Efficient algorithms for estimating the general linear model

Petko Yanev a,*, Erricos John Kontoghiorghes b,c

a IRISA/INRIA – Rennes, Campus de Beaulieu, 35042 Rennes cedex, France
b Department of Public and Business Administration, University of Cyprus, P.O. Box 20537, CY-1678 Nicosia, Cyprus
c School of Computer Science and Information Systems, Birkbeck College, University of London, Malet Street, London WC1E 7HX, UK

Received 9 December 2004; received in revised form 7 June 2005; accepted 21 June 2005
Available online 28 November 2005

Abstract

Computationally efficient serial and parallel algorithms for estimating the general linear model are proposed. The sequential block-recursive algorithm is an adaptation of a known Givens strategy that has as a main component the Generalized QR decomposition. The proposed algorithm is based on orthogonal transformations and exploits the triangular structure of the Cholesky QRD factor of the variance–covariance matrix. Specifically, it computes the estimator of the general linear model by solving recursively a series of smaller and smaller generalized linear least squares problems. The new algorithm is found to outperform significantly the corresponding LAPACK routine. A parallel version of the new sequential algorithm which utilizes an efficient distribution of the matrices over the processors and has low inter-processor communication is developed. The theoretical computational complexity of the parallel algorithms is derived and analyzed. Experimental results are presented which confirm the theoretical analysis. The parallel strategy is found to be scalable and highly efficient for estimating large-scale general linear estimation problems.

© 2005 Elsevier B.V. All rights reserved.

Keywords: General linear model; Generalized QR decomposition; Parallel algorithms

1. Introduction

The estimation of the general linear model (GLM) is a well studied problem in statistics [17–19]. It arises in diverse areas such as econometrics, genetics and engineering to name but a few [5,15,16,21]. Emphasis is given to the properties of the estimator rather than the computational merits of the estimation procedures. The GLM is written as

\[ y = X\beta + \varepsilon, \quad \varepsilon \sim (0, \sigma^2\Omega), \]  

(1)
where \( y \in \mathbb{R}^m \) is the response vector, \( X \in \mathbb{R}^{m \times n} \) is the full rank exogenous data matrix, \( \beta \in \mathbb{R}^n \) are the coefficients to be estimated and \( e \in \mathbb{R}^m \) is the noise with zero mean and variance–covariance matrix \( \sigma^2 \Omega \). It is assumed that \( \Omega \) has full rank.

The best linear unbiased estimator (BLUE) of \( \beta \), say \( \hat{\beta} \), is obtained from the solution of the normal equations:

\[
X^T \Omega^{-1} X \hat{\beta} = \Omega^{-1} X^T y.
\]

That is,

\[
\hat{\beta} = (X^T \Omega^{-1} X)^{-1} \Omega^{-1} X^T y. \tag{2}
\]

The derivation of \( \hat{\beta} \) using (2) is computationally expensive and numerically unstable when \( X \) is ill-conditioned. This can be avoided by reformulating the estimation of the GLM as the generalized linear least squares problem (GLLSP)

\[
\arg\min_{u, \beta} u^T u \quad \text{subject to} \quad y = X\beta + Cu. \tag{3}
\]

Here \( \Omega = CC^T \), \( C \in \mathbb{R}^{m \times m} \), \( u \) is a random vector defined by \( Cu = e \), i.e. \( u \sim (0, \sigma^2 I_m) \). The matrix \( C \) is the triangular Cholesky factor of the variance–covariance matrix \( \Omega \). In many applications the triangular \( C \) is known rather than \( \Omega \) [6,10–12,15]. Hereafter it is assumed that \( C \) is available and has an upper-triangular form.

The GLLSP can be solved using the generalized QR decomposition (GQRD) of \( X \equiv (X \ y) \) and \( C \):

\[
Q^T \tilde{X} = \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix} \equiv \begin{pmatrix} R & \tilde{y} \\ 0 & \eta \end{pmatrix} \begin{pmatrix} n & 1 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} n \\ m - n - 1 \end{pmatrix} \tag{4a}
\]

and

\[
(Q^T C) \Pi = U \equiv \begin{pmatrix} U_{1,1} & r & U_{1,2} \\ 0 & \delta & g \\ 0 & 0 & U_{2,2} \end{pmatrix} \begin{pmatrix} n \\ m - n - 1 \end{pmatrix} \tag{4b}
\]

