716

Recursive least-squares using Householder transformations on massively parallel SIMD systems

E. J. Kontoghiorghes[a,c,1], M. Clint[b] and E. Dinenis[c]

[a] *Department of Computer Science, Queen Mary and Westfield College, Mile End Road, London E1 4NS, UK*

[b] *Dept. of Computer Science, The Queen's University of Belfast, Belfast, N. Ireland BT7 1NN, UK.*

[c] *Centre for Mathematical Trading and Finance, City University Business School, Frobisher Crescent, Barbican Centre, London EC2Y 8HB, UK.*

Abstract: Within the context of recursive least-squares, the implementation of a Householder algorithm for block updating the QR decomposition, on a massively parallel SIMD system, is considered. Initially, two implementations based on different mapping strategies for distributing the data matrices on the processing elements of the parallel computer are investigated. Timing models show that neither of these implementations is superior in all cases. In order to increase computational speed, a hybrid implementation uses performance models to partition the problem into two subproblems which are then solved by the first and second implementation, respectively.

Key words: Least-squares; Householder transformations; QR decomposition; Timing models; SIMD parallelism.

1 Introduction

It is frequently required to obtain updated least-squares solutions of a regression model where a block of data is repeatedly added. The updating can be computed by recursive formulae which have as a basic component the recalculation of the QR decomposition (QRD) [3,5,7,9,10,14]. The *recursive least-squares* (RLS) problem may be formulated as

$$\underset{x_t}{\operatorname{argmin}} \|A_t x_t - y_t\|^2; \qquad t = 1, 2, \ldots, \, , \tag{1}$$

where $\| \bullet \|$ denotes Euclidean length, A_t is an $m_t \times (n-1)$ full column rank matrix ($m_t \geq n$), $y_t \in \mathbb{R}^{m_t}$ is the response vector, x_t is the unknown vector of

[1] Corresponding author. Email: ricos@dcs.qmw.ac.uk

$n-1$ coefficients and

$$\hat{A}_t = \begin{pmatrix} A_t & y_t \end{pmatrix} = \begin{array}{c} \begin{pmatrix} \hat{A}_{t-1} \\ \tilde{A}_t \end{pmatrix} \end{array} \begin{array}{l} m_{t-1} \\ \tilde{m}_t \end{array} \qquad (m_0 = 0). \tag{2}$$

Given the QRD

$$Q_t^T \hat{A}_t = \begin{pmatrix} \hat{R}_t \\ 0 \end{pmatrix} = \begin{array}{c} \begin{array}{cc} n-1 & 1 \end{array} \\ \begin{pmatrix} R_t & u_t \\ 0 & s_t \\ 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \\ n-1 \\ 1 \\ m_t - n \end{array}, \tag{3}$$

the LS estimator of x_t is derived from the solution of $R_t x_t = u_t$, where R_t is an upper triangular non-singular matrix and Q_t is an $m_t \times m_t$ orthogonal matrix. By computing the orthogonal factorization

$$\tilde{Q}_{t+1}^T \begin{pmatrix} \hat{R}_t \\ \tilde{A}_{t+1} \end{pmatrix} = \begin{pmatrix} \hat{R}_{t+1} \\ 0 \end{pmatrix} = \begin{pmatrix} R_{t+1} & u_{t+1} \\ 0 & s_{t+1} \\ 0 & 0 \end{pmatrix}, \tag{4}$$

the updated LS estimator of x_{t+1} may be obtained by solving

$$R_{t+1} x_{t+1} = u_{t+1}, \tag{5}$$

where $\tilde{Q}_{t+1}$ is an $(n + \tilde{m}_{t+1}) \times (n + \tilde{m}_{t+1})$ orthogonal matrix and $\hat{R}_{t+1} \in \mathbb{R}^{n \times n}$ is upper triangular. Thus, after computing the QRD of $\hat{A}_1$, the recursive formulae (4) and (5) may be used to derive the updated LS estimator of x_{t+1} for $t = 1, 2, \ldots,$. Observe that the orthogonal matrix Q_{t+1}^T in the QRD of $\hat{A}_{t+1}$ is given by

