering [5].

The experimental results are shown in Table 1 and Table 2, where m denotes number of rows, n number of columns, r number repetitions of a row corresponding to a square element, $|A|$ number of nonzeros in matrix A, $|R|$ number of nonzeros in factor R, *s_node* number of supernodes, "fac_time" numerical factorization time, "MFLOPS" number of mega flops performed per second during numerical factorization, and k maximum number of nonzero

```
$num\_msg = 0$;
for each block $B$ of $U_C$ do
    let $s$ be the processor to which $B$ is mapped;
    for each seqment $S$ of $B$ do
        let $t$ be the target processor of $S$;
        if $p = s$ then
            if $p = t$ then
                rotate $S$ into the block of $F_K$ mapped to $p$;
                if $p \neq LP(F_K)$ then send the resulting block to $p + 1$;
            else
                send $S$ to $t$;
            end if
        else
            if $p = t$ then
                receive $S$ and rotate it into the block of $F_K$ mapped to $p$;
                if $p \neq LP(F_K)$ then send the resulting block to $p + 1$;
            end if
            if $s < p$ and $p <= LP(F_K)$ then $num\_msg = num\_msg + 1$;
        end if
    end for
end for

while ($num\_msg > 0$) do
    receive a block from $p - 1$ and rotate it into the block of $F_K$ mapped to $p$;
    $num\_msg = num\_msg - 1$;
    if $p \neq LP(F_K)$ then send the resulting block to $p + 1$;
end while
```

FIG. 3. *Distributed merge of an update matrix U_C into a frontal matrix F_K on processor p*

entries per row for a random problem. A flop is either a multiplicative operation or an additive operation. The notation k-2D represents a $k \times k$ nine-point grid while k-3D represents a $k \times k \times k$ twenty-seven point grid. The reduction of A_j to an upper trapezoidal matrix R_j is done by Householder transformations.

5. Concluding remarks. We have described an efficient algorithm for computing the sparse QR factorization of a large sparse matrix on distributed-memory multiprocessors. The distributed sparse submatrix merge is accomplished by a block approach on a chain of processors. This approach tries to reduce the communication cost and avoids the potential danger of communication deadlock on a ring of processors such as the algorithm described in [1]. No practical performance results are given in [1]. Our approach also avoids the disadvantage of the algorithm described in [8] where the amount of arithmetic work increases as number of processors increases. Experiments on problems with irregular sparsity structure

TABLE 1
sparse QR factorizations for grid problems on an iPSC/860 with 32 nodes.

problems	m	n	r	$\|A\|$	$\|R\|$	*s_node*	fac_time	MFLOPS
127-2D	158,760	16,129	10	635,040	518,578	8,191	3.740	40.44
127-2D	793,800	16,129	50	3,175,200	518,578	8,191	4.220	66.49
24-3D	60,835	13,824	5	486,680	2,806,944	4,824	92.529	42.98

TABLE 2
sparse QR factorizations for random problems on an iPSC/860 with 32 nodes.

m	n	k	$\|A\|$	$\|R\|$	*s_node*	fac_time	MFLOPS
10,000	1,000	2	19,993	252,117	330	76.015	62.53
10,000	1,000	3	29,971	381,405	158	155.841	71.39
10,000	2,000	2	19,997	555,185	1,034	128.166	60.81
10,000	3,000	2	19,998	746,525	1,892	147.334	55.91

from practical applications will be conducted. A comprehensive description of our algorithm and experimental results will be given in [2].

Acknowledgements. This research was partially supported by the Cornell Theory Center, which receives major funding from the National Science Foundation and the IBM corporation, with additional support from New York State and members of its Corporate Research Institute. Numerical experiments were performed on a 32-node Intel iPSC/860 in the Computer Science Department, Cornell University.

REFERENCES

[1] E. CHU AND A. GEORGE, *Sparse orthogonal decomposition on a hypercube multiprocessor*, SIAM J. Mat. Anal. Appl., 11 (1990), pp. 453–465.
[2] T. F. COLEMAN AND C. SUN, *Parallel orthogonal factorizations of large sparse matrices on distributed-memory multiprocessors.* Work in preparation, 1992.
[3] J. A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.
[4] J. A. GEORGE AND M. T. HEATH, *Solution of sparse linear least squares problems using Givens rotations*, Linear Algebra and its Appl., 34 (1980), pp. 69–83.
[5] J. A. GEORGE AND J. W. H. LIU, *The evolution of the minimum degree algorithm*, SIAM Review, 31 (1989), pp. 1–19.
[6] J. W. H. LIU, *A compact row storage scheme for Cholesky factors using elimination trees*, ACM Trans. on Math. Software, 12 (1986), pp. 127–148.
[7] ——, *On general row merging schemes for sparse Givens transformations*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 1190–1211.
[8] P. E. PLASSMANN, *Sparse Jacobian estimation and factorization on a multiprocessor*, in Large-Scale Numerical Optimization, T. F. Coleman and Y. Li, eds., SIAM, Philadelphia, 1990, pp. 152–179.
[9] A. POTHEN AND C. SUN, *A distributed multifrontal algorithm using clique trees*, Tech. Report 91-24, Computer Science, Pennsylvania State University, University Park, PA, 1991.