Here \( \tilde{R} \) and \( U \) are upper triangular matrices of orders \( n \) and \( m \), respectively, and \( Q, \Pi \in \mathbb{R}^{m \times m} \) are orthogonal [2,4]. The GLLSP (3) is equivalent to

\[
\arg\min_{u, \beta} \|\Pi^T u\|^2 \quad \text{subject to} \quad Q^T y = Q^T X\beta + Q^T C \Pi \Pi^T u,
\]

which can be written also as

\[
\arg\min_{v_1, v_2, \beta} (\|v_1\|^2 + v^2 + \|v_2\|^2) \quad \text{subject to} \quad \begin{cases} \tilde{y} = R\beta + U_{1,1} v_1 + rv + U_{1,2} v_2, \\ \eta = \delta v + gv_2, \\ 0 = U_{2,2} v_2, \end{cases} \tag{5}
\]

where the vector \( u^T \Pi \) is partitioned as \( (v_1^T \ v^T \ v_2^T) \) and \( \| \cdot \| \) denotes the Euclidean norm. The values of \( v_2 = 0 \) and \( v = \eta/\delta \) can be derived from the last two constraints in (5). Then, setting \( v_1 = 0 \), the BLUE of \( \beta \) is obtained by solving

\[
R\beta = \tilde{y} - \eta r/\delta. \tag{6}
\]

The LAPACK routine (DGGGLM) which computes the estimator of the GLM makes no assumptions on the structure of \( C \). However, the commonly available triangular structure of \( C \) facilitates the development of efficient algorithms for solving the GLLSP [7,12–14]. In this work block recursive sequential and parallel strategies which exploit the structure of the matrices are proposed. The new methods are rich in BLAS-3 operations and solve a series of reduced size GLLSPs.

The algorithms have been implemented on 32 CPUs IBM’s p690+ high-end compute node with 27 GB distributed memory. This parallel machine performs 6.8 Gflop/s has 3077 Mbyte/s bandwidth and communication latency of 1.92 \( \mu \)s. The communications between the processors are realized using the MPI library. The
The performance of the algorithms has been evaluated experimentally. In addition, the theoretical complexities of the algorithms in number of flops (floating point operations) are also presented.

In the Section 2 the serial block-Givens strategy is described and compared with the existing LAPACK routine for estimating the GLM. In Section 3 the parallel algorithm is considered and the theoretical and computational results are presented. In Section 4 some conclusions are drawn. The Appendix shows in detail the derivation of the complexities of the various algorithms.

2. Serial block Givens strategy

An efficient sequential algorithm based on Givens rotations for estimating the GLM has been proposed by Paige [12]. Here a block version based on this sequential strategy is investigated [8,22]. Consider the GLLSP (3), where the matrices $\tilde{X} \equiv (X \ y)$ and $C$ are partitioned, respectively, as

$$
\tilde{X} = \begin{pmatrix}
X_1 & y_1 \\
X_2 & y_2 \\
\vdots & \\
X_k & y_k
\end{pmatrix}^n \quad \text{and} \quad C = \begin{pmatrix}
C_{1,1} & \cdots & C_{1,k} \\
C_{2,1} & \cdots & C_{2,k} \\
\vdots & \ddots & \vdots \\
C_{k,1} & \cdots & C_{k,k}
\end{pmatrix}^n
$$

(7)

Here $C_{i,i}$ ($i = 1, \ldots, k$) are upper triangular where, for simplicity, it is assumed that $m = kn$. The sequential block algorithm computes the solution of the GLLSP (3) in $k$ steps. The GQRD (4a) is computed during the first ($k/2$) steps. For $j = k - i$ the $i$th ($i = 1, \ldots, k - 1$) step computes the smaller GQRD of $\tilde{X}_j y$ and $C_j$.

$$
\begin{pmatrix}
X_j & y_j \\
\tilde{X}_{j+1} & y_{j+1}
\end{pmatrix} \quad \text{and} \quad \begin{pmatrix}
C_{j,j} & \tilde{C}_{j,j+1} \\
0 & \tilde{C}_{j+1,j+1}
\end{pmatrix},
$$

(8)

where $y_j^{(i)} = y_{k-i}$, $y_{k+i}^{(i)} = y_k$, $\tilde{X}_j = X_k$, $\tilde{C}_{k-1,k} = C_{k-1,k}$ and $\tilde{C}_{k,k} = C_{k,k}$. That is, initially the QRD of the first matrix in (8) is computed by:

$$
Q_i^T \begin{pmatrix}
X_j & y_j^{(i)} \\
\tilde{X}_{j+1} & y_{j+1}^{(i)}
\end{pmatrix} = \begin{pmatrix}
\tilde{X}_j & \tilde{y}_j^{(i)} \\
0 & \eta_i
\end{pmatrix}^n 1
$$

