Ordinary linear model estimation on a massively parallel SIMD computer

ERRICOS JOHN KONTOGHIORGES
Institut d’informatique, Université de Neuchâtel, Emile-Argand 11, CH-2007 Neuchâtel, Switzerland
(e-mail: erricos.kontoghiorghes@info.unine.ch)

Dedicated to the late M. R. B. Clarke

SUMMARY
Efficient algorithms for estimating the coefficient parameters of the ordinary linear model on a massively parallel SIMD computer are presented. The numerical stability of the algorithms is ensured by using orthogonal transformations in the form of Householder reflections and Givens plane rotations to compute the complete orthogonal decomposition of the coefficient matrix. Algorithms for reconstructing the orthogonal matrices involved in the decompositions are also designed, implemented and analyzed. The implementation of all algorithms on the targeted SIMD array processor is considered in detail. Timing models for predicting the execution time of the implementations are derived using regression modelling methods. The timing models also provide an insight into how the algorithms interact with the parallel computer. The predetermined factors used in the regression fits are derived from the number of memory layers involved in the factorization process of the matrices. Experimental results show the high accuracy and predictive power of the timing models. Copyright © 1999 John Wiley & Sons, Ltd.

1. INTRODUCTION
A common problem in statistics is that of estimating parameters of some assumed relationship between one or more variables. A specification of individual statistical relationships includes a noise term whose specific value in any single observation cannot be predicted. The simplest form of a statistical relation is called an ordinary linear model (OLM) and has applicability in diverse areas such as signal processing, financial forecasting and econometric modelling\[1–3\]. The OLM model is formally described as

\[ y = Ax + \varepsilon \]  

where \( A \in \mathbb{R}^{m \times n} \) (\( m > n \)) is the exogenous data matrix, \( y \in \mathbb{R}^m \) is the response vector and \( \varepsilon \in \mathbb{R}^m \) is the noise vector with zero mean and dispersion matrix \( \sigma^2 I_m \).

The most frequently used estimating technique for the OLM is least squares. The least squares estimator of the parameter vector \( x \in \mathbb{R}^n \) is determined by minimizing the sum of squares of errors

\[ \underset{x}{\text{argmin}} \, \varepsilon^T \varepsilon = \underset{x}{\text{argmin}} \| Ax - y \|^2 \]  

*Correspondence to: E. J. Kontogiorghes, Institut d’informatique, Université de Neuchâtel, Emile-Argand 11, CH-2007 Neuchâtel, Switzerland.

CCC 1040-3108/99/070323–19$17.50
Received
Copyright © 1999 John Wiley & Sons, Ltd.
Revised 2 June 1998
where \( \| \cdot \| \) denotes the Euclidean norm. If \( A \) does not have full rank, then (2) has an infinite number of solutions. However, a unique minimum 2-norm estimator of \( x \), say \( \hat{x} \), can be computed.

Let the rank of \( A \) be given by \( k ( k \leq n ) \). The solution of (2) is computed in two stages. In the first stage the coefficient matrix \( A \) is reduced to lower trapezoidal form; in the second stage the lower trapezoid is triangularized. The orthogonal decompositions of the first and second stages are given respectively by:

\[
Q^TA \Pi = \begin{pmatrix}
\begin{array}{cc}
0 & 0 \\
L_1 & L_2
\end{array}
\end{pmatrix}
\]

(3)

and

\[
(L_1 L_2) P = \begin{pmatrix}
\begin{array}{cc}
n - k & k \\
0 & L
\end{array}
\end{pmatrix}
\]

(4)

where \( Q \in \mathbb{R}^{m \times m} \) and \( P \in \mathbb{R}^{n \times n} \) are orthogonal, \( L \) and \( L_2 \) are lower-triangular and non-singular, and \( \Pi \in \mathbb{R}^{n \times n} \) is a permutation matrix. That is,

\[
Q^T A \Pi P = \begin{pmatrix}
\begin{array}{cc}
n - k & k \\
k & 0
\end{array}
\end{pmatrix}
\]

(5)

The orthogonal decompositions (3) and (5) are called QL decomposition (QLD) and complete QLD, respectively.

The minimum 2-norm best linear unbiased estimator of \( x \) is given by

\[
\hat{x} = P_2 L^{-1} Q_2^T y
\]

where

\[
\Pi P = \begin{pmatrix}
P_1 & P_2
\end{pmatrix}
\]

and

\[
Q = \begin{pmatrix}
Q_1 & Q_2
\end{pmatrix}
\]

Numerous methods have been proposed for computing the orthogonal factorizations (3) and (4), both on serial computers and MIMD parallel systems[4–7]. In this paper, algorithms are designed, implemented and analyzed for computing the complete QLD on the massively parallel SIMD computer, the CPP DAP 510 (this machine was previously known as AMT DAP 510[8,9]). The algorithms employ Householder reflections and Givens plane rotations. Algorithms are also proposed for reconstructing the orthogonal matrices involved in the decompositions when the data which define the orthogonal transformations are stored in the annihilated parts of the coefficient matrix \( A \). The implementation and execution time models of all algorithms on the CPP DAP 510 (abbreviated here to DAP) are considered in detail. It is expected that the implementation and time estimation techniques presented in this paper can be used for other parallel linear algebra algorithms.
The main reason for choosing a massively parallel SIMD system as the computer architecture to develop the OLM estimation algorithms is that these architectures support applications, such as signal processing and econometrics, where the solution of linear model estimation problems is often required[10]. SIMD architectures are especially designed to perform operations on an array in a data-parallel mode. Hardware and software facilities allow efficient communication between the processing elements. The implementation aspects of matrix computation algorithms are different from those found in MIMD and uni-processor architectures. Therefore, the design and implementation of OLM estimation algorithms that are efficient on a range of SIMD architectures needs to be investigated.

Although the algorithms presented in this paper were implemented on the DAP, the implementation principles should be in general applicable to any massively parallel SIMD computer[11]. The timing models will be of the same order when similar SIMD architectures are used, but without the same coefficient parameters due to the differences in software and hardware designs that exist among different parallel computers. The algorithms were implemented using the DAP Fortran-PLUS enhanced language. This is a Fortran 77 language with array extension of which some of them are similar to those of Fortran 90[12,13]. Thus, the code is not directly portable to other machines.

1.1. Notation

A general notation based on the DAP Fortran-PLUS enhanced and Fortran-90 is described. An \( m \times n \) real matrix \( A \) with elements \( a_{ij} \) (1 \( \leq i \leq m \) and 1 \( \leq j \leq n \)) will be denoted by \( A = [a_{ij}] \in \mathbb{R}^{m \times n} \). Similarly, \( V = [v_i] \in \mathbb{R}^n \) will denote a vector with real elements \( v_i \) (1 \( \leq i \leq m \)). The \( k \)th column and row of \( A \) are given by \( a_{i,k} \) and \( a_{k,i} \), respectively. \( A_{i,k,j:s} \) is a \((k-i+1) \times (s-j+1)\) submatrix of \( A \) with element \( a_{ij} \) in its top left-hand corner and \( V_{i,k} \) is a \((k-i+1)\)-element subvector of \( V \) starting with element \( v_i \). A zero dimension denotes a null matrix or vector and all vectors are considered to be column vectors unless transposed. That is, \( a_{:,k} \) is a column vector and \( a_{k,:} \) is a row vector.

Given a matrix \( A \), \( \text{sumc}(A) \) and \( \text{sumr}(A) \) return a vector obtained from summing the rows and columns of \( A \), respectively. The sum of all of the elements of \( A \) is given by \( \text{sum}(A) \). The two built-in Fortran-PLUS functions \( \text{matc} \) and \( \text{matr} \) are used to create matrices from a vector[12]. If \( V \) is a vector and \( k \) an integer, then \( \text{matc}(V,k) \) creates a matrix of \( k \) columns, each column being \( V \), and \( \text{matr}(V,k) \) creates a matrix of \( k \) rows, each row being \( V \). In Fortran-90 the reduction functions \( \text{sumc}(A) \) and \( \text{sumr}(A) \) are equivalent to \( \text{sum}(A, \text{dim} = 2) \) and \( \text{sum}(A, \text{dim} = 1) \), respectively, while the replication functions \( \text{matc}(V,k) \) and \( \text{matr}(V,k) \) are equivalent to \( \text{spread}(V, \text{dim} = 2, \text{ncopies} = k) \) and \( \text{spread}(V, \text{dim} = 1, \text{ncopies} = k) \).

The application of arithmetic operations and functions on an array are equivalent to the application of the corresponding serial arithmetic operations and functions to the elements of this array. When an operation is applied to an array and a scalar, the scalar is expanded to the mode of the array. All of the algorithms in this paper were implemented on the 1024-processor DAP using double-precision arithmetic. The timing models are defined in ms. The Edge Size of the DAP is denoted by ES.
2. THE QL DECOMPOSITION OF THE COEFFICIENT MATRIX

The computation of the QL decomposition (3) using Householder reflections with column pivoting is considered. This method is also used when \( A \) is of full column rank but ill-conditioned. Let the elementary permutation matrix \( P_{n,i} \) denote the identity \( n \times n \) matrix \( I_n \) with columns \( n - i + 1 \) and \( \mu \) interchanged and

\[
Q_i^T = I_m - \frac{h(i,h(i))^T}{b_i}
\]

(6)
denote an \( m \times m \) Householder matrix which annihilates the first \( m - i \) elements of \( a_{n-i+1} \) (pivot column), when it is multiplied by \( A \) on the right. The matrices \( Q \) and \( \Pi \) in (3) are defined by

\[
Q^n = \prod_{i=1}^{k} Q^n_{k-i+1} = Q^n_{k} Q^n_{k-1} \ldots Q^n_1 \quad \text{and} \quad \Pi = \prod_{i=1}^{k} I^{(i,\mu_i)} = I^{(1,\mu_1)} I^{(2,\mu_2)} \ldots I^{(k,\mu_k)}
\]

To describe briefly the process of computing the QLD (3) let, at the \( i \)-th (0 \( \leq \) \( i \) \( \leq \) \( k \)) step,

\[
A^{(i)} = Q_i^n \ldots Q_1^n A I_{n}^{(1,\mu_1)} \ldots I_{n}^{(i,\mu_i)}
\]

\[
A^{(i)} = m - 1 \begin{pmatrix} L_{11}^{(i)} & 0 \\ L_{21}^{(i)} & L_{22}^{(i)} \end{pmatrix}
\]

where \( L_{22}^{(i)} \) is non-singular and lower-triangular with its diagonal elements in increasing order of magnitude. The value of \( \mu_{i+1} \) is the index of the column of \( L_{11}^{(i)} \) with maximum Euclidean norm. The criterion used to decide that \( \text{rank}(A) = k \) is \( \|a_{1m-k,\mu_{i+1}}\|^2 < \tau \), where \( a_{1m-k,\mu_{i+1}} \) is the \( \mu_{i+1} \) column of \( L_{11}^{(k)} \) and \( \tau \) is an absolute tolerance parameter whose value depends on the scaling of \( A[6,7] \). The value of \( \tau \) is assumed to be given.

The permutation matrix \( \Pi \) can be stored and computed using one of the two \( n \) element integer vectors \( \xi \) and \( \zeta \), where

\[
\Pi = (e_{\xi_1} \ldots e_{\xi_n}) = (e_{\xi_1} \ldots e_{\xi_n})^T
\]

A permutation \( I_{n}^{(i,\mu_i)} \) is equivalent to swapping first the elements \( \zeta_{n-i+1} \) and \( \zeta_{\mu_i} \) of the \( \xi \) vector and then swapping the elements \( n - i + 1 \) and \( \mu_i \) of the \( \zeta \) vector where, initially, \( \xi_i = \zeta_i = i \) (\( i = 1, \ldots, n \)).

2.1. SIMD Implementation

The QL decomposition (3) has been computed on the DAP under the assumption that \( n = \text{NES}, m = \text{MES} \) and \( 1 < m \leq \text{ES}^2 \). That is, the dimension of the matrix \( A \) is an exact multiple of the Edge Size of the array processor and \( \lceil m/\text{ES}^2 \rceil = \lceil M/\text{ES} \rceil = 1 \). A sample of execution times has been generated from the application of a single Householder reflection on matrices with various dimensions. A regression model has been fitted to this sample with the execution time denoting the response variable. The predetermined
1: let $\zeta_i = \xi_i = i$ ($i = 1, \ldots, n$)
2: for $i := 1, 2, \ldots, N$ do
3:  let $m_i = (M - i + 1)ES, n_i = (N - i + 1)ES$ and $\tilde{A} = A_{1:m_i,1:n_i}$
4:  $V_{1:n_i} := \sum_{k}(A \ast \tilde{A})$
5: for $j := 1, 2, \ldots, ES$ do
6:  $\mu_j := \max(V_1, \ldots, V_{n_i-j+1})$
7: if $V_{\mu_j} < \tau$ then
8:  STOP
9: else
10:  column_swap($A, V, \xi, \zeta, n_i - j + 1, \mu_j$)
11:  $h := 0$
12:  $h_{1:m_i-j+1} := A_{1:m_i-j+1, n_i-j+1}$
13:  $s := \sqrt{V_{n_i-j+1}}$
14: if $h_{m_i-j+1} < 0$ then $s := -s$
15:  $h_{m_i-j+1} := h_{m_i-j+1} + s$
16:  $V_{n_i-j+1} := h_{m_i-j+1}$
17:  $b := s \ast h_{m_i-j+1}$
18:  $\tilde{A}_{m_i-j+1, n_i-j+1} := -s$
19:  $W = \text{matr}(h, n_i - j)$
20:  $X = \sum_{k}(W \ast A_{1:m_i-j+1,1:n_i-j})/b$
21:  $\tilde{A}_{1:n_i-j} := \tilde{A}_{1:n_i-j} - W \ast \text{matr}(X, m_i)$
22:  $V_{1:n_i-j} := V_{1:n_i-j} - (\tilde{A}_{m_i-j+1, n_i-j})^2$
23: end if
24: end for
25: end for

**Algorithm 1. The QL decomposition of $A$**

Factors of the timing model are derived from the number of ES $\times$ ES layers involved in the arithmetic computations and the number of times layers are replicated or reduced. The time required to construct and apply the $i$th Householder reflection is found to be

$$T_i(M, N) = a_0 + a_1[m/ES^2] + a_2[m/ES] + a_3[m/ES][n/ES]$$
$$= a_1' + a_2M + a_3MN$$

(7)

where $a_1' = a_0 + a_1 = 2.43, a_2 = 0.014$ and $a_3 = 1.683$.

Notice that $Q^T_kA$ affects only the leading $(m - i + 1) \times (n - i + 1)$ submatrix of $A$. In order to reduce the execution time, the Householder reflections are divided into the $K$ sets $S^{(1)}, \ldots, S^{(K)}$, where $K = [k/ES]$. The set $S^{(i)}, i = 1, \ldots, K$, consists of the reflections $S^{(i)}_1, \ldots, S^{(i)}_{t_i}$, where $S^{(i)}_j, j = 1, \ldots, t_i$, is equivalent to $Q^{T_{(i-1)ES+j}}_{t_i} = k - (K - 1)ES$ and $t_1 = \cdots = t_{K-1} = ES$. If $m_i = (M - i + 1)ES$ and $n_i = (N - i + 1)ES$, then the Householder reflection $S^{(i)}$ is applied only to the submatrix $\tilde{A}^{(i)} = A_{1:m_i,1:n_i}$. The Householder reflections $S^{(i)}_1, \ldots, S^{(i)}_{t_i}$ use the same number of ES $\times$ ES layers of memory and thus they have the same execution time when applied on the left of $\tilde{A}^{(i)}$.

Algorithm 1 effects the computation of the QL decomposition (3). The procedure *column_swap* performs the appropriate column and element interchanges on its matrix and vector arguments if $n_i - j + 1 \neq \mu_j$. Initially, the squared Euclidean norms of the columns
Table 1. Computing the QLD \((3)\) in seconds, where \(m = MES\) and \(n = NES\)

<table>
<thead>
<tr>
<th>Matrix dimension</th>
<th>Execution time of Algorithm 1</th>
<th>Predictions (\times 10^3)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>(m)</td>
<td>(n)</td>
<td>(T_{QL}(M, N))</td>
<td>(T_{2}(M, N))</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>0.22</td>
<td>0.22</td>
</tr>
<tr>
<td>128</td>
<td>32</td>
<td>0.34</td>
<td>0.34</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
<td>0.58</td>
<td>0.56</td>
</tr>
<tr>
<td>256</td>
<td>224</td>
<td>1.56</td>
<td>1.56</td>
</tr>
<tr>
<td>256</td>
<td>224</td>
<td>10.56</td>
<td>10.56</td>
</tr>
<tr>
<td>288</td>
<td>32</td>
<td>0.64</td>
<td>0.64</td>
</tr>
<tr>
<td>288</td>
<td>128</td>
<td>5.08</td>
<td>5.08</td>
</tr>
<tr>
<td>288</td>
<td>256</td>
<td>10.56</td>
<td>10.56</td>
</tr>
<tr>
<td>320</td>
<td>32</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>320</td>
<td>96</td>
<td>3.58</td>
<td>3.58</td>
</tr>
<tr>
<td>320</td>
<td>288</td>
<td>20.12</td>
<td>20.12</td>
</tr>
<tr>
<td>384</td>
<td>32</td>
<td>0.82</td>
<td>0.82</td>
</tr>
<tr>
<td>384</td>
<td>96</td>
<td>4.27</td>
<td>4.27</td>
</tr>
<tr>
<td>384</td>
<td>352</td>
<td>34.24</td>
<td>34.23</td>
</tr>
<tr>
<td>480</td>
<td>32</td>
<td>0.99</td>
<td>1.00</td>
</tr>
<tr>
<td>480</td>
<td>96</td>
<td>5.31</td>
<td>5.31</td>
</tr>
<tr>
<td>480</td>
<td>288</td>
<td>32.82</td>
<td>32.82</td>
</tr>
<tr>
<td>480</td>
<td>352</td>
<td>45.38</td>
<td>45.38</td>
</tr>
<tr>
<td>480</td>
<td>416</td>
<td>58.90</td>
<td>58.90</td>
</tr>
</tbody>
</table>

of \(\tilde{A}^{(i)}\) (denoted by \(\tilde{A}\)) are stored in \(V_{1:n_{i}}\). For greater accuracy \(V_{1:n_{i}}\) is recomputed prior to the applications of the Householder reflections in \(S^{(i)}\). The Householder vectors are stored in the annihilated positions of \(A\) and in the last \(k\) elements of \(V\). Within the context of OLM estimation, Algorithm 1 can be applied to the augmented matrix \((yA)\), except that the permutations are performed only on the columns of \(A\).

The total time spent in applying the Householder reflections is \(\sum_{i=1}^{K} t_i T_i(m_i, n_i)\). If \(A\) has full column rank, i.e. \(k = n\) and \(t_i = ES\), \(i = 1, \ldots, N\), then the total time is evaluated as

\[
T_2(M, N) = ES N \left(\frac{6a_1 + 3a_2(2M - N + 1) + a_3(3MN + 3M - N^2 + 1)}{6}\right)
\]

In this case, the estimated execution time of Algorithm 1 on the DP is found to be

\[
T_{QL}(M, N) = N(105.92 + 31.79M + 5.86N + 27.81MN - 9.18N^2)
\]

This estimation is derived using the backward stepwise regression method, where the explained variable is the execution time and the initial explanatory variables are determined after evaluating \(T_2(M, N)\)[3,11]. Table 1 shows the high accuracy of the timing models \(T_2(M, N)\) and \(T_{QL}(M, N)\) when used to predict the execution time of Algorithm 1. Notice that the time spent in applying the Householder reflections is approximately 90% of the total time of Algorithm 1, i.e. \(T_2(M, N)/T_{QL}(M, N) \approx 0.90\).

3. TRIANGULARIZING THE LOWER TRAPEZOIDAL

The computation of the factorization \((4)\) can be considered as being similar to the block updating of the QL decomposition[14]. Two methods are presented. The first uses
Householder reflections and annihilates one row at a time of the $L_1$ matrix; the second uses Givens rotations for simultaneously annihilating elements in various rows of $L_1$.