$$\begin{pmatrix} \tilde{Q}_{t+1}^T & 0 \\ 0 & I_{m_t - n} \end{pmatrix} \begin{pmatrix} I_n & 0 & 0 \\ 0 & 0 & I_{\tilde{m}_{t+1}} \\ 0 & I_{m_t - n} & 0 \end{pmatrix} \begin{pmatrix} \tilde{Q}_t^T & 0 \\ 0 & I_{\tilde{m}_{t+1}} \end{pmatrix}.$$

Householder reflections and Givens rotations are the main numerically stable methods used to compute the factorization in (4) [4–10,16,17,19]. Givens method is found to be superior in speed to Householder method only when the blocks of new observations $\tilde{A}_t$ comprise a small number of rows compared with the number of regressors [7,10]. Under the assumption that $\tilde{m}_{t+1} > n$, parallel algorithms based on Householder transformations are considered below for computing the factorization (4) on a massively parallel SIMD machine. The orthogonal matrix $\tilde{Q}_{t+1}$ will not be explicitly constructed and the trivial solution of the triangular system (5) will not be discussed.

2 SIMD Implementation

The orthogonal matrix $\tilde{Q}_{t+1}^T$ in (4) may be defined as a product of Householder reflectors, where the ith reflector reduces to zero the ith column of $\tilde{A}_{t+1}$ by preserving the upper triangular structure of $\hat{R}_t$. Figure 1 shows the steps in applying the Householder transformations in a data-parallel mode, where $\hat{R}_t$ is overwritten by $\hat{R}_{t+1}$. For notational convenience we let $m = \tilde{m}_{t+1}$ and denote the matrices $\hat{R}_t$ and $\tilde{A}_{t+1}$ by R and A, respectively.

The Householder algorithm has been implemented in MasPar-Fortran on the MasPar MP-1208 [15]. This SIMD system has 8192 Processing Elements (PEs) arranged in a 2D-array of size $\mathrm{e}_2 \times \mathrm{e}_1$, where $\mathrm{e}_1 = 128$ and $\mathrm{e}_2 = 64$. The main mapping layouts for distributing the $m \times n$ matrix A over the PEs are the (default) *cyclic*, *column* and *row* layouts, which use $\lceil m/\mathrm{e}_1 \rceil \lceil n/\mathrm{e}_2 \rceil$, $n\lceil m/\mathrm{e}_1\mathrm{e}_2 \rceil$ and $m\lceil n/\mathrm{e}_1\mathrm{e}_2 \rceil$ layers of memory, respectively [1]. The memory layers have dimension $\mathrm{e}_2 \times \mathrm{e}_1$ and are assumed to be arranged in an $M \times N$ grid G, where M and N depend on the chosen layout. Under the *cyclic-layout*, the ith row of A, $A_{i,:}$, resides in the jth column of layers $G_{k,:}$, where $j = ((i-1) \bmod \mathrm{e}_1) + 1$, $k = \lceil i/\mathrm{e}_1 \rceil$, $M = \lceil m/\mathrm{e}_1 \rceil$ and $N = \lceil n/\mathrm{e}_1 \rceil$. In the *column-layout* the ith column of A, $A_{:,i}$, resides in $G_{:,i}$, where $M = \lceil m/\mathrm{e}_1\mathrm{e}_2 \rceil$ and $N = n$, while in the *row-layout* $A_{i,:}$ resides in $G_{i,:}$, where $M = m$ and $N = \lceil n/\mathrm{e}_1\mathrm{e}_2 \rceil$.