(9)

where $Q_i \in \mathbb{R}^{2n \times 2n}$ is orthogonal and $\tilde{X}_j$ is upper triangular. Then, the orthogonal matrix $Q_i^T$ is applied from the left of the second matrix in (8) which is then re-triangularized from the right, i.e.

$$
\left(\begin{array}{cc}
Q_i^T & \tilde{C}_{j,j+1} \\
0 & \tilde{C}_{j+1,j+1}
\end{array}\right) \Pi_i = \begin{pmatrix}
\tilde{C}_{j,j} & r_{i}^{(i)} \\
0 & \delta_i \\
0 & \gamma_i
\end{pmatrix}^n 1
$$

(10)

Here $\Pi_i$ is orthogonal and of order $2n$; $\tilde{C}_{j,j}$ and $\tilde{C}_{j+1,j+1}$ are upper triangular. Once the $i$th GQRD of (8) has been computed, $\Pi_i$ is applied from the right of the affected $j$th and $(j+1)$th block-columns of $C$, i.e. the product

$$
\begin{pmatrix}
C_{1,j} & \tilde{C}_{1,j+1} \\
C_{2,j} & \tilde{C}_{2,j+1} \\
\vdots & \vdots \\
C_{j-1,j} & \tilde{C}_{j-1,j+1}
\end{pmatrix} \Pi_i = \begin{pmatrix}
C_{1,j} & r_{i}^{(i)} \\
C_{2,j} & \tilde{C}_{2,j+1} \\
\vdots & \vdots \\
C_{j-1,j} & \tilde{C}_{j-1,j+1}
\end{pmatrix}^n 1
$$

(11)
Algorithm 1. The sequential block Givens algorithm for solving the GLLSP (3).

1. Let \( \tilde{X} \) and \( \tilde{C} \) in (4a) be partitioned as in (7), where \( m = kn \).
2. Let \( y_{k-1} = y_k \), \( \tilde{X}_k = X_k \), \( \tilde{C}_{k-1,k} = C_{k-1,k} \) and \( \tilde{C}_{k,k} = C_{k,k} \).
3. for \( i = 1, \ldots, k - 1 \) do
4.  Set \( j = k - i \)
5.  Compute the GQRD of \( \frac{X_j}{\tilde{X}_{j+1}} \frac{y_j}{y_{j+1}} \) and \( \frac{C_{j,j}}{\tilde{C}_{j,j+1}} \) as in (9) and (10)
6.  if \( i \neq k - 1 \) then
7.  Compute \( (\tilde{C}_{1,j-1,j} \tilde{C}_{1,j-1,j+1}) \Pi_i \) \( \frac{C_{1,j-1,j}}{\tilde{C}_{1,j-1,j+1}} \) as in (9) and (10)
8.  Update the vector \( y_i \) \( \frac{y_j}{y_{j+1}} \) as in (9) and (10)
9.  end if
10. end for
11. Solve \( \tilde{X}_1 \beta = y_1^{(k)} \)
Algorithm 1 summarizes the steps of this block Givens strategy for estimating the GLM. For the factorizations (9) and (10) Householder transformations are employed. The orthogonal matrices $Q_i$ and $P_i$ ($i = 1, \ldots, k - 1$) are not explicitly constructed. The theoretical complexity (see Appendix) of this algorithm is given by:

$$T_{BG}(m, n) \approx 4m^2n + 14mn^2 - 18n^3. \quad (13)$$

Table 1 shows the execution times in seconds and the theoretical complexities in number of flops of Block–Givens Algorithm 1 (BG) and the LAPACK (LP) routine DGGGLM which estimates the GLM for some values of $n$ and $k$, where $m = kn$ [1]. The theoretical complexity of LAPACK is given by:

$$T_{LP}(m, n) \approx (4m^3 + 12m^2n - 2n^3)/3. \quad (14)$$

Note that, theoretically, Algorithm 1 is approximately $m/3n$ times faster than the LAPACK routine, which is confirmed by the experimental results. This improvement is due to the fact that the Algorithm 1 exploits the triangular structure of the large and computationally expensive matrix $C$ in (3), while the LAPACK routine assumes that $C$ is dense. The experimental results (LP/BG) confirm the theoretical ones ($T_{LP}/T_{BG}$). There is a negligible discrepancy between the two ratios when $k$ is big and $n$ is relatively much smaller. This is due to the increasing overheads which occur from the frequent data exchanges of the submatrices in (7).

### 3. Parallel algorithm

The computation of the product (11) is the most time consuming task in Algorithm 1. This cost can be reduced by applying the orthogonal matrices $P_i$ ($i = 1, \ldots, k - 1$) to the block columns of $C$ (see line 7 of Algorithm 1) in parallel. An efficient parallel algorithm requires a load-balanced distribution of the matrices over the processors and low inter-processor communication [23]. Let $p$ denotes the number of processors and assume that $(k - 2)$ is a multiple of $p$, where $m = kn$.

Consider the partitioning of the matrices $\tilde{X}$ and $C$ as in (7). To achieve low inter-processor communication, the GQRDs computed in line 5 of Algorithm 1 are executed simultaneously by all processors. That is, the data matrix $X$, the last two block rows of $y$ and the main, sub- and super-block diagonals of $C$ are duplicated on each processor. The remaining $(k - 2)$ block-columns of $C$ and $y$ are allocated to the processors using a block-row cyclic distribution. Specifically, $C_{i,i}$ ($i = k-2, \ldots, 1$) is allocated to the processor $P_{i,i}$, where $i = p - (i - 1) \mod p$. The same distribution scheme is used for the vector $y$. This distribution will result in the processor $P_j$ ($j = 1, \ldots, p$) being allocated the vector

$$\gamma^{(j)} = (y_{p+1-j}^T \ y_{2p+1-j}^T \ \cdots \ y_{k-1-j}^T)^T$$

and the matrix

$$C^{(j)} = \begin{bmatrix} C_{p+1-j}^T & C_{2p+1-j}^T & \cdots & C_{k-1-j}^T \end{bmatrix}^T,$$

where $\gamma^{(j)} \in \mathbb{R}^{(k-2)/p \times 1}$ and $C^{(j)} \in \mathbb{R}^{(k-2)/p \times m}$. Fig. 1 shows the distribution of the matrices $\tilde{X}$ and $C$ over the processors, with $p = 4$ and $k = 18$. The shaded blocks indicate those copied to all processors. The blank, unshaded, blocks are null and are unaffected during the computation.
The parallel algorithm solves the GLLSP (3) in \( k \) steps. During the first \((k - 1)\) steps, all processors initially compute the same factorizations (9) and (10). Then each processor updates its allocated submatrix \( C^j \) and subvector \( \gamma^j \). Note that, at the \( i \)th step, each processor updates \([q/p]\) blocks, where \( q = k - i - 1 \). Thus, the processors have equal computational loads. When the local computations have been completed one processor, \( P_j \) say, sends one block from \( C^j \) and \( \gamma^j \), which are required for the next step, to the other processors \( P_r (r = 1, \ldots, p \text{ and } r \neq j) \). That is, at each step only one processor broadcasts an \( n \times n \) submatrix and an \( n \)-element subvector. This broadcast acts as a barrier-synchronization point for the processors before the next step commences. Note that the duplication of the computations results into a one-to-all broadcast which has a low communication cost compared to the extremely expensive all-to-all broadcast. The latter, i.e all-to-all broadcast, is required when there are no duplicating computations and consequently results to a highly inefficient algorithm. The parallel strategy is summarized in Algorithm 2. The broadcast performed by processor \( P_j \) is shown in lines 13–17 of the parallel algorithm.

**Algorithm 2.** The parallel algorithm for solving the GLLSP (3) on \( p \) processors.

1: Let \( \tilde{X} \) and \( C \) be partitioned as in (7), where \( m = kn \) and \( k - 2 \) is a multiple of \( p \).
2: Allocate \( X, y_k - 1, y_k, C_{k,k}, C_{i,i} \) and \( C_{i,i+1} \) \((i = 1, \ldots, k - 1)\) to all processors.
3: Allocate \( \gamma^j \) and \( C^j \) as in (15) and (16), respectively, to processor \( P_j \) \((j = 1, \ldots, p)\).
4: each processor \( P_j (j = 1, \ldots, p) \) do in parallel:
5: for \( i = 1, \ldots, k - 1 \) do
6: Set \( t := