Data Parallel Algorithms for Solving Least Squares Problems by QR Decomposition

Kontoghiorghes, Erricos

702

Data Parallel Algorithms for Solving Least Squares Problems by QR Decomposition

Erricos John Kontoghiorghes

Department of Computer Science, Queen Mary and Westfield College, Mile End Road, London E1 4NS, UK.

Summary

The QR decomposition is an important operation for solving least squares problems. Data parallel algorithms for forming the QRD are described. The algorithms are based on the Householder, Givens rotations and Gram-Schmidt methods. Using regression, accurate timing models have been constructed for measuring the performance of the algorithms on a massively parallel SIMD system. A comparison of the timing models reveal the superiority in performance of the data parallel Householder algorithm, when the orthogonal matrix is not formed.

Keywords: QR Decomposition; Data Parallelism; SIMD Systems; Timing Models;

Introduction

The application of the QR Decomposition (QRD) for solving least squares problems is well known [1, 9, 12]. The data matrix A is transformed into the upper triangular matrix R after it is premultiplied by the orthogonal Q^T matrix. Throughout the paper, the QRD has the form

$$Q^T A \;=\; \begin{pmatrix} R \\ 0 \end{pmatrix}, \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$, $Q \in \mathbb{R}^{m\times m}$, $R \in \mathbb{R}^{n\times n}$ and $m > n$. It is assumed that A has full column rank and the orthogonal matrix Q is not required to be stored.

Within the context of the standard linear model

$$y \;=\; Ax + \varepsilon\;;\quad \varepsilon \sim N(0, \sigma^2 I_m)$$

after computing (1), the least squares estimator of x is derived as the solution of the triangular system $Rx = y_1$, where y_1 consists of the first n elements of $Q^T y$. If $\varepsilon \sim N(0, \sigma^2 \Omega)$, where Ω is non-diagonal, then the QRD can be used to derive the best linear unbiased estimator of x by treating the estimation problem as a restricted linear least squares problem [5, 6, 8].

Householder, Givens rotations and Gram-Schmidt are the main methods for computing the QRD. Compound Givens rotations are normally considered appropriate for zeroing targeted individual elements of A while Householder method compute the QRD of A by annihilating elements on the grand scale. The (classical and modified) Gram-Schmidt method derives the upper triangular matrix R by orthogonalizing the columns of A.

In this paper we describe data parallel algorithms for forming the QRD and compare their performance on a massively parallel SIMD (Single Instruction, Multiple Data) system using accurate timing models. The massively parallel computer used is the MasPar MP-1208 with 8192 processing elements.

Data Parallelism and the MasPar SIMD System

In a data parallel programming paradigm, the program instructions are executed serially, but instructions operate (optionally) on many elements of a large data structure simultaneously. Data parallel paradigm is not *constraint* to a particular parallel architecture and provides a natural way of programming parallel computers. The programmer does not explicitly manage processes, communication or synchronization. However, it is possible to describe how data structures such as arrays are partitioned and distributed among processors, since mapping of the data can affect performance significantly. Examples of languages that support data parallelism are Fortran 90 and High Performance Fortran (HPF) [14].

The MasPar SIMD system is composed of a *front end* (a DECstation 5000) and a Data Parallel Unit (DPU). The parallel computations are processed by the Processing Element (PE) array in the DPU, while serial operations are performed on the *front end*. The 8192 PEs of the MP-1208 are arranged in a $\mathrm{e}_1 \times \mathrm{e}_2$ matrix, where $\mathrm{e}_1 = 128$ and $\mathrm{e}_2 = 64$. The default mapping distribution in the MasPar is cyclic. In a cyclic distribution, an n element vector and an $m \times n$ element matrix are mapped on $\lceil n/\mathrm{e}_1\mathrm{e}_2 \rceil$ and $\lceil m/\mathrm{e}_1 \rceil \lceil n/\mathrm{e}_2 \rceil$ layers of memory respectively. Figure *1* shows the mapping of a 160×100 matrix and a 16384-element vector on the MP-1208. Other processor mappings are available for mapping efficiently arrays on the PE array, when the default cyclic distribution is not the best choise [10].

The main languages for programming the MasPar are the *MasPar Fortran* (hereafter MF) and *MasPar Programming Language*. The language used for implementing the algorithms was MF, which is based on Fortran 77 supplemented with array processing extensions from the standard Fortran 90.

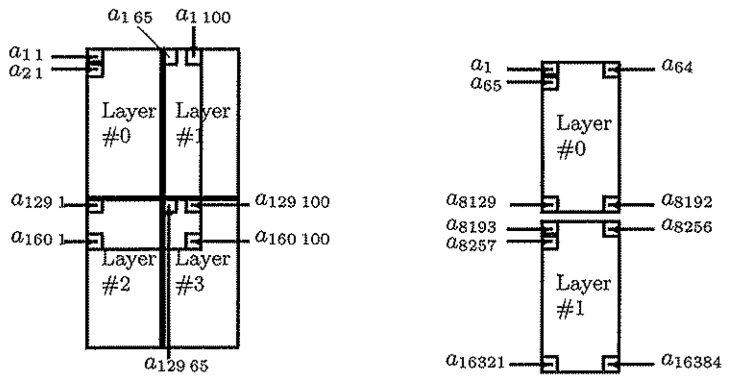

Figure 1: Cyclic mapping of a matrix and a vector on the MasPar MP-1208.

These array processing extensions map naturally on the DPU of the MasPar. MF also support the **forall** statement of HPF, which looks most like a parallel **do** loop. For example given the vectors $h \in \mathbb{R}^m$ and $z \in \mathbb{R}^n$, the product $A = hz^T$ can be computed in parallel by

$$\textbf{forall}(i = 1 : m,\ j = 1 : n)\ A_{i,j} = h_i * z_j \tag{2}$$

or

$$A = \textbf{spread}(h, 2, n) * \textbf{spread}(z, 1, m)\,. \tag{3}$$

The computations in the *rhs* of (2) are executed simultaneously $\forall i, j$. In (3) the **spread** commands construct two $m \times n$ matrices. Each column of the first matrix is a copy of h and each row of the second matrix is a copy of z. The matrix A is computed by multiplying the two matrices element by element. In both cases, the assignment on matrix A's elements can be made conditional using a conformable logical matrix to mask with true values the elements participating in the assignment [7].

The time to execute a single arithmetic operation such as $*$, $+$ or **sqrt** on an $m \times n$ matrix ($m, n \leq \mathrm{e}_1\mathrm{e}_2$), depends on the number of memory layers require to map the matrix on the DPU, that is $\lceil m/\mathrm{e}_1 \rceil \lceil n/\mathrm{e}_2 \rceil$. If however a replication, reduction or a permutation function such as **spread** or **sum** is applied to the $m \times n$ matrix, then the execution time also depends on $\lceil m/\mathrm{e}_1 \rceil$ and $\lceil n/\mathrm{e}_2 \rceil$ [2]. This implies that the execution time model of a sequence of arithmetic operations and (standard) array transformation functions on an $m \times n$ matrix, is given by

$$\phi_1(m, n) \quad = \quad c_0 + c_1 \lceil m/\mathrm{e}_1 \rceil + c_2 \lceil n/\mathrm{e}_2 \rceil + c_3 \lceil m/\mathrm{e}_1 \rceil \lceil n/\mathrm{e}_2 \rceil\,, \tag{4}$$

where c_i ($i = 0, \ldots, 3$) are constants and they can be found by experiments. The above model can describe adequately the execution time of (2) and (3).

If m or n is greater than e_1e_2, then the timing model (4) should also include combination of the factors $\lceil m/e_1e_2 \rceil$ and $\lceil n/e_1e_2 \rceil$, which correspond to the number of layers required to map a column and a row of the matrix on the DPU. In order to simplify the performance analysis of the parallel algorithms, it is assumed that $m, n \leq e_1e_2$ and the dimension of the data matrix is multiple of e_1 and e_2.

The Householder Factorization Method

The orthogonal matrix Q^T in (1) is the product of the n Householder transformations $H^{(n)}, \ldots, H^{(2)}, H^{(1)}$. The $m \times m$ Householder transformation $H^{(i)}$ is of the form $H^{(i)} = \begin{pmatrix} I_{i-1} & \\ & \tilde{H}^{(i)} \end{pmatrix}$, where $\tilde{H}^{(i)} = I_{m-i+1} - hh^T/b$, $b = h^Th/2$ and a zero dimension denotes a null matrix. It can be verified that $H^{(i)}$ is symmetric and orthogonal, that is $H^{(i)T} = H^{(i)}$ and $H^2 = I_m$. If $A^{(0)} = A$ and

$$A^{(i)} = H^{(i)}A^{(i-1)} = \begin{array}{c c} \begin{array}{cc} i & n-i \end{array} & \\ \begin{pmatrix} R_{11}^{(i)} & R_{12}^{(i)} \\ & \tilde{A}^{(i)} \end{pmatrix} & \begin{array}{l} i \\ m-i \end{array} \end{array} \quad (1 \leq i < n), \tag{5}$$

where $R_{11}^{(i)}$ is upper triangular, then $H^{(i+1)}$ is applied from the left of $A^{(i)}$ to annihilate the last $m - i - 1$ elements of the first column of $\tilde{A}^{(i)}$. The transformation $H^{(i+1)}A^{(i)}$ affects only $\tilde{A}^{(i)}$ and it follows that $A^{(n)} = \binom{R}{0}$. A complete discreption of the Householder factorization method can be found in [1, 12].

The steps for computing the QRD (1) by data parallel Householder transformations are given by the compact *Algorithm 1*. A notation similar to that of Fortran 90 has been used for sectioning arrays. The matrix references $A_{:,i}$ and $A_{i:,i:}$ denote the ith column of A and the submatrix $\tilde{A}^{(i-1)}$, respectively. The application of $H^{(i)}A^{(i-1)}$ is activated by line 3 and the time required to compute this transformation is given by $\phi_1(m - i + 1, n - i + 1)$. Thus the total time spent on computing all the Householder transformations is $\phi_2(m, n) = \sum_{i=1}^{n} \phi_1(m - i + 1, n - i + 1)$. It can be observed that the application of the ith and jth transformation have the same execution time if $\lceil (m-i+1)/e_1 \rceil = \lceil (m-j+1)/e_1 \rceil$ and $\lceil (n-i+1)/e_2 \rceil = \lceil (n-j+1)/e_2 \rceil$.

Algorithm *1* has been implemented on the MP-1208 and a sample of approximately 400 execution times has been generated for various M and N, where $m = Me_1$ and $n = Ne_2$. Evaluating $\phi_2(Me_1, Ne_2)$ and using regression analysis, the estimated execution time (sec x 10^2) of *Algorithm 1* is found to be

$$T_1(M, N) \quad = \quad N(14.15 + 3.09N - 0.62N^2 + 5.71M + 3.67MN).$$

The above timing model includes the overheads which are mainly the reference to the submatrix $A_{i:,i:}$ in line 3. This matrix reference results in the

```
1    def Househ_QRD $(A, m, n)$ =
2      for $i := 1$ until $n$ do
3        apply transform ( $A_{i:,i:}, m - i + 1, n - i + 1$ )
4      end do
5    end def
6    def transform $(A, m, n)$ =
7      $h := A_{:,1}$
8      $s :=$ sqrt(sum($h * h$))
9      If ($h_1 < 0.0$) then $s := -s$
10     $h_1 := h_1 + s$
11     $b := h_1 * s$
12     $z :=$ sum(spread($h, 2, n$) $* A, 1)/b$
13     forall($i = 1 : m,\ j = 1 : n$) $A_{i,j} := A_{i,j} - h_i * z_j$
15   end def
```

Algorithm 1: Data parallel Householder factorization of an $m \times n$ matrix A.

assignment of an array section of A into a temporary array and then when the procedure *transform* in line 6 has been completed, the temporary array is reassigned back into A. The overheads can be reduce by referencing a submatrix of A only if it uses less memory layers than a previous extracted submatrix (see *Algorithm 2*). This slight modification improves significantly the execution time of the algorithm which now becomes

$$T_2(M, N) \quad = \quad N(14.99 + 2.09N - 0.20N^2 + 3.19M + 1.17MN)\,.$$

The accuracy of the timing models is shown in *Table 1.*

The Gram-Schmidt Orthogonalization Method

The Modified Gram-Schmidt (*MGS*) method derives the upper triangular matrix R row by row and constructs an orthogonal basis Q_B of A [1]. With A overwritten by Q_B, the algorithmic steps of the MGS are

```
for $i := 1$ until $n$ do
  $R_{i,i} :=$ sqrt($A_{:,i}^T A_{:,i}$)
  $A_{:,i} := A_{:,i}/R_{i,i}$
  $R_{i,j} := A_{:,i}^T A_{:,j}$              $(j = i + 1, \ldots, n)$
  $A_{:,j} := A_{:,j} - R_{i,j} A_{:,i}$   $(j = i + 1, \ldots, n)$
end do
```

As in the case of *Algorithm 1*, the performance of the straightforward implementation of the MGS method will be reduce significantly by the overheads. Therefore, the $n = N\mathrm{e}_2$ steps of the MGS method are applied in N stages. At the ith stage, e_2 steps are applied to orthogonalize the $(i-1)\mathrm{e}_2+1$ to $i\mathrm{e}_2$ columns of A and also construct the corresponding rows of R. Each step of the ith ($i = 1, \ldots, N$) stage have the same execution time, namely

$$\phi_1(M\mathrm{e}_1, (N-i+1)\mathrm{e}_2) \quad = \quad c_0 + c_1 M + c_2(N-i+1) + c_3 M(N-i+1)\,.$$

Thus, the execution time of applying all $N\mathrm{e}_2$ steps of the MGS method is given by $\phi_3(M\mathrm{e}_1, N\mathrm{e}_2) = \mathrm{e}_2 \sum_{i=1}^{N} \phi_1(M\mathrm{e}_1, (N-i+1)\mathrm{e}_2)$.

The data parallel MGS orthogonalization method is shown in *Algorithm 2*. The logical matrices *maskR* and *maskA* are used so that computations are performed only on the affected parts of the $A_{:,i:}$ and $R_{i:,i:}$ submatrices. It can be observed that the latter submatrices are referenced only N times. The total execution time of *Algorithm 2* is given by

$$T_3(M,N) \quad = \quad N(9.15 + 3.12N - 0.01N^2 + 4.95M + 1.31MN)\,.$$

```
1   **def** MGS_QRD $(A, M\mathrm{e}_1, N\mathrm{e}_2)$ =
2     **for** $i := 1$ **step** $e_2$ **until** $N\mathrm{e}_2$ **do**
3       **apply** orthogonal $(A_{:,i:}, R_{i:,i:}, M\mathrm{e}_1, (N-i+1)\mathrm{e}_2)$
4     **end do**
5   **end def**
6   **def** orthogonal $(A, m, n)$ =
7     maskR := **true**
8     maskA := **true**
9     **for** $i := 1$ **until** $e_2$ **do**
10      $\mathrm{maskR}_i$ := **false**
11      $\mathrm{maskA}_{:,i}$ := **false**
12      $R_{i,i} :=$ **sqrt**(**sum**($A_{:,i} * A_{:,i}$))
13      $A_{:,i} := A_{:,i}/R_{i,i}$
14      **where**(maskR) $R_{i,:} :=$ **sum**(**spread**($A_{:,i}, 2, n) * A, 1$)
15      **forall**($j = 1 : m,\ k = 1 : n,\ \mathrm{maskA}_{j,k}$) $A_{j,k} := A_{j,k} - R_{i,k} * A_{j,i}$
16    **end do**
17  **end def**
```

Algorithm 2: Data parallel MGS orthogonalization method for computing the QRD.

It can be seen from *Table 1* that the Householder method performs better than the MGS method. The difference in the performance of the two methods arises mainly because at the ith step the MGS and Householder methods work with the $m \times (n-i+1)$ and $(m-i+1) \times (n-i+1)$ matrices respectively. An analysis of $T_2(M,N)$ and $T_3(M,N)$ reveals that for $M > N$, *Algorithm 2* is expected to perform better than the efficient implementation of *Algorithm 1* only when $N = 1$ and $M = 2$.

The Givens Rotation Method

The orthogonal matrix Q^T in (1) is the product of a sequence of *compound disjoint Givens rotations* (*cdgr*), with each compound rotation reducing to zero elements of A below the main diagonal by preserving previously annihilated elements. Figure *2* shows two sequences of *cdgr* for computing the QRD of a 12×6 matrix, where a number denotes the elements annihilated by the corresponding *cdgr*. The first Givens sequence (hereafter GS-1) has been proposed by Sameh and Kuck in [13]. It applies $m+n-2$ *cdgr* and elements are annihilated by rotating adjacent rows. The second Givens sequence (GS-2) applies fewer *cdgr* than GS-1, but when it comes to implementation the advantage of the GS-2 is offset from the communication overheads occure during the construction and application of the compound rotations [2, 11].

$$\begin{pmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ 11 & \bullet & \bullet & \bullet & \bullet & \bullet \\ 10 & 12 & \bullet & \bullet & \bullet & \bullet \\ 9 & 11 & 13 & \bullet & \bullet & \bullet \\ 8 & 10 & 12 & 14 & \bullet & \bullet \\ 7 & 9 & 11 & 13 & 15 & \bullet \\ 6 & 8 & 10 & 12 & 14 & 16 \\ 5 & 7 & 9 & 11 & 13 & 15 \\ 4 & 6 & 8 & 10 & 12 & 14 \\ 3 & 5 & 7 & 9 & 11 & 13 \\ 2 & 4 & 6 & 8 & 10 & 12 \\ 1 & 3 & 5 & 7 & 9 & 11 \end{pmatrix} \qquad \begin{pmatrix} \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ 4 & \bullet & \bullet & \bullet & \bullet & \bullet \\ 3 & 6 & \bullet & \bullet & \bullet & \bullet \\ 2 & 5 & 8 & \bullet & \bullet & \bullet \\ 2 & 4 & 7 & 10 & \bullet & \bullet \\ 2 & 4 & 6 & 9 & 12 & \bullet \\ 1 & 3 & 6 & 8 & 11 & 14 \\ 1 & 3 & 5 & 7 & 10 & 13 \\ 1 & 3 & 5 & 7 & 9 & 12 \\ 1 & 2 & 4 & 6 & 8 & 11 \\ 1 & 2 & 4 & 6 & 8 & 10 \\ 1 & 2 & 3 & 5 & 7 & 9 \end{pmatrix}$$

(a) (b)

Figure 2: Examples of Givens rotations schemes for computing the QRD.

Tha adaptation and implementation of the GS-1 to compute various forms of orthogonal factorizations on the SIMD AMT DAP-510, has been describe in [2, 4, 6]. Although a different massively parallel system has been used, the computational details are general for SIMD array processors. On the MP-1208, the execution time of computing the QRD of an $M\mathrm{e}_1 \times N\mathrm{e}_2$ matrix using the GS-1 annihilation scheme, is found to be

$$T_4(M,N) \quad = \quad N(25.64 + 5.51N - 7.94N^2 + 11.1M + 15.99MN) + 41.96M\,.$$

Clearly, the implementation of the GS-1 has the worst performance compared with the Householder and MGS data parallel algorithms.

Discussion

The performance of three algorithms to compute the QRD of a dence matrix on a massively parallel SIMD has been compared. The Householder factorization method was found to be the most efficient in terms of speed, followed by the MGS algorithm which was only slightly slower than the data parallel

		Algorithm 1		*Improved Algor. 1*		*Algorithm 2*		*GS-1*	
M	N	**Exec. Time**	$T_1(M,N)$ x 10^{-2}	**Exec. Time**	$T_2(M,N)$ x 10^{-2}	**Exec. Time**	$T_3(M,N)$ x 10^{-2}	**Exec. Time**	$T_4(M,N)$ x 10^{-2}
10	3	5.48	5.55	2.58	2.59	3.21	3.22	21.16	21.04
10	7	22.15	22.34	9.28	9.34	12.07	12.08	67.73	67.59
10	9	33.80	34.09	13.80	13.92	18.49	18.48	92.65	92.62
14	5	17.48	17.54	7.36	7.35	9.30	9.30	62.27	62.36
14	9	47.86	48.03	18.75	18.86	24.52	24.50	150.05	150.11
14	13	90.30	90.56	34.38	34.54	46.64	46.64	242.63	242.68
18	5	22.34	22.35	9.14	9.15	11.60	11.59	82.03	82.25
18	9	61.90	61.98	23.77	23.79	30.68	30.52	207.47	207.61
18	17	189.16	189.03	69.42	69.31	94.41	94.26	503.44	503.51
22	7	48.61	48.71	19.10	18.90	23.98	23.93	175.85	175.96
22	15	188.79	188.49	68.88	68.57	89.86	89.82	585.56	585.64
22	19	287.23	286.33	103.31	102.81	138.59	138.27	805.45	805.71

Table 1: Times (in sec) of computing the QRD of a $128M \times 64N$ matrix.

Householder algorithm. The implementation of the GS-1 produce by far the worst performance.

The comparison of the performance of the data parallel implementations was made using accurate timing models. These models provide an effective tool for measuring the computational speed of algorithms and they can also be used to reveal inefficiencies of parallel implementations [7]. Comparisons with performance models of various algorithms implemented on other SIMD systems, demonstrate the scalability of the execution time models presented in this paper [2, 4]. If the dimension of the data matrix A in (1) is not as it was assumed a multiple of the physical array processor, then the timing models can be used to give a range of the expected execution time of the algorithms.

Currently, within the context of SURE model estimation, two new parallel algorithms are investigated to compute the QRD of a matrix on the MasPar. The first algorithm uses the partition-updating method reported in [3], while the second algorithm uses the (vectorized) Householder factorization method, with the data matrix stored in the PE array as a long vector of order mn.

Acknowledgements

The work was supported by the Bowring Company Ltd. The facilities of the London Parallel Application Centre and Department of Computer Science at Queen Mary and Westfield College (University of London) have been used.

References

[1] G.H. Golub and C.F. Van Loan. *Matrix computations.* North Oxford Academic, 1983.

[2] E.J. Kontoghiorghes. *Algorithms for linear model estimation on massively parallel systems.* PhD Thesis, University of London, 1993.

[3] E.J. Kontoghiorghes. New parallel strategies for block updating the QR decomposition. *Parallel Algorithms and Applications*, 5(1+2):229-239, 1995.

[4] E.J. Kontoghiorghes and M.R.B. Clarke. Solving the updated and downdated ordinary linear model on massively parallel SIMD systems. *Parallel Algorithms and Applications*, 1(2):243–252, 1993.

[5] E.J. Kontoghiorghes and M.R.B. Clarke. An alternative approach for the numerical solution of seemingly unrelated regression equation models. *Computational Statistics & Data Analysis*, 19(4):369–377, 1995.

[6] E.J. Kontoghiorghes and M.R.B. Clarke. Solving the general linear model on a SIMD array processor. *Computers and Artificial Intelligence*, 1995. (in press).

[7] E.J. Kontoghiorghes, M.R.B. Clarke, and A. Balou. Improving the performance of optimum parallel algorithms on SIMD array processors: programming techniques and methods. In *Proceedings of the IEEE TENCON'93*, pages 1203–1206, Beijing, 1993. International Academic Publishers.

[8] S. Kourouklis and C.C. Paige. A constrained least squares approach to the general Guass–Markov linear model. *Journal of the American Statistical Association*, 76(375), 1981.

[9] C.L. Lawson and R.J. Hanson. *Solving Least Squares Problems*. Prentice–Hall Englewood Cliff, 1974.

[10] MasPar Computer Corporation. *MasPar System Overview*, 1992.

[11] J.J. Modi and M.R.B. Clarke. An alternative Givens ordering. *Numerische Mathematik*, 43:83–90, 1984.

[12] C.R. Rao. *Handbook of Statistics 9 (Computational Statistics)*. North-Holland, 1993.

[13] A.H. Sameh and D.J. Kuck. On stable parallel linear system solvers. *Journal of the ACM*, 25(1):81–91, 1978.

[14] C.H. Koelbel D.B. Lovemac R.S. Schreiber R.S. Steele and M.E. Zosel. *The High Performance Fortran Handbook*. The MIT Press, 1994.