3.1. The Householder method

A Householder algorithm is considered for computing the factorization (4). For notational convenience let $\tilde{n} \equiv n - k$, $L_1 \equiv \tilde{L} = [\tilde{l}_{ij}]$ and $L_2 \equiv \tilde{L} = [\tilde{l}_{ij}]$. The $i$th Householder reflection has the form

$$ P_i = I_n - \frac{1}{c_i} \left( \tilde{l}_{i,:} \tilde{l}_{i,:}^T \right) $$

where $c_i = \tilde{l}_{ii} \pm s$, $s^2 = \|\tilde{l}_{ii}\|^2 + \tilde{l}_{ii}^2$, $1 \leq i \leq k$, and $I_k = (e_1 \ldots e_k)$. The application of $P_i$ on the right of $(\tilde{L}, \tilde{L})$ gives

$$(\tilde{L}, \tilde{L}) P_i = (\tilde{L} - x \tilde{l}_{i,:}^T \tilde{L} - \gamma_i x e_i^T), \quad \text{where} \quad x = (\tilde{l}_{ii} + \gamma_i \tilde{l}_{i,:}) / c_i $$

The reflection (9) annihilates the $i$th row of $\tilde{L}$ by modifying all of $\tilde{L}$ but only the $i$th column of $\tilde{L}$. After the reflections $P_1, \ldots, P_k$, the first $i$ rows of $\tilde{L}$ are zero and $\tilde{L}$ remains lower-triangular. The orthogonal matrix $P$ in (4) is defined as $P = P_1 P_2 \ldots P_k$. Figure 1 shows the annihilation pattern for $\tilde{L}$ when using Householder reflections.

The Householder reflections are divided into sets $S^{(1)} \ldots S^{(K)}$, where $K = [k/ES]$. The set $S^{(i)}$, $i = 1, \ldots, K$, comprises the reflections $S^{(1)}_1, \ldots, S^{(1)}_{t_1}$, $S^{(2)}_1, \ldots, S^{(2)}_{t_2}$, $\ldots$, $S^{(K)}_1, \ldots, S^{(K)}_{t_K}$, where $t_1 = \delta = k - (K - 1) ES$ and $t_2 = \ldots = t_k = ES$. The reflection $S^{(i)}_j$, $j = 1, \ldots, t_i$, is equivalent to $P_{j+t_1(i)}$, where $t_1(i) = \sum_{q=1}^{i-1} t_q$.

The reflection $S^{(i)}_j$ is applied to the $k_i \times \tilde{n}$ submatrix $\tilde{L}^{(i)} = L_{t_1(i)+1:k,1:k_i}$ and $k_i$-element subvector $\tilde{l}_{(i)} = \tilde{l}_{i,1:k_i}$, where $k_j = k_i - t^{(i)}$. Algorithm 2 effects the computation of the orthogonal factorization (4) using Householder reflections, where the function sign returns the sign of its argument, $\tilde{l}^* = [\tilde{l}_{ij}^*] \in \mathcal{R}^{k_i \times \tilde{n}}$, $\tilde{L}^* = [\tilde{l}_{ij}^*] \in \mathcal{R}^{k_i \times \tilde{n}}$ and $\tilde{y} = [\tilde{y}_j] \in \mathcal{R}^{k_i}$. The vectors $\tilde{l}_{ij}$ and scalars $\gamma_i$ of the Householder reflections $P_1, \ldots, P_k$ are stored in the annihilated positions of $\tilde{L}$ and in the $k_i$-element array $\gamma$.

The time required to apply $S^{(i)}_j$ is found to be

$$ T(k_i, \eta) = a_0 + a_1 \eta + a_2 [k_i/ES] + a_3 \eta [k_i/ES] $$

Figure 1. Annihilation pattern of (4) using Householder reflections

1: let \( \tilde{n} = n - k, K = [k/ES], t_1 = k - (K - 1)ES \) and \( t_2 = \cdots = t_k = ES \)
2: for \( i = 1, 2, \ldots, K \) do
3: let \( t^{(i)} = \sum_{q=1}^{i-1} t_q, k_i = k - t^{(i)}, \hat{L}^* = \hat{L}_{t^{(i)}+1:k,1:}\tilde{n} \),
   \( \hat{L}^* = \hat{L}_{t^{(i)}+1:k,t^{(i)}+1:t^{(i+1)}} \) and \( \tilde{y} = y_{1:t^{(i)}+1:t^{(i+1)}} \)
4: for \( j = 1, 2, \ldots, t_i \) do
5: \( s := \text{sign}(\hat{t}_{ij}^*) \sqrt{\sum(\hat{t}_{j,\cdot}^* \cdot \hat{t}_{ij}^*) + (\hat{t}_{ij}^*)^2} \)
6: \( \tilde{y}_j := \hat{t}_{ij}^* + s \)
7: \( c := \tilde{y}_j * s \)
8: \( x := (\text{sumc}(\hat{L}^* * \text{matr}(\hat{t}_{j,\cdot}^*, k_i)) + \tilde{y}_j * \hat{t}_{ij}^*) / c \)
9: \( \hat{L}_{j+1:k,1:}\tilde{n}^* := \hat{L}_{j+1:k,1:}\tilde{n}^* - \text{matc}(x_{j+1:k}, \tilde{n}) * \text{matr}(\hat{t}_{j,\cdot}^*, k_i - j) \)
10: \( \hat{t}_{ij}^* := \hat{t}_{j+1:k,j} - \tilde{y}_j * x_{j+1:k} \)
11: \( \hat{t}_{ij}^* := -s \)
12: end for
13: end for

Algorithm 2. Triangularizing the lower trapezoidal using Householder reflections

where \( \eta = [\tilde{n}/ES], a_0 = 1.57, a_1 = 0.06, a_2 = 0.22 \) and \( a_3 = 0.54 \). Thus, the total time spent in applying the Householder reflections in Algorithm 2 is given by

\[
T_{CQL}(K, \eta) = \sum_{i=1}^{K} t_i T(k_i, \eta)
\]

\[
= \delta(a_0 + a_1 \eta + a_2 K + a_3 \eta K) + ES \frac{(K - 1)}{2} (2a_0 + 2a_1 \eta + a_2 K + a_3 \eta K)
\]

The timing model \( T_{CQL}(K, \eta) \) can be used to predict the total execution time of Algorithm 2, which has negligible organizational overheads.