A mapping layout is chosen so that the maximum number of PEs remains active during the computations without, however, increasing the communication overheads between the PEs. Since $m > n$, the *row-layout* will be inefficient compared with the *column-layout*. Consequently, the performances of the Householder algorithm using *cyclic* and *column* layouts are considered. The computational details of the implementations on the MasPar are not shown. However, a pseudo code version is given in Fig. 1. Similar implementations have been previously considered for computing the QRD under the assumption that the dimensions of the matrices are exact multiples of the corresponding dimensions of the array processor and that none of their dimensions exceed $\mathrm{e}_1\mathrm{e}_2$ [2,11,12].Here the only constraint imposed on the dimensions of the matrices is that $m > n$.

```
1   for $i = 1, 2, \ldots, n$ do
2     $s$ := sqrt $(R_{i,i}^2 + \|A_{:,i}\|^2)$
3     if $(R_{i,i} < 0)$ then $s := -s$
4     $\eta$ := $R_{i,i} + s$
5     $c$ := $s * \eta$
6     $Z^T$ := $(\eta * R_{i,i:} + A_{:,i}^T A_{:,i:})/c$
7     $R_{i,i:}$ := $R_{i,i:} - \eta * Z^T$
8     $A_{:,i:}$ := $A_{:,i:} - A_{:,i} Z^T$
9   end-do
```

Fig. 1. The *pseudo* data-parallel Householder algorithm.

2.1 Performance Models

Statistical methods are employed to construct performance models for the Householder implementations. These models can realistically evaluate the execution speed of the algorithms for various values of m and n. The order of complexity of the algorithms will be the same if other 2D SIMD array processors are used.

Using a *cyclic-layout* to map the matrices $A \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{n\times n}$ onto the PE array, the time (sec $\times 10^{-3}$) required to apply the ith Householder transformation is found to be

$$\begin{aligned}\Phi_1(m,n,i) = {} & 10.65\lceil (n+1-i)/\mathrm{e}_2\rceil + 1.66\lceil (n+1-i)/\mathrm{e}_2\rceil\lceil m/\mathrm{e}_1\rceil \\ & + 4.72\lceil m/\mathrm{e}_1\rceil + 0.03(n+1-i)\lceil m/\mathrm{e}_1\rceil . \end{aligned} \tag{6}$$

The explanatory factors of the above timing model have been derived by considering the number of layers involved in the arithmetic computations and the number of times layers are replicated or reduced. Regression analysis is then employed to find the least squares estimators of the coefficients in the model.

The total time spent in applying the Householder reflections is thus given by $\sum_{i=1}^{n}\Phi_1(m,n,i)$, which is equivalent to :

$$\begin{aligned}\Phi_2(m,n) = {} & \mathrm{e}_2\sum_{i=1}^{N}(10.65i + 1.66Mi + 4.72M) + 0.03n(n+1)M/2 \\ & - (N\,\mathrm{e}_2 - n)(10.65N + 1.66NM + 4.72M), \end{aligned} \tag{7}$$

where $N = \lceil n/\mathrm{e}_2\rceil$ and $M = \lceil m/\mathrm{e}_1\rceil$. However, the overheads of the implementation which are mainly the passing of arguments (subarrays) to various routines are not included in this model. Evaluating $\Phi_2(m,n)$, and using backward stepwise regression on a sample of more than 5000 execution times, a highly accurate timing model for the *cyclic-layout* Householder implementation is found to be :

$$\begin{aligned}T_1(m,n) = {} & N(1391.0 + 66.8N^2 + 388.2M + 115.5NM) \\ & - (N\,\mathrm{e}_2 - n)(17.42 + 4.53M + 2.79N^2 + 3.66NM) \\ = {} & N(276.12 + 98.28M - 118.74NM - 111.76N^2) \\ & + n(17.42 + 4.53M + 2.79N^2 + 3.66NM). \end{aligned} \tag{8}$$

Calculations shown that the residuals are normally distributed. Thus, the hypothesis tests made during the selection of this model are justified [18]. The adequacy of the latter model, measured by the coefficient of determination, is found to be 99.99%. Figure 2 shows the ratio between the predicted and actual execution times using $T_1(m,n)$ and $\Phi_2(m,n)$. It may be observed that, overall, the predictions given by $T_1(m,n)$ are more accurate than those of $\Phi_2(m,n)$.

Using *cyclic* and *column* layouts to map respectively the matrices R and A onto the PEs, a model for estimating the execution time of the ith Householder reflection is :

$$\Phi_3(m,n,i) = c_0 + c_1(n+1-i) + c_2(n+1-i)\lceil m/\mathrm{e}_1\,\mathrm{e}_2\rceil + c_3\lceil n/\mathrm{e}_2\rceil, \tag{9}$$

where $c_0,\ldots,c_3$ are constants. Evaluating $\sum_{i=1}^{n}\Phi_3(m,n,i)$ and using regression analysis, the execution time of the *column-layout* implementation is found

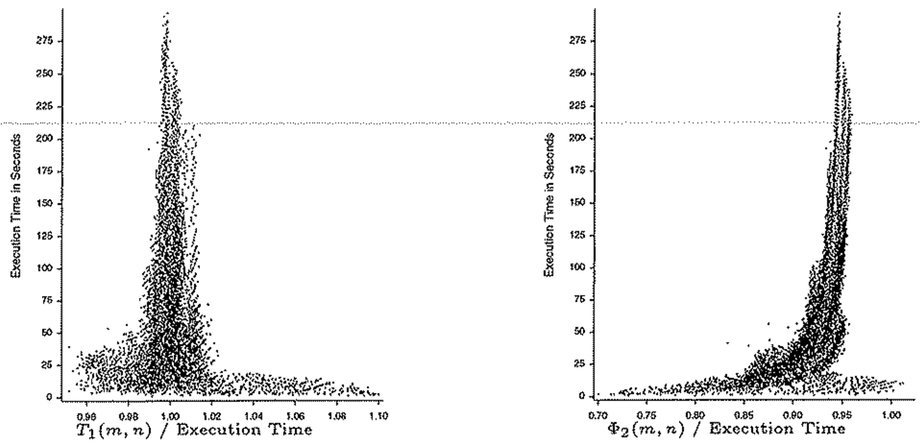

Fig. 2. Ratio between predicted and actual execution times.

to be :

$$T_2(m,n) = n\Big(13.51 + 2.69n + 1.33(n+1)\lceil m/\mathrm{e}_1\,\mathrm{e}_2\rceil + 2.98\lceil n/\mathrm{e}_2\rceil\Big). \quad (10)$$

From Fig. 3 it can be observed that neither of the implementations is superior in all cases. The efficiency of the *cyclic-layout* implementation improves compared with that of the *column-layout* implementation, for fixed m and increasing n. Table 1 shows that the *column-layout* is superior for very large m and relatively small n.

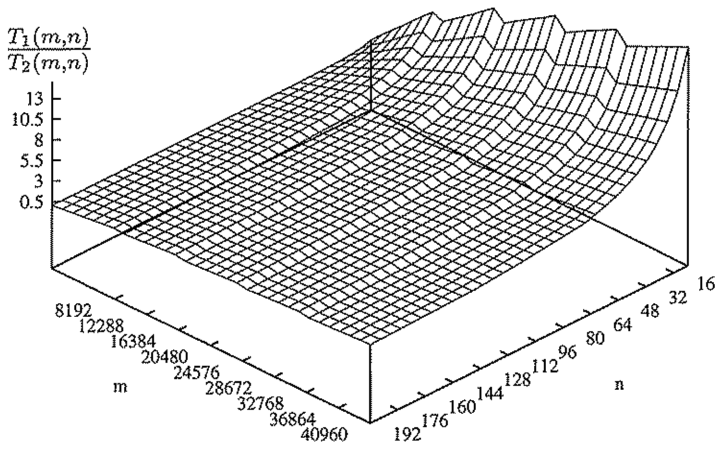

Fig. 3. Ratio of the execution times produce by the models of the *cyclic-layout* and *column-layout* implementations.

3 The Hybrid SIMD Implementation

The results above suggest that the application of the n required Householder reflections be divided into two parts. In the first part, n_1 reflections are applied

Table 1
Execution times in seconds for $m = 11264$.

n	*Cyclic*	$T_1(m,n)$	*Column*	$T_2(m,n)$	n	*Cyclic*	$T_1(m,n)$	*Column*	$T_2(m,n)$
22	14.06	14.68	3.04	3.02	62	43.60	44.30	21.79	21.80
30	20.39	20.39	5.63	5.40	70	50.62	50.17	27.90	27.82
38	27.43	26.53	8.68	8.47	78	58.13	58.74	34.46	34.35
46	33.52	32.45	12.18	12.23	86	66.56	67.31	41.71	41.56
54	38.91	38.38	16.65	16.67	94	75.94	75.88	49.46	49.46

All timing model results have been multiplied by 10^3.

to annihilate the first n_1 columns of A using *cyclic-layout*; in the second stage the remaining $n_2 = n - n_1$ reflections reduce to zero the submatrix $A_{:,n_1+1:}$ using *column-layout*, where $A_{:,n_1+1:}$ comprise the last n_2 columns of A. Let $\tilde{T}_1(m,n,n_1)$ be the time required to complete the first stage. Then the total execution time of the hybrid implementation is given by

$$T_4(m,n,n_1) = \tilde{T}_1(m,n,n_1) + T_2(m,n-n_1) + T_3(m,n-n_1), \qquad (11)$$

where

$$T_3(m,n-n_1) = \lceil m/\mathrm{e}_1 \rceil (65.45 \lceil (n-n_1)/\mathrm{e}_2 \rceil + 0.62(n-n_1))$$

is the time (sec $\times 10^{-6}$) required to remap the submatrix $A_{:,n_1+1:}$ from *cyclic-layout* to *column-layout*.

The value of n_1, which may not be unique, is chosen to minimize $T_4(m,n,n_1)$. That is, n_1 is the solution of

$$\underset{n_1}{\operatorname{argmin}}\, T_4(m,n,n_1) \quad \text{subject to} \quad \begin{cases} 0 \le n_1 \le n \\ n_1 \text{ is integer} \end{cases}, \qquad (12)$$

which may be easily determined by (simultaneously) computing $T_4(m,n,n_1)$ for $n_1 = 0,\ldots,n$ and selecting the value(s) which minimize T_4. Note that $T_4(m,n,n_1)$ is a statistical timing model and includes an unpredicted random error component. Consequently, the solution of (12) may not yield the true value of n_1 which minimizes the execution time of the hybrid algorithm.

An alternative method for estimating n_1 is to compare the total number of memory layers used for different values of n_1. In this case n_1 is the solution of

$$\underset{n_1}{\operatorname{argmin}} \left(\sum_{i=1}^{n_1} \lceil m/\mathrm{e}_1 \rceil \lceil (n+1-i)/\mathrm{e}_2 \rceil + \frac{1}{2}(n-n_1)(n-n_1+1) \lceil m/\mathrm{e}_1\,\mathrm{e}_2 \rceil \right), \qquad (13)$$

where $0 \ge n_1 \ge n$. This estimation method does not, however, take into account the cost of remapping $A_{:,n_1+1:}$. Better estimates of n_1 might possibly be obtained by constructing more accurate timing models than that given by (12), or by introducing a weighting factor into (13) which will take account of the implementation overheads.

Table 2 shows the actual execution times for the three implementations, where the negligible time required to compute the estimates of n_1 is not included and $\tilde{T}_1(m,n,n_1)$ has been computed as $\sum_{i=1}^{n_1} \Phi_1(m,n,i)$. The estimates for n_1 obtained using (12) and (13) are denoted by $\hat{n}_1$ and n_1^*, respectively. In most cases, the two estimation methods yield different values for n_1. The execution time for the hybrid implementation, using the estimation n_1^*, is found to be more accurate than those using $\hat{n}_1$ in approximately half of the cases. In some instances, for very small n and relatively large M, the hybrid implementation

Table 2
Execution times in seconds of the three implementations, where $M = m/\mathrm{e}_1$.

M	n	$\hat{n}_1$	Hybrid with $\hat{n}_1$	n_1^*	Hybrid with n_1^*	*Cyclic*	*Column*	M	n	$\hat{n}_1$	Hybrid with $\hat{n}_1$	n_1^*	Hybrid with n_1^*	*Cyclic*	*Column*
10	32	25	3.52	23	2.81	3.28	5.16	10	128	121	15.94	119	15.47	15.70	68.91
10	64	57	6.10	55	6.09	6.33	18.05	10	160	153	21.79	151	21.56	21.80	106.17
10	96	89	10.78	87	11.01	10.78	39.61	10	192	185	27.43	183	27.66	27.66	151.64
20	32	16	4.21	13	4.46	5.62	5.63	20	128	112	27.19	109	26.96	27.66	70.78
20	64	48	10.78	45	10.54	11.24	19.46	20	160	144	38.20	141	38.44	38.67	109.22
20	96	80	18.98	77	18.99	19.45	41.24	20	192	176	48.74	173	48.76	49.22	154.69
30	32	7	5.63	3	5.85	7.97	6.33	30	128	103	37.97	99	37.97	39.85	72.90
30	64	39	14.53	35	14.07	15.94	20.39	30	160	135	54.15	131	54.14	56.02	111.79
30	96	71	26.49	67	26.01	28.13	42.89	30	192	167	69.15	163	68.91	70.78	158.20
30	128	103	37.97	99	37.97	39.61	72.89	40	128	93	48.29	89	48.05	51.79	75.24
40	32	0	6.80	0	6.80	10.54	6.79	40	160	125	69.15	121	68.91	73.13	114.60
40	64	29	17.10	25	16.88	21.09	21.32	40	192	157	88.83	153	88.59	92.57	161.26
40	96	61	32.57	57	32.81	36.79	44.29	60	128	52	69.61	9	75.46	76.17	79.46
40	128	93	47.81	89	47.82	52.04	75.24	60	160	84	100.31	0	120.00	107.35	119.29
50	32	0	7.51	0	7.27	12.89	7.27	60	192	116	129.38	13	158.44	136.18	167.10
50	64	18	19.22	15	19.69	26.02	22.26	80	128	52	92.10	89	88.60	100.54	105.93
50	96	50	38.67	0	46.18	45.47	45.94	80	160	84	132.42	121	129.61	141.57	159.85
50	128	82	57.43	29	65.63	63.98	77.58	80	192	116	170.62	153	167.35	179.30	223.83
60	32	0	7.97	0	7.74	15.23	7.73	100	128	26	102.65	29	103.12	124.92	109.93
60	64	5	22.49	5	22.49	30.93	23.67	100	160	58	153.04	61	153.52	175.78	165.01
60	96	20	44.99	0	47.82	54.14	47.58	100	192	90	200.62	93	201.33	222.65	230.39
60	128	52	69.38	9	75.24	76.18	79.22	120	128	0	115.08	9	112.03	149.07	114.14
70	32	0	9.85	0	9.85	17.82	9.38	120	160	0	170.85	0	170.62	210.00	170.15
70	64	14	26.72	30	27.43	35.62	29.77	120	192	31	228.29	13	230.39	265.54	236.26
70	96	46	53.44	62	53.90	62.57	61.87	140	128	8	137.35	35	138.05	173.90	140.62
70	128	78	79.46	94	79.69	88.36	103.59	140	160	14	206.26	67	207.65	243.98	210.23
80	32	0	10.31	0	10.31	20.15	10.54	140	192	46	278.44	99	274.21	309.38	292.26
80	64	5	30.00	25	28.83	40.78	31.17	32	128	101	40.55	97	39.84	42.18	73.59
80	96	20	59.77	57	59.30	71.01	63.29	32	160	133	57.19	129	57.19	59.30	112.26
80	128	52	91.87	89	88.36	100.31	106.18	32	192	165	73.12	161	73.12	75.23	158.68
90	32	0	11.01	0	10.78	22.73	10.78	32	224	197	94.92	193	94.93	97.04	213.76
90	64	0	32.57	20	30.46	45.71	32.34	32	256	229	113.90	225	114.14	116.26	276.10
90	96	8	63.29	7	63.51	79.69	65.15	64	128	46	72.42	1	80.15	81.10	79.93
90	128	40	99.14	39	98.68	112.73	108.05	64	160	78	105.01	0	121.17	114.15	120.70
100	32	0	11.48	0	11.48	25.08	11.48	64	192	110	135.71	1	168.51	144.60	168.76
100	64	0	33.52	15	31.87	50.15	33.04	64	224	142	177.43	0	225.93	186.10	225.23
100	96	0	66.79	0	66.80	88.36	66.56	64	256	174	214.93	1	289.22	223.59	289.46
100	128	26	102.65	29	103.12	124.69	109.93	96	128	32	101.48	33	101.24	120.00	109.21
110	32	0	11.95	0	12.19	27.65	11.71	96	160	64	149.77	65	149.77	168.76	163.35
110	64	0	34.46	10	33.51	55.55	34.45	96	192	96	195.46	97	195.94	213.99	228.29
110	96	0	68.20	0	68.68	96.79	67.97	96	224	128	256.65	129	256.88	275.86	305.62
110	128	11	108.52	19	107.35	136.88	112.26	96	256	160	312.42	161	311.96	330.70	392.10
120	32	0	12.66	0	12.42	30.00	12.18	128	128	0	116.49	1	116.24	158.91	116.02
120	64	0	35.85	5	34.93	60.46	35.40	128	160	0	172.74	0	172.96	223.36	172.26
120	96	0	70.07	0	69.85	105.47	69.61	128	192	14	234.15	1	239.29	283.35	239.30

is reduced to the *column-layout* implementation, that is, the estimated value of n_1 is zero (see for example the case $n = 32$ and $M = 100$). With the exception of few cases, the hybrid implementation is more efficient than both the *cyclic-layout* and *column-layout* implementations.

4 Conclusion

The performance, on a SIMD computer, of the data-parallel Householder algorithm for computing the orthogonal factorization given by (4), has been considered. Timing models have been constructed for estimating the execution speed of the algorithm when *cyclic-layout* and *column-layout* mapping strategies are used. These accurate timing models revealed that neither of the data-mapping distributions is superior for all values of m and n $(m > n)$. A new hybrid implementation which switches from *cyclic-layout* to *column-*

layout has been proposed. The hybrid algorithm first applies n_1 $(0 \geq n_1 \geq n)$ transformations under the *cyclic-layout* regime, and then the *column-layout* implementation is employed to complete the factorization.

Two methods have been used for deriving an estimate for n_1. The first method is based on minimizing the estimated execution time provided by the timing model for the algorithm, while the second method is based on minimizing the total number of memory layers used. The two estimators of n_1 are not always identical and are probably different from the optimum value of n_1, which minimizes the execution time of the hybrid algorithm. However, in nearly all the experiments performed, the hybrid algorithm using both estimators of n_1 is found to have the best performance of the three candidates. The improvement in performance become significant for real-time applications in which a very large number of data updatings are required.

For other SIMD systems, the value of n_1 may be best derived using the straightforward minimization of the total number of memory layers used, rather than minimizing the estimated time given by performance model in (11) which requires the time consuming re-determination of the coefficients of the various timing models.

Similar hybrid algorithms may be used to compute the orthogonal factorization (4), based on Givens rotations and Householder reflections, when $m < n$. In this case, the efficiency of the *row-layout* mapping strategy should be investigated as a (possibly) more efficient alternative. The hybrid approach could also be employed to improve the efficiency of the SIMD algorithms proposed in [3,9–13].

Acknowledgements

The work was carried out using the facilities of the London Parallel Applications Centre at Queen Mary and Westfield College, University of London. The research of the first author was supported by the C.T. Bowring Company Ltd. The authors are grateful to Prof. Dennis Parkinson for his constructive comments.

References

[1] C. Bendtsen, C. Hansen, K. Madsen, H.B. Nielsen, and M. Pinar. Implementation of QR up- and downdating on a massively parallel computer. *Parallel Computing*, 21:49–61, 1995.

[2] G.S.J. Bowgen and J.J. Modi. Implementation of QR factorization on the DAP using Householder transformations. *Computer Physics communications*, 37,

1985.

[3] M. Clint, J.S. Weston, and J.B. Flannagan. Efficient Gram-Schmidt orthogonalisation on an array processor. In B. Buchberger and J. Volkert, editors, *Parallel Processing: CONPAR 94-VAPP VI*, volume 854 of *LNCS*, pages 218–228. Springer–Verlag, 1994.

[4] J.W. Daniel, W.B. Gragg, L. Kaufman, and G.W. Stewart. Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization. *Mathematics of Computation*, 30(136):772–795, 1976.

[5] R.W. Farebrother. *Linear least squares computations (Statistics : Textbooks and Monographs)*, volume 91. Marcel Dekker, Inc., 1988.

[6] P.E. Gill, G.H. Golub, W. Murray, and M.A. Saunders. Methods for modifying matrix factorizations. *Mathematics of Computation*, 28(126):505–535, 1974.

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

[8] C.S. Henkel and R.J. Plemmons. Recursive least squares on a hypercube multiprocessor using the covariance factorization. *SIAM Journal of Scientific and Statistical Computing*, 12(1):95–106, 1991.

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

[10] 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.

[11] E.J. Kontoghiorghes and M.R.B. Clarke. Solving the general linear model on a SIMD array processor. *Computers and Artificial Intelligence*, 14(4):353–370, 1995.

[12] E.J. Kontoghiorghes and E. Dinenis. Data parallel QR decompositions of a set of equal size matrices used in SURE model estimation. *Journal of Mathematical Modelling and Scientific Computing*, 6, 1996. (in press).

[13] E.J. Kontoghiorghes and E. Dinenis. Solving triangular seemingly unrelated regression equation models on massively parallel systems. In M. Gilli, editor, *Computational Economic Systems: Models, Methods & Econometrics*, volume 5 of *Advances in Computational Economics*, pages 191–201. Kluwer Academic Publishers, 1996.

[14] C.L. Lawson and R.J. Hanson. *Solving least squares problems*. Prentice–Hall Englewood Cliffs, 1974.

[15] MasPar computer corporation. *MasPar System Overview*, 1992.

[16] S.J. Olszanskyj, J.M. Lebak, and A.W. Bojanczyk. Rank-k modification methods for recursive least squares problems. *Numerical Algorithms*, 7:325–354, 1994.

[17] C.T. Pan and R.J. Plemmons. Least squares modifications with inverse factorizations: parallel implications. *Journal of Computational and Applied Mathematics*, 27:109–127, 1989.

[18] G.A.F. Seber. *Linear regression analysis.* John Wiley and Sons Inc., 1977.

[19] D.M. Smith. *Regression using QR decomposition methods.* PhD Thesis, University of Kent, UK, 1991.