LA-SUB--95-161

PARALLEL COMPUTATION OF LARGE LEAST SQUARES PROBLEMS INVOLVING KRONECKER PRODUCTS ON THE CONNECTION MACHINE 5

Charles T. Fulton and Limin Wu
Department of Applied Mathematics
Florida Institute of Technology
Melbourne, Florida 32901

July 1995

Abstract

We present in this paper some timing results for a Data Parallel Version of a Kronecker Product Least Squares Code on the Connection Machine 5

This research was supported under Contract No. 1030N0014-9H with Los Alamos National Laboratory.

1 Introduction

In this paper we describe the implementation of an algorithm for computing the solution of the Kronecker Product least squares problem

$$(A \otimes B)x = t \tag{1}$$

or, equivalently,

$$(B \cdot X \cdot A^T) = T, \tag{2}$$

with x = vec(X), and t = vec(T), in the $full - rank$ case in Data Parallel CMFortran on the Connection Machine 5. Here A and B are real matrices of dimensions $m \times p$, $m > p$, and $n \times q$, $n > q$, with rank(A) = p and rank(B) = q. Also, X is $q \times p$, T is $n \times m$, and vec(X) denotes the vector consisting of the stacked columns of the matrix X. The algorithm is essentially that given for the i860 Intel in [2], although a few modifications were required in the porting to CMFortran, and in the adaptation to the CMSSL (Connection Machine Scientific Software Library) library routines.

We recall briefly the algorithm. Let the matrices A and B be decomposed by QR factorizations with column pivoting so that

$$A \cdot P_A = Q_A \cdot \left\{ \begin{matrix} R_A \\ 0 \end{matrix} \right\} \tag{3}$$

and

$$B \cdot P_B = Q_B \cdot \left\{ \begin{matrix} R_B \\ 0 \end{matrix} \right\} \tag{4}$$

where Q_A and Q_B are $m \times m$ and $n \times n$ real orthogonal matrices respectively, R_A and R_B are $p \times p$ and $q \times q$ real upper triangular matrices respectively, and P_A and P_B are $p \times p$ and $q \times q$ permutation matrices respectively.

Letting

$$Q_A = (Q_{A1}, Q_{A2}), \qquad Q_B = (Q_{B1}, Q_{B2}) \tag{5}$$

where Q_{A1} is the matrix of the first p columns of Q_A and Q_{B1} is the matrix of the first q columns of Q_B, and putting

$$Y = P_B^T \cdot X \cdot P_A \tag{6}$$

the least squares problem (2) may be written in the equivalent form

$$\left\{ \begin{matrix} R_B \cdot Y \cdot R_A^T & 0^{(2)} \\ 0^{(1)} & 0^{(3)} \end{matrix} \right\} \tag{7}$$

$$= \left\{ \begin{matrix} Q_{B1}^T \cdot T \cdot Q_{A1} & Q_{B1}^T \cdot T \cdot Q_{A2} \\ Q_{B2}^T \cdot T \cdot Q_{A1} & Q_{B2}^T \cdot T \cdot Q_{A2} \end{matrix} \right\} \tag{8}$$

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

where $0^{(1)}$, $0^{(2)}$, and $0^{(3)}$ are zero matrices of order $n-q \times p$, $q \times m-p$, and $n-q \times m-p$ respectively. It follows that the least squares solution of (8) or (2) is the exact solution of the nonsingular system

$$R_B \cdot Y \cdot R_A^T = Q_{B1}^T \cdot T \cdot Q_{A1} \tag{9}$$

or, equivalently,

$$(R_A \otimes R_B)vec(Y) = vec(Q_{B1}^T \cdot T \cdot Q_{A1}). \tag{10}$$

With

$$H = Q_{B1}^T \cdot T \cdot Q_{A1} \tag{11}$$

we have the two step procedure for computing Y from H: Let

$$Z = R_B \cdot Y \tag{12}$$

and write $Z \cdot R_A^T = H$ in transposed form as

$$R_A \cdot Z^T = H^T. \tag{13}$$

The backsolves indicated in equations (13) and (12) are perfectly parallel since they can be performed independently to generate the columns of Z^T, Y from the columns of the 'right hand sides' H^T, Z respectively. The basic algorithm is therefore as follows:

Step 1: Compute the QR factorization of A.

Step 2: Compute the QR factorization of B.

