This paper appears in
Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing, D. H. Bailey, P. E. Bjørstad, J. R. Gilbert, M. V. Mascagni, R. S. Schreiber, H. D. Simon, V. J. Torczon, and L. T. Watson, eds., SIAM, Philadelphia, 1995, pp.418-423.

Parallel multifrontal solution of sparse linear least squares problems on distributed-memory multiprocessors[1]

Chunguang Sun
Advanced Computing Research Institute
Cornell Theory Center
Cornell University
Ithaca, NY 14853-3801
csun@cs.cornell.edu

Abstract

We describe the issues involved in the design and implementation of efficient parallel algorithms for solving sparse linear least squares problems on distributed-memory multiprocessors. We consider both the QR factorization method and the method of corrected semi-normal equations. The sparse QR factorization is accomplished by a parallel multifrontal scheme recently introduced. A new parallel algorithm for solving the related sparse triangular systems is proposed. Experimental results on an Intel iPSC/860 machine are described.

[1]This work was supported in part by the Cornell Theory Center which receives funding from members of its Corporate Research Institute, the National Science Foundation (NSF), the Advanced Research Projects Agency (ARPA), the National Institutes of Health (NIH), New York State, and IBM corporation.

Parallel Multifrontal Solution of Sparse Linear Least Squares Problems on Distributed-Memory Multiprocessors †

Chunguang Sun ‡

Abstract

We describe the issues involved in the design and implementation of efficient parallel algorithms for solving sparse linear least squares problems on distributed-memory multiprocessors. We consider both the QR factorization method and the method of corrected semi-normal equations. The sparse QR factorization is accomplished by a parallel multifrontal scheme recently introduced. A new parallel algorithm for solving the related sparse triangular systems is proposed. Experimental results on an Intel iPSC/860 machine are described.

1 Introduction

Consider the numerical solution of large and sparse linear least squares problems

$$\min_x \|Ax - b\|_2, \tag{1}$$

where A is an $M \times N$ matrix with full column rank. The QR factorization method [2] for solving (1) is to reduce A to an upper triangular matrix R by an orthogonal matrix Q

$$Q^t A = \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{2}$$

and form $Q^t b$. The solution is then obtained by solving the triangular system $Rx = c$, where c is the first N components of $Q^t b$. This method is numerically backward stable. However, the orthogonal matrix Q needs to be retained if right hand sides are not yet available when A is factorized. If A is large and sparse, it is often too expensive to store Q in the main memory.

Once R is computed, the semi-normal equations $R^t R x = A^t b$ can be used to handle new right hand sides without storing Q. Björck [1] shows that the method of semi-normal equations(SNE) is numerically instable and proposes that an iterative refinement step be added to the SNE method as follows.

$$\begin{cases} r \leftarrow b - Ax, \\ R^t R \delta x = A^t r, \\ x \leftarrow x + \delta x. \end{cases} \tag{3}$$

Björck [1] shows that this new method — the method of corrected semi-normal equations(CSNE) is, in general, as accurate as the QR factorization method. However, it is not always backward stable and may not be accurate for "stiff" problems.

†This work was supported in part by the Cornell Theory Center which receives funding from members of its Corporate Research Institute, the National Science Foundation (NSF), the Advanced Research Projects Agency (ARPA), the National Institutes of Health (NIH), New York State, and IBM corporation.

‡Advanced Computing Research Institute, Cornell Theory Center, Cornell University, Ithaca, NY 14853.

In this paper we present new parallel algorithms based on the QR factorization method, the SNE method and the CSNE method for solving sparse linear least squares problems on distributed-memory multiprocessors. The major tasks involved are sparse QR factorization and sparse triangular solution. The sparse QR factorization is accomplished by a parallel multifrontal scheme described in [4]. A new parallel multifrontal algorithm for solving the related sparse triangular systems is proposed. Numerical experiments on an Intel iPSC/860 machine are described.

2 Parallel multifrontal sparse QR factorization

The first step in a direct method for solving a sparse linear least squares problem is to compute a sparse QR factorization. The parallel algorithm proposed in [4] is used to perform parallel sparse QR factorization. The supernodal structure of R is fully exploited in the numeric factorization. Let $S = \{i, i+1, \cdots, j\}$ be a supernode consisting of rows $i, i+1, \cdots, j$ of R. During numerical factorization, the supernode S is associated with an upper trapezoidal matrix F called the *frontal matrix* of S. Without loss of generality, we assume that F is a dense $n \times n$ upper triangular matrix, where n is the number of nonzeros in row i of R. The first $|S| = j - i + 1$ rows of F are referred to as the *factor rows* of S. The matrix formed by the factor rows of S is referred to as the factor matrix of S.