Notice that, if \( t_1 = \cdots = t_{K-1} = ES, t_K = \delta, K > 1 \) and \( \delta \neq ES \), then the execution time of Algorithm 2 will increase by \( (ES - \delta)(K - 1)(a_2 + a_3 \eta) \) ms. Observe also that, if \( D = (\hat{L}^* \hat{t}^*) \) and \( f^T = (\hat{t}^*_{\cdot,j})^T \tilde{y}_j \), then lines 8–10 of Algorithm 2 may be expressed as

\[
x := \text{sumc}(D * \text{matr}(f, k_i)) / c
\]

\[
D_{j+1:k,1:}\tilde{n} + 1 := D_{j+1:k,1:}\tilde{n} + 1 - \text{matc}(x_{j+1:k}, \tilde{n} + 1) * \text{matr}(f, k_i - j)
\]

This formulation uses fewer \( ES \times ES \) layers of memory than Algorithm 2 when \( \eta = [\tilde{n}/ES] \). However, on the DAP, the slight improvement in speed of using fewer memory layers is lost from the overheads incurred in setting \( \hat{t}_{\cdot,j}^* \) in \( D \) and then storing it in \( \hat{L} \).

3.2. The Givens method

A parallel Givens sequence (hereafter called PGS) requiring \( n - 1 \) compound disjoint Givens rotations (CDGRs) to compute the orthogonal decomposition (4) is proposed. A
CDGR is the product of disjoint Givens rotations that simultaneously annihilate elements of a matrix\[14–16\]. A single Givens rotation that annihilates a specific element affects two rows (columns) of a matrix when it is applied on the left (right)\[6\]. Therefore, at most \(\lfloor n/2 \rfloor\) disjoint Givens rotations can be applied simultaneously on the right of a matrix which has \(n\) columns\[15,17,18\]. The PGS does not create any non-zero elements above the main diagonal of \(L\), and previously created zeros of \(L\) are preserved. Let the Givens rotation \(G_{i,j} \in \mathbb{R}^{n \times n}\) be defined as

\[
G_{i,j} = \begin{pmatrix}
1 & & & & \\
& \ddots & & & \\
& & c & s & \\
& & & 1 & \\
& & & & \ddots \\
\end{pmatrix}
\]

The application of \(G_{i,j}\) on the right of \((\bar{L}, \bar{L})\) annihilates \(\bar{I}_{ij}\). At most \(\min(\bar{n}, k)\) disjoint Givens rotations of the form (10) can be applied simultaneously: here it is assumed that \(\bar{n} \leq k\). The elements annihilated after the application of the \(i\)th CDGR, denoted by \(G^{(i)}\), lie on a diagonal, and their total number is given by

\[
e_i = \begin{cases} 
  i & \text{if } 1 \leq i < \bar{n} \\
  \bar{n} & \text{if } \bar{n} \leq i \leq k \\
  \bar{n} + k - i & \text{if } k \leq i < \bar{n} + k 
\end{cases}
\]

Table 2 shows the Givens rotations comprising \(G^{(i)}\) of the PGS, the diagonals and the total number of elements annihilated after the application of \(G^{(i)}\), where \(d_{i,j}\) denotes the diagonal of \(\bar{L}\) starting at element \(\bar{I}_{ij}\). In Figure 2 \(i\), \(1 \leq i < n\), corresponds to the elements of \(\bar{L}\) annihilated from the applications of \(G^{(i)}\), where \(k = 16\) and \(n = 22\) (\(\bar{n} = 6\)). Figure 3 illustrates an alternative annihilation scheme equivalent to the PGS. Modified versions of the PGS have been efficiently employed to reorthogonalize the QR decomposition after deleting columns and have also been used in the solution of the modified OLM, generalized linear model and simultaneous equation models\[14,19,20\].

The application of \(G^{(i)}\) affects \(e_i\) consecutive columns of each \(\bar{L}\) and \(\bar{L}\). Let the submatrices comprising these columns be denoted by \(\bar{L}^{(i)}\) and \(\bar{L}^{(i)}\), respectively. Furthermore, let \(x = (x_1, \ldots, x_{e_i})\) denote the elements of \(\bar{L}^{(i)}\) annihilated after the application of \(G^{(i)}\), and the corresponding elements of \(\bar{L}^{(i)}\) used in constructing the rotations be given by \(y = (y_1, \ldots, y_{e_i})\). The application of \(G^{(i)}, i = 1, \ldots, n - 1,\) on
Table 2. The CDGRs of the PGS for computing the factorization (4)

<table>
<thead>
<tr>
<th>i</th>
<th>( G^{(i)} = )</th>
<th>Diagonals annihilated</th>
<th>( e_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( G_{1,\hat{n}} )</td>
<td>( d_{1,\hat{n}} )</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>( G_{1,\hat{n}-1}G_{2,\hat{n}} )</td>
<td>( d_{1,\hat{n}-1} )</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>( G_{1,\hat{n}-2}G_{2,\hat{n}-1}G_{3,\hat{n}} )</td>
<td>( d_{1,\hat{n}-2} )</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>:</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>( \hat{n} )</td>
<td>( G_{1,1}G_{2,2} \cdots G_{\hat{n},\hat{n}} )</td>
<td>( d_{1,1} )</td>
<td>( \hat{n} )</td>
</tr>
<tr>
<td>( \hat{n} + 1 )</td>
<td>( G_{2,1}G_{3,2} \cdots G_{\hat{n}+1,\hat{n}} )</td>
<td>( d_{2,1} )</td>
<td>( \hat{n} )</td>
</tr>
<tr>
<td>...</td>
<td>:</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>( k )</td>
<td>( G_{k-\hat{n}+1,1}G_{k-\hat{n}+2,1} \cdots G_{k,\hat{n}} )</td>
<td>( d_{k-\hat{n}+1,1} )</td>
<td>( \hat{n} )</td>
</tr>
<tr>
<td>( k + 1 )</td>
<td>( G_{k-\hat{n}+2,1}G_{k-\hat{n}+3,2} \cdots G_{k,\hat{n}-1} )</td>
<td>( d_{k-\hat{n}+2,1} )</td>
<td>( \hat{n} - 1 )</td>
</tr>
<tr>
<td>...</td>
<td>:</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td>( k + \hat{n} - 2 )</td>
<td>( G_{k-1,1}G_{k,2} )</td>
<td>( d_{k-1,1} )</td>
<td>2</td>
</tr>
<tr>
<td>( k + \hat{n} - 1 )</td>
<td>( G_{k,1} )</td>
<td>( d_{k,1} )</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2. PGS

Figure 3. Alternative PGS

the right of \((\hat{L} \hat{L})\) can be written as

\[
\left( \begin{array}{cccc}
    c_1 & \cdots & s_1 & \\
    \cdots & \ddots & \cdots & \\
    -s_1 & \cdots & c_1 & \cdots \\
    \cdots & \cdots & \cdots & c_{e_f}
\end{array} \right)
\]

\[
:= (\hat{L}^{(i)} \ast C - \hat{L}^{(i)} \ast S) \hat{L}^{(i)} \ast S + \hat{L}^{(i)} \ast C
\]  \hspace{1cm} (11)

where \( s = x/t, c = x/t, t = \sqrt{(x^2 + y^2)} \),

\[
C = \begin{pmatrix}
    c_1 & c_2 & \cdots & c_{e_i} \\
    c_1 & c_2 & \cdots & c_{e_i} \\
    \vdots & \vdots & \ddots & \vdots \\
    c_1 & c_2 & \cdots & c_{e_i} \\
\end{pmatrix}
\]

and \( S = \text{matr}(s, k) = \begin{pmatrix}
    s_1 & s_1 & \cdots & s_{e_i} \\
    s_1 & s_2 & \cdots & s_{e_i} \\
    \vdots & \vdots & \ddots & \vdots \\
    s_1 & s_2 & \cdots & s_{e_i} \\
\end{pmatrix} \)

In the implementation of the PGS on the DAP, let \( \tilde{n} = \eta \text{ES} \) and \( k = K\text{ES} \). The application of the CDGRs is divided into the three phases \( \Phi_1, \Phi_2 \) and \( \Phi_3 \). Phase \( \Phi_1 \) applies the CDGRs in \( S^{(i)}, i = 1, 2, 3 \), where

\[
S^{(i)}_j = \begin{cases}
    G^{(i)}(j) & \text{if } i = 1 \text{ and } j = 1, \ldots, \tilde{n} - 1 \\
    G^{(i)}(\tilde{n} + j - 1) & \text{if } i = 2 \text{ and } j = 1, \ldots, k - \tilde{n} \\
    G^{(i)}(k + j - 1) & \text{if } i = 3 \text{ and } j = 1, \ldots, \tilde{n} 
\end{cases}
\]

Phase \( \Phi_2 \) is divided into the \( K - \eta \) sub-phases \( \varphi_1, \ldots, \varphi_{K-\eta} \), where, at sub-phase \( \varphi_i \), the diagonals \( d_{1,1}, d_{2,1}, \ldots, d_{\text{ES},1} \) of the \( (K + 1 - i) \text{ES} \times \eta \text{ES} \) submatrix \( \tilde{L}_{(i-1)\text{ES}+1,k,1,\tilde{n}} \) are annihilated. In phases \( \Phi_2 \) and \( \Phi_3 \) previously annihilated \( \text{ES} \times \text{ES} \) submatrices of \( \tilde{L} \) and \( \tilde{L} \) are excluded from the computations. Figure 4 shows the phases and the sub-phases of PGS, where \( \text{ES} = 4 \).

A timing model for the PGS needs to be derived in order to compare the performance of the Givens and Householder algorithms on the DAP. This model can be obtained by adding the (parametrized) estimated execution times of the different phases of the algorithm. Then the factors of the total execution time model, say \( T_{\text{ROT}}(K, \eta) \), are used in the stepwise regression to construct a timing model of the Given’s algorithm, say \( T_{\text{PGS}}(K, \eta) \), which also takes into account the various organizational aspects of the implementations that do not occur in the different phases. Furthermore, the analysis and comparison of \( T_{\text{ROT}}(K, \eta) \) and \( T_{\text{PGS}}(K, \eta) \) will indicate whether any inefficiencies occur during the implementation of the algorithm.

From experiments, the estimated execution time for constructing and applying the CDGR \( G^{(i)} \) in (11) is found to be

\[
\tilde{T}(K, e_i) = 1.5 + K [e_i / \text{ES}] (0.887 + 0.004K - 0.006[e_i / \text{ES}]), \quad i = 1, \ldots, n - 1
\]

Thus, the time required to apply all the CDGRs of the PGS is calculated as

\[
T_{\text{ROT}}(K, \eta) = \text{ES} \sum_{i=1}^{\eta} \tilde{T}(K, i) - T(K, \eta) \quad \text{(from phase } \Phi_1) \\
\quad + \text{ES} \sum_{i=1}^{K-\eta} \tilde{T}(K + 1 - i, \eta) \quad \text{(from phase } \Phi_2) \\
\quad + \text{ES} \sum_{i=1}^{\eta} \tilde{T}(\eta + 1 - i, \eta + 1 - i) \quad \text{(from phase } \Phi_3)
\]
At the beginning of sub-phase $\varphi_3$, the top $8 \times 8$ (shadow frame) is zero and is excluded from the computation (11).

Figure 4. Illustration of the implementation phases of the PGS, where $ES = 4$

### Table 3. Computing the factorization (4) (times are in seconds), where $k = KES$ and $n - k = \eta ES$

<table>
<thead>
<tr>
<th>Matrix dimension</th>
<th>Execution time of Algorithm 2</th>
<th>Predictions $\times 10^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>$n - k$</td>
<td>$T_{CQL}(K, \eta)$</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>0.08</td>
</tr>
<tr>
<td>192</td>
<td>32</td>
<td>0.83</td>
</tr>
<tr>
<td>192</td>
<td>96</td>
<td>1.58</td>
</tr>
<tr>
<td>192</td>
<td>160</td>
<td>2.34</td>
</tr>
<tr>
<td>352</td>
<td>32</td>
<td>2.20</td>
</tr>
<tr>
<td>352</td>
<td>96</td>
<td>4.55</td>
</tr>
<tr>
<td>352</td>
<td>288</td>
<td>11.59</td>
</tr>
<tr>
<td>352</td>
<td>352</td>
<td>13.94</td>
</tr>
<tr>
<td>512</td>
<td>32</td>
<td>4.21</td>
</tr>
<tr>
<td>512</td>
<td>96</td>
<td>9.06</td>
</tr>
<tr>
<td>512</td>
<td>288</td>
<td>23.44</td>
</tr>
<tr>
<td>512</td>
<td>416</td>
<td>33.09</td>
</tr>
<tr>
<td>512</td>
<td>480</td>
<td>37.92</td>
</tr>
</tbody>
</table>

After expansion and using backward stepwise regression, the execution time model of the PGS (including the overheads) on the DAP is given by

$$T_{PGS}(K, \eta) = K(a_0 + a_1 \eta + a_2 K + a_3 \eta K + a_4 \eta^2) + \eta(a_5 - a_6 \eta^2)$$

where $a_0 = 35.70$, $a_1 = 38.77$, $a_2 = 1.10$, $a_3 = 17.82$, $a_4 = 15.07$, $a_5 = 65.93$ and $a_6 = 2.92$. From Table 3 it is obvious that the Householder reflections method is superior.
to the Givens rotation method. Notice that, to two significant digits, $T_{PGS}(K, \eta)$ and the execution time of PGS on the DAP are the same.

4. RECONSTRUCTING THE ORTHOGONAL MATRICES

In some cases, such as in the deletion of data from the OLM and in recursive constrained OLM, the orthogonal matrices are required to be formed\[14,21\]. The algorithms used in computing the orthogonal factorizations (3) and (4) hold the information necessary for regenerating the orthogonal matrices. Algorithms for the explicit formation of the orthogonal matrices $Q$ and $P$ are given for the case where the factorizations (3) and (4) are computed using Householder reflections.

4.1. Reconstructing the orthogonal matrix $Q$

The reconstruction of the orthogonal matrix $Q^T$ in (3) can be considered to be similar to that of Algorithm 1 when the Householder vectors $h^{(i)}$, $i = 1, \ldots, k$, in (6) are known. After execution of Algorithm 1 the value of $b_i$, $i = 1, \ldots, k$, in (6) can be computed as $b_i = -V_{n-i+1}A_{m-i+1,n-i+1}$. However, executing Algorithm 2 after Algorithm 1 results in the entries of $A_{m-i+1,n-i+1}$ to be overwritten by $A_{m-i+1,n-i+1} - y_{k+i-1}$. Thus, $b_i$ is computed as $-V_{n-i+1}(A_{m-i+1,n-i+1} + y_{k+i-1})$, where $y_i, i = 1, \ldots, k$, is defined in (9).

For simplicity let $H = (h_1, h_2, \ldots, h_k)$ and $\beta = (\beta_1, \beta_2, \ldots, \beta_k)$, where $h_i$ and $\beta_i$ correspond to the Householder vector $h^{(k+i)}$ and $b_{k+i-1}$ in (6). Algorithm 3 defines the steps in reconstructing the orthogonal matrix $Q^T$.

The execution time of Algorithm 3 is given by

$$T_Q(M, k) = \sum_{i=1}^{K} t_i (a_0 + a_1 M + a_2 M(M + 1 - i))$$

$$= ES\left(\frac{K-1}{2}\right)(2a_0 + 2a_1 M + a_2 M(2M + 2 - K))$$

$$+ \delta(a_0 + a_1 M + a_2 M(M + 1 - K))$$

where $m = MES$, $K = \lceil k/ES \rceil$, $a_0 = 0.45$, $a_1 = 0.19$ and $a_2 = 0.55$.

4.2. Reconstructing the orthogonal matrix $P$

The data defining the $i$th Householder matrix $P_i$, $i = 1, \ldots, k$, occupy the $i$th row of $\hat{L} \equiv L_1$, the $i$th position of the $k$-element array $\gamma$ and the $i$th diagonal element of $L = A_{m-k+1,m,n-k+1,n}$. The value of $c_i$ in (8) can be computed as $-\gamma_i A_{m-k+i-1,n-k+i-1}$. It can be proved that the matrix $P^{(\lambda)} = P_1 \ldots P_k$ has a special structure which facilitates the development of an efficient algorithm for computing $P = P^{(\lambda)}$.

**Theorem 1**

The structure of the matrix $P^{(\lambda)} = P_1 \ldots P_k$ is given by

$$P^{(\lambda)} = \tilde{\mu} + \lambda \begin{pmatrix} D^{(\lambda)} & W^{(\lambda)} & 0 \\ 0 & 0 & I \end{pmatrix}$$

(12)
1: let \( K = \lceil k/ES \rceil \) and \( \delta = k - (K - 1)ES \)
2: \( Q^T := I_m \)
3: let \( t_1 = \ldots = t_{K-1} = ES \) and \( t_K = \delta \)
4: for \( i = 1, \ldots, K \) do
5: let \( m_i = (M - i + 1)ES, n_i = (N - i + 1)ES, k_i = k + 1 - \sum_{j=1}^{i} t_j \),
6: \( \tilde{Q}^T = Q^T_{1:m_i, 1:m}, \tilde{H} = H_{1:m_i, k_i:j_i+\delta} \) and \( \tilde{\beta} = \beta_{k_i:j_i+\delta} \)
7: for \( j := t_i, t_i - 1, \ldots, 1 \) do
8: \( W = \text{matrix}(\tilde{h}::, m) \)
9: \( \tilde{Q}^T = \tilde{Q}^T - \text{matc(sumc}(\tilde{Q}^T \ast \tilde{W}j, m_i) \ast W \)
10: end for
end for

Algorithm 3. The reconstruction of the orthogonal matrix \( Q \) in (3)

where the bottom \( \lambda \times \lambda \) submatrix of \( W^{(\lambda)} \) is lower-triangular.

Proof

Base step: For \( \lambda = 1 \) the matrix \( P^{(\lambda)} \) is given by

\[
P^{(1)} = P_1 = I_n - \frac{1}{c_1} \left( \tilde{I}_{1:} \gamma_1 e_1 \right) \left( \tilde{I}_{1:}^T \gamma_1 e_1^T \right) = \begin{pmatrix}
I_n - \frac{\tilde{I}_{1:}}{c_1} & -\frac{\gamma_1}{c_1} \tilde{I}_{1:}
\frac{\gamma_1}{c_1} \tilde{I}_{1:}^T & 0
\end{pmatrix}
\]

which satisfies (12) with

\[
D^{(1)} = \begin{pmatrix}
I_n - \frac{\tilde{I}_{1:}}{c_1} & -\frac{\gamma_1}{c_1} \tilde{I}_{1:}
\frac{\gamma_1}{c_1} \tilde{I}_{1:}^T & 0
\end{pmatrix}
\quad \text{and} \quad
W^{(1)} = \begin{pmatrix}
-\frac{\gamma_1}{c_1} \tilde{I}_{1:}
1 - \frac{\gamma_1^2}{c_1}
\end{pmatrix}
\]

Inductive step: Assume that \( P^{(\lambda)} \) has the structure defined in (12). It must be shown that \( P^{(\lambda+1)} \) also has this structure.

Now, \( P^{(\lambda+1)} = P^{(\lambda)} P_{\lambda+1} \), which is computed as

\[
P^{(\lambda+1)} = P^{(\lambda)} P_{\lambda+1} = \begin{pmatrix}
D^{(\lambda)} & W^{(\lambda)} & 0 & 0
0 & 0 & 1 & 0
0 & 0 & 0 & I_{k-\lambda-1}
\end{pmatrix} - \frac{1}{c_{\lambda+1}} \begin{pmatrix}
D^{(\lambda)} & W^{(\lambda)} & 0 & 0
0 & 0 & 1 & 0
0 & 0 & 0 & I_{k-\lambda-1}
\end{pmatrix}
\times \begin{pmatrix}
\tilde{I}_{\lambda+1:} & \gamma_{\lambda+1} e_{\lambda+1} \gamma_{\lambda+1} e_{\lambda+1}^T
\end{pmatrix}
\]
Thus, given $P_\theta$, the calculations of $D^{(k)}$ and $W^{(k)}$ can be summarized as follows:

$$
\begin{align*}
D^{(k+1)} &= \begin{pmatrix}
D^{(k)} & D^{(k)} \tilde{y}_{\lambda+1}^T \\
\gamma_{\lambda+1} & \gamma_{\lambda+1}
\end{pmatrix} \quad \text{and} \quad W^{(k+1)} = \begin{pmatrix}
W^{(k)} & W^{(k)} \tilde{y}_{\lambda+1} \\
0 & 1 - \frac{\gamma_{\lambda+1}^2}{\gamma_{\lambda+1}}
\end{pmatrix}
\end{align*}
$$

Notice that, given $W^{(k)}$, $W^{(k+1)}$ can be derived by computing only its $i$th column $w_{i,\lambda+1}$. Thus, given $P^{(k)}$, the calculations of $D^{(k+1)}$ and $w_{i,\lambda+1}$ can be summarized as follows:

$$
D^{(k+1)} = \begin{pmatrix}
D^{(k)} & D^{(k)} \tilde{y}_{\lambda+1}^T \\
\gamma_{\lambda+1} & \gamma_{\lambda+1}
\end{pmatrix} \quad \text{and} \quad W^{(k+1)} = \begin{pmatrix}
W^{(k)} & W^{(k)} \tilde{y}_{\lambda+1} \\
0 & 1 - \frac{\gamma_{\lambda+1}^2}{\gamma_{\lambda+1}}
\end{pmatrix}
$$

where $e_{n+\lambda+1}$ is the last column of the identity matrix $I_{n+\lambda+1}$. Algorithm 4 shows, in data-parallel mode, the reconstruction of the orthogonal matrix $P$.

On the DAP the implementation of Algorithm 4 comprises $\mu$ phases $\Phi_1, \ldots, \Phi_\mu$, where $\mu = \lceil n/\text{ES} \rceil - \eta + 1$ and $\eta = \lceil (\tilde{n} + 1)/\text{ES} \rceil$. During $\Phi_1, \ldots, \Phi_{\mu-1}$ the affected submatrix of $P_{1, \tilde{n}, \tilde{n}}$ is a multiple of ES. Phase $\Phi_i$, $i = 1, \ldots, \mu$, has $s_i$ steps, with the $j$th step being equivalent to step $(j + \sum_{q=1}^{i-1} s_q)$ of Algorithm 4, where

$$
s_i = \begin{cases}
\delta = \eta \text{ES} - \tilde{n} & \text{if } i = 1 \\
\text{ES} & \text{if } i = 2, \ldots, \mu - 1 \\
k - \delta - (\mu - 2) \text{ES} & \text{if } i = \mu \text{ and } \mu > 2
\end{cases}
$$
In Figure 5 a grid and a shaded box denote the submatrices \( P_{1,x_1,1,n_1} \) and \( P_{1,x_1+s_1,1,n_1} \), \( i = 1, \ldots, \mu \), where \( k = 17, n_1 = 6 \) and \( ES = 4 \).

The execution time of the \( i \)-th step of Algorithm 4 is found to be \( a_0 + \lceil (n + i)/ES \rceil (a_1 + a_2 [n/ES]) \), where \( a = 0.78, a_1 = 0.21 \) and \( a_3 = 0.55 \). Thus, for \( n_i = n + \sum_{j=1}^{i} s_j \), the estimated time of implementing Algorithm 4 on the DAP, excluding the overheads, is given by

\[
T_P(k, \hat{n}) = \sum_{i=1}^{\mu} s_i (a_0 + (a_1 + a_2 [\hat{n}/ES]) [n_i/ES])
\]

\[
= \sum_{i=1}^{\mu} s_i (a_0 + (a_1 + a_2 [\hat{n}/ES]) (n + 1 - i))
\]

If \( \hat{n} = \hat{n}ES \) and \( k = KES \), which implies that \( \mu = K \) and \( s_i = ES, i = 1, \ldots, \mu \), then
ORDINARY LINEAR MODEL ESTIMATION

$T_P(k, \hat{n})$ may be written as

$$T_P(K_{ES}, \hat{n}_{ES}) = ES \sum_{i=1}^{K} (a_0 + (a_1 + a_2\hat{n})(\hat{n} + i))$$

$$= \frac{ES}{2} (2a_0 + (a_1 + a_2\hat{n})(2\hat{n} + K + 1)).$$

Table 4 shows (in seconds) $T_Q(x/ES, y)$ and $T_P(x, y)$ for some values $x$ and $y$. The corresponding execution times of Algorithm 3 and Algorithm 4 are also given. As in the previous cases the timing models are highly accurate.

5. CONCLUSIONS

Aspects of the implementation methods of estimating the OLM on a massively parallel SIMD computer have been considered. Timing models have been constructed for predicting the execution time of the implementations and for providing insight into how the algorithms should be implemented efficiently on the targeted parallel machine. The timing models have been constructed using regression modelling methods. Initially, a timing model of the general step of the algorithm is derived by fitting a regression to a sample of times generated from the execution of this step on matrices with varying dimensions. The predetermined factors used in the regression fit are found from the number of ES × ES memory layers used. The timing model of the single step is then used to provide a theoretical performance model of the algorithm. The theoretical model does not, however, include all of the organizational overheads of the implementations which might account for a large proportion of the execution time. Finally, the factors of the theoretical model are used as the initial set of predetermined variables in a backwards stepwise regression procedure which is applied to a large sample of execution times. This approach results in highly accurate performance models as shown by the results in Tables 1, 3 and 4.

From the experimental results and analysis of the timing models it has been found that the Householder method is superior to that of the Givens method for computing the orthogonal factorization (4). The performance of the algorithms has not been considered when the rank of the $A$ matrix is close to $n$ (number of exogenous variables). In this case, a large number of processing elements will remain idle during the computations if the default mapping is applied to distribute the $L$ and $A$ over the processing elements. This will result in degradation of the performance of the algorithms. The implementation of the algorithms using different mapping layouts for distributing the matrices over the processing elements in order to achieve maximum performance remains to be investigated [22].

Furthermore, in the extreme case where $n - k = 1$ in the factorization (4), the Householder algorithm is equivalent to the PGS, which is in turn equivalent to a simple sequential Givens algorithm. It may be that the PGS performs better than the Householder algorithm when $k$ is very close to $n$. The design of a hybrid algorithm similar to the one in [23] also merits investigation. The Givens and Householder algorithms, based on different mapping strategies, should be combined to achieve the best performance.

The algorithms presented here and those that have been suggested can also be used for block updating of the QR decomposition, which can be seen as being equivalent to the orthogonal factorization (4) [14, 15, 24]. Furthermore, the theoretical and practical techniques discussed in this paper can be used for other parallel linear algebra algorithms.
ACKNOWLEDGEMENTS

The work was carried out using the facilities of the Computer Science Department at Queen Mary and Westfield College, University of London. The author is grateful to Maurice Clint, Hans-Heinrich Nägeli and Sharon Silverne, and the anonymous referees for their constructive comments and suggestions.

REFERENCES