Step 3: Form the right hand side vectors for the backsolves in equation (13) by computing the matrix product

$$H^T = Q_{A1}^T \cdot T^T \cdot Q_{B1}. \tag{14}$$

Step 4: Perform the backsolves in equation (13) by distributing the columns of H^T 'equally' across the processors.

Step 5: Compute the transpose of Z^T to get the right hand side vectors Z for equation (12).

Step 6: Perform the backsolves in equation (12) by distributing the columns of Z 'equally' across the processors.

Step 7: Compute the least squares solution in matrix form using equation (6),viz.

$$X = P_B \cdot Y \cdot P_A^T. \tag{15}$$

Step 8: Finally, the residual is computed from (2) as the Frobenius norm of

$$T - (B \cdot X \cdot A^T) \tag{16}$$

2 Implementation of Least Squares Algorithm in CMFortran

The three main paradigms currently available for parallel programming on High Performance Computers are (i) shared-memory (ii) explicit message-passing and (iii) data parallel. On a given machine these paradigms may exist either in hardware or software, or a combination. On the Connection Machine 5 the data parallel and message-passing paradigms are available, but the shared- memory paradigm is not. The above described algorithm was implemented on the Connection Machine 5 using CMFortran in the (global) data parallel paradigm.

All steps of the algorithm were implemented using standard CMSSL (Connection Machine Scientific Software Library) routines from [3], except for the backsolves in steps 4 and 6, which were coded explicitly in data parallel CMFortran. The reason for this is that in the CMSSL Library (see [3], Chap. 5) the *gen−lu−factor* and *gen−lu−solve* routines are organized as a coupled set for performing Gaussian elimination, and one cannot make use of the backsolver without first performing the LU decomposition, even for an upper triangular input matrix. At this writing we have been advised that Thinking Machines Corporation does not have plans to produce a stand-alone backsolver for upper triangular matrices in future releases of the CMSSL Library.

Each step of the above algorithm was timed separately. Here we describe which CMSSL routines were used for these timings. For steps 1 and 2 the QR-factorizations in (3) and (4) were computed using CMSSL routine $gen-qr-factor$; the times reported include the times to recover the Q, R and P-matrices in full storage mode using the CMSSL routines $gen-qr-apply-q$, $gen-qr-get-r$, and $gen-qr-apply-p$. For step 3 the right hand side matrix H was computed using (11) and CMSSL routines *matmul* abd *transpose*. For step 5 the transpose of Z^T was computed using CMSSL routine *transpose*. For step 7 the solution matrix was computed using (15) and the *matmul* and *transpose* routines. Here the permutation matrices P_A and P_B were used in full storage mode. The reason for this inefficient computation using permutation matrices is that the CMSSL Library routines associated with the QR-factorization make no provision for direct recovery of the pivot vector associated with the column pivoting in the QR-factorization of the input matrix. It seems that this is due to the fact that the QR-factorizations are performed on a block cyclic permutation of the input matrix, so that the pivot vector associated with the input matrix itself never gets generated. For step 8 the residual is computed using (16) and the Frobenius norm using CMSSL routines *matmul*, *transpose* and *sum*.

The A and B matrices (and the right hand side matrix T yielding known solution X) were generated in parallel on the CM-5 using the random number generator RNG, but this preliminary step was not timed.

In the Data Parallel paradigm the smallest partition size which can be used on the CM5 at Los Alamos National Laboratory is 32. Accordingly, with 4 VUs/node, a 32-node partition functions as a SIMD machine with 128 processing elements all operating in parallel.

3 Backsolve Coding

We describe here the manner in which the backsolve coding for steps 4 and 6 was accomplished. Since the parallelism in the backsolving consists of doing equal numbers of backsolves on each processing element (number processing elements = 4*NPROCS, where NPROCS is the number of processors in the partition) we made use of a "serial" axis across the rows of the H^T and Z matrices and a "news" axis across their columns. This layout forces all components of each right hand side vector to reside on a single VU, and takes care of the load balancing by placing $N/NPROCS$ backsolves on each VU; the load balancing is therefore perfect when N is divisible by $NPROCS$, while some processors will have one more backsolve to do when N is not divisible by $NPROCS$. The upper triangular matrices R_A and R_B are, on the other hand, front end arrays stored as one-dimensional arrays with a "serial" layout directives. If H is the matrix of right hand side vectors, and Y is the matrix of solution vectors, then the CMFortran code for the N backsolves of the upper triangular matrix R_A (or R_B) is as follows:

Code Segment for Backsolves (steps 4 & 6):

```
CMF$      LAYOUT Y(:SERIAL,:NEWS)
CMF$      LAYOUT H(:SERIAL,:NEWS)
CMF$      LAYOUT T(:NEWS)
CMF$      LAYOUT R(:SERIAL)

Do 10 I = N,1,-1
      Y(I,:) = H(I,:)
      T(:) = 0.0
      DO K = I+1, N
      T(:) = R(JBEG+K)*Y(K,:) + T(:)
      ENDO
      Y(I,:) = (Y(I,:) - T(:))/R(JBEG+I)
C        JBEG = index of 1-D array R such that C
      R(JBEG+I) = (I,I)-element of R_A

10 Continue
```

4 Timing Data

Timing data was collected only in the case of square matrices A and B of order $N \times N$ ($N = m = p = n = q$). For comparison with the actual timing data we list in Table 1 the (serial) operation counts for each step of the algorithm.

Table 1: Operation Counts for $N \times N$ A and B-Matrices

N	1st QR	2nd QR	RHS	1st BS
N	$(4/3)N^3$	$(4/3)N^3$	$4N^3$	N^3

transp	2nd BS	Perm	Total	Res
-	N^3	$4N^3$	$(38/3)N^3$	$4N^3 + N^2$

In Table 2 we give the CM Busy times in seconds for each step of the algorithm when the order N of A and B is 1024. This run was done on a 32-node partition; consequently, 128 VUs were employed, so the number of backsolves per VU done in steps 4 and 6 was 1024/128 = 8. For this run the residual from the 1024×1024 matrix in (16) was

$$||T - (B \cdot X \cdot A^T)||_F = 0.000546.$$

Table 2: CM Busy Time (in seconds) for 1024 × 1024 A and B-Matrices

NPROCS	1st QR	2nd QR	RHS	1st BS	transp	2nd BS	Perm	Total	Res
32	12.742	12.793	3.959	10.838	0.099	10.839	3.926	55.196	3.940

In Table 3 the megaflop rate for each step of the algorithm is given using the data from Tables 1 and 2.

Table 3: Megaflops/sec for 1024 × 1024 A and B-Matrices (1M = 1 Megaflop/sec)

NPROCS	1st QR	2nd QR	RHS	1st BS	transp	2nd BS	Perm	Total	Res
32	112.36M	111.91M	1084.86M	99.00M	-	99.06M	1098.98M	166.09M	-

The effectiveness of the explicit backsolve coding in section 3 can be seen by comparing the measured time in Table 2 to the minimum theoretically feasible time. Since one store operation requires 1.0 microseconds and one operation count (multiplication, addition or division) requires 0.5 microseconds the total expected theoretical time for one backsolve of an $N \times N$ matrix is

$$1.0(N^2/2 + 2.5N) + 0.5N^2 = N^2 + 2.5N \, microsecs,$$

or 1.05 seconds when $N = 1024$. Accordingly, by the load balancing implemented in the backsolve coding, there will be 8 backsolves done on each of 128 VUs, so the expected minimum time on each VU is 8*1.05 = 8.42 seconds. The measured CM Busy time of 10.84 seconds for each of the two sets of backolves thus represents 128.7 % of the theoretically expected minimum time.

5 Acknowledgements

We thank Vance Faber for his support of this work. The first author acknowledges support from Contract No. 1030N0014-9H and the second author acknowledges support from a graduate student research assistantship at Los Alamos National Laboratory. We thank several colleagues from the Computer Research and Applications Division at LANL, particularly Ralph Brickner, Wayne Joubert, Olaf Lubeck, and Tom Manteuffel for helpful discussions, and several consultants from Thinking Machines Corporation, particularly Pablo Tamayo, Daryl Grunau and Richard Shapiro for frequent and useful help.

References

[1] D.W. Fausett and C.T. Fulton, Large Least Squares Problems involving Kronecker Products, SIAM J. Matrix Anal. Appl. 15 (1) (1994), 219-227.

[2] D.W. Fausett, C.T. Fulton and H. Hashish, Improved Parallel QR Method for Large Least Squares Problems involving Kronecker Products, Technical Report, Department of Applied Mathematics, Florida Insitute of Technology, 1994.

[3] CMSSL for CMFortran, Vers. 3.2, Thinking Machines Corporation, Cambridge, Mass., April 1994.