A sparse QR factorization is decomposed into a set of computational tasks according to the supernodal tree structure. A task is to form the frontal matrix of a supernode. The computational tasks are mapped onto node processors by the proportional mapping scheme described in [3]. If a task associated with a supernode S is partitioned among a set of p processors, then the frontal matrix F of the supernode S is partitioned among the set of processors by the equal-volume partitioning strategy introduced in [4]. In other words, the rows of F are partitioned into p blocks such that every block consists of a set of adjacent rows of F and contains approximately $|F|/p$ numerical values of F. The storage scheme for R used in the numerical factorization is also used in the sparse triangular solution.

3 Parallel multifrontal sparse triangular solution

In this section we describe a parallel multifrontal algorithm for solving $R^t Rx = c$, a crucial step for the SNE or CSNE method. A sparse forward elimination or back substitution is decomposed into a sequence of solutions of dense trapezoidal systems. The overall sparse triangular solution is accomplished by processing all dense trapezoidal systems in a multifrontal framework. The advantages of the multifrontal method such as data locality and dense matrix computations are effectively exploited in the sparse triangular solution.

Let S denote a supernode consisting of rows $i_1, i_2, \cdots, i_m$ of R, where $i_{j+1} = i_j + 1$ for $1 \leq j < m$. Let $S_a = \{i_1, \cdots, i_m, i_{m+1}, \cdots, i_n\}$ denote the nonzero column indices of row i_1 of R in increasing order. Let $S_f = \{i_1, \cdots, i_m\}$ and $S_u = \{i_{m+1}, \cdots, i_n\}$. The variables associated with S_f and S_u are referred to as the *factor variables* and the *update variables* of S, respectively.

An *index sequence* is defined as a subset of $\mathcal{N} = \{1, 2, \cdots, N\}$ in increasing order. Obviously, S_a, S_f and S_u defined above are index sequences. A sparse vector is represented as a dense vector and an index sequence. While the dense vector contains the numerical values of the non-zero entries of the sparse vector, the index sequence contains the indices of the corresponding non-zero entries. We use $[val, ind]$ to denote a sparse vector, where val is the dense vector and ind is the corresponding index sequence. Let $I = \{i_1, \cdots, i_\alpha\}$ and $J = \{j_1, \cdots, j_\beta\}$ be index sequences. We assume that I is a subset of J. The *scatter* of a sparse vector $[v, I]$ to another sparse vector $[w, J]$ is defined as

$$[w, J] \leftarrow [w, J] \oplus [v, I], \tag{4}$$

where $w_h = w_h + v_k$ if $i_k = j_h$ for $1 \leq k \leq \alpha$. The *gather* of a sparse vector $[w, J]$ to another sparse vector $[v, I]$ is defined as

$$[v, I] \leftarrow [v, I] \lhd [w, J], \tag{5}$$

where $v_k = w_h$ if $i_k = j_h$ for $1 \leq k \leq \alpha$.

Sparse forward elimination. Assume that a supernode S is defined as above. The lower trapezoidal system associated with S is written as $T^t z = d$, where T is the factor matrix of S, z is a vector of length m to be determined, and d is a vector of length n. Partition T and d as

$$T = (T_f \;\; T_u) \;\text{ and }\; d = \begin{pmatrix} d_f \\ d_u \end{pmatrix}, \tag{6}$$

where T_f is an $m \times m$ upper triangular matrix and d_f contains the first m components of vector d. The vector d_f is initialized with values of the right hand side vector c—i.e., $d_j = c_{i_j}$ for $1 \leq j \leq m$. The vector d_u is initialized to zero. If S is a leaf in the supernode tree, the system $T^t z = d$ is solved as follows.

$$T_f^t z = d_f \text{ and } d_u = d_u - T_u^t z. \tag{7}$$

The resulting vector d_u represents the effects on the update variables of S by solving the factor variables of S. The vector d_u is called the *effect vector* of S and is denoted by $e(S)$. If S is an interior supernode in the supernode tree, the vector d needs to be updated by the effect vectors from all children of S. Let C be a child of S. The update on the vector d by $e(C)$, the effect vector of C, can be described by the *scatter* operation defined in (4):

$$[d, S_a] \leftarrow [d, S_a] \oplus [e(C), C_u]. \tag{8}$$

After d has been updated by effect vectors from all children of S, the associated trapezoidal system $T^t z = d$ is solved as in (7). The computed effect vector $e(S) = d_u$ represents the accumulated effects on the update variables of S by solving all factor variables in the subtree rooted at supernode S.

Let $proc(S)$ be the set of processors among which the frontal matrix of S is partitioned. Let $scatter(S)$ denote the processor in $proc(S)$ for scattering the effect vectors of the children of S into the right hand side vector of the lower trapezoidal system associated with S. Let $store\_e(S)$ denote the processor in $proc(S)$ for storing $e(S)$ after the lower trapezoidal system associated with S is solved. The parallel multifrontal sparse forward elimination algorithm is presented in Fig. 1, where $parent(S)$ is the parent of S.

Sparse back substitution. The upper trapezoidal system associated with the supernode S is described as $Ty = z$, where T is the factor matrix of S, y is a vector of length n, and z, a vector of length m, is the solution vector to the lower trapezoidal system of S computed in the sparse forward elimination phase.

Let $store\_s(S)$ denote the processor in $proc(S)$ for storing the solution vector to the upper trapezoidal system of S. Let $gather(S)$ denote the processor in $proc(S)$ for gathering solution components corresponding to the update variables of the supernode S from the solution vector to the upper trapezoidal system of the parent of S. The parallel multifrontal sparse back substitution algorithm is described in Fig. 2.

for each supernode S in a postorder **do**
    $[d_f, S_f] \leftarrow [d_f, S_f] \lhd [c, \mathcal{N}]$, $d_u = 0$;
    if $proc(S) = \{\mu\}$ **then**
        for each child C of S **do** $[d, S_a] \leftarrow [d, S_a] \oplus [e(C), C_u]$;
        solve $T^t z = d$;
        if S is not a root and $\mu \neq scatter(parent(S))$ **then**
            send $e(S)$ to $scatter(parent(S))$;
    else if $\mu \in proc(S)$ **then**
        if $\mu = scatter(S)$ **then**
            for each child C of S **do**
                if $\mu \neq store\_e(C)$ **then** receive $e(C)$;
                $[d, S_a] \leftarrow [d, S_a] \oplus [e(C), C_u]$;
            end for
        end if
        solve $T^t z = d$ in parallel;
        if $\mu = store\_e(S)$ and S is not a root and $\mu \neq scatter(parent(S))$ **then**
            send $e(S)$ to $scatter(parent(S))$;
    end if
end for

FIG. 1. *A parallel multifrontal sparse forward elimination algorithm on a processor μ.*

4 Experimental results

Our test problems include two-dimensional and three-dimensional grid problems shown in Table 1, where M is the number of rows, N the number of columns, $|A|$ the number of nonzeros in matrix A, $|R|$ the number of nonzeros in factor R, "GRIDk" represents a $k \times k$ grid. "GRIDk.3" represents a three-dimensional $k \times k \times k$ grid. All programs are written in C and double-precision floating-point arithmetic is used. The matrix A is distributed among processors by rows. Each processor computes a local portion of the product $A^T b$ or $A^T(b - Ax)$. Then all processors cooperate to obtain the product $A^T b$ or $A^T(b - Ax)$.

TABLE 1
Test problems

problem	M	N	$\|A\|$	$\|R\|$
GRID200	158404	40000	633616	1481803
GRID250	248004	62500	992016	2459325
GRID24.3	97336	13824	778688	2806944
GRID28.3	157464	21952	1259712	5445708

In Table 2, "qr_time", "fe_time", "bs_time", "mv_time" are the running times for numerical factorization, forward elimination, back substitution, and the local portion of the matrix-vector product $A^t(b - Ax)$, respectively. The "gs_time" is the time spent on global summation for computing the matrix-vector product $A^t(b - Ax)$. A running time is obtained by measuring the time spent on each processor and taking the maximum time spent on a processor.

for each supernode S in the reverse order of a postorder **do**
    if $proc(S) = \{\mu\}$ **then**
        if S is not a root **then**
            if $\mu \neq store\_s(J)$ **then** receive y_J, where $J = parent(S)$;
            $[y_u, S_u] \leftarrow [y_u, S_u] \lhd [y_J, J_a]$;
        end if
        solve $Ty = z$;
    else if $\mu \in proc(S)$ **then**
        if $\mu = gather(S)$ **then**
            if S is not a root **then**
                if $\mu \neq store\_s(J)$ **then** receive y_J, where $J = parent(S)$;
                $[y_u, S_u] \leftarrow [y_u, S_u] \lhd [y_J, J_a]$;
            end if
        end if
        solve $Ty = z$ in parallel;
        if $\mu = store\_s(S)$ **then**
            for each child C of S **do**
                if $\mu \neq gather(C)$ **then** send y to $gather(C)$;
        end if
    end if
end for

FIG. 2. *A parallel multifrontal sparse back substitution algorithm on a processor μ.*

TABLE 2
Running times of our parallel algorithms on a 32-node iPSC/860 in seconds

problem	p	qr_time	fe_time	bs_time	mv_time	gs_time
	8	13.896	0.282	0.243	0.089	0.670
GRID200	16	8.313	0.189	0.147	0.046	1.056
	32	5.112	0.155	0.106	0.025	1.195
GRID250	32	8.622	0.223	0.148	0.040	1.965
GRID24.3	16	49.109	0.301	0.146	0.048	0.390
	32	30.026	0.309	0.120	0.024	0.480
GRID28.3	32	68.832	0.497	0.181	0.040	0.721

The "mflops" in Table 3 is the execution rate defined as the number of megaflops performed per second during numerical factorization. The "QR" and "SNE" in Table 3 represent the running times for the QR factorization method and the SNE method, respectively. The "C_STEP" shows the running time for a correction step in the CSNE method. The time for the QR factorization method includes "qr_time", the time for forming $Q^t b$ and "bs_time". The time for the SNE method includes "qr_time", "fe_time", "bs_time", "mv_time" and "gs_time". The running time for a correction step is the sum of "fe_time", "bs_time", "mv_time" and "gs_time".

More experimental results are provided in [5]. The arithmetic and communication complexities of our algorithms on regular grid problems are derived and described in [5].

TABLE 3

Performance of the QR factorization, SNE and CSNE methods on a 32-node iPSC/860

problem	p	mflops	QR	SNE	C_STEP
GRID200	8	36.059	14.524	15.180	1.284
	16	60.276	8.664	9.751	1.438
	32	98.020	5.259	6.593	1.481
GRID250	32	110.950	8.878	10.998	2.376
GRID24.3	16	82.604	49.560	49.994	0.885
	32	135.103	30.284	30.959	0.933
GRID28.3	32	144.114	69.249	70.271	1.439

5 Concluding remarks

We have described three parallel approaches based on the QR factorization method, the SNE method and the CSNE method for solving sparse linear least squares problems on distributed-memory multiprocessors. The parallel sparse QR factorization has been discussed in [4]. Central to our approaches is a new parallel multifrontal algorithm for solving the related sparse triangular systems. The choice among the three approaches depends on the availability of the right hand sides, the size of the primary storage space and the desired accuracy for the computed solutions.

We are currently experimenting a generalized scheme of our equal-volume partitioning strategy proposed in [4]. Assume that a frontal matrix F is mapped to p processors. Our generalized scheme partitions F into $k \geq p$ blocks such that every block consists of a set of adjacent rows of F and contains approximately $|F|/k$ numerical values of F. Our preliminary results show that the generalized scheme significantly improves the performance of our parallel sparse QR factorization algorithm.

References

[1] A. BJÖRCK, *Stability analysis of the method of seminormal equations for linear least squares problems*, Linear Algebra Appl., 88/89 (1987), pp. 31–48.

[2] G. GOLUB, *Numerical methods for solving linear least squares problems*, Numer. Math., 7 (1965), pp. 206–216.

[3] A. POTHEN AND C. SUN, *A mapping algorithm for parallel sparse Cholesky factorization*, SIAM J. Sci. Comput., 14 (1993), pp. 1253–1257.

[4] C. SUN, *Parallel sparse orthogonal factorization on distributed-memory multiprocessors*, Tech. Report CTC93TR162, Advanced Computing Research Institute, Center for Theory and Simulation in Science and Engineering, Cornell University, Ithaca, NY, Dec. 1993.

[5] ———, *Parallel multifrontal solution of sparse linear least squares problems on distributed-memory multiprocessors*, Tech. Report CTC94TR185, Advanced Computing Research Institute, Center for Theory and Simulation in Science and Engineering, Cornell University, Ithaca, NY, July 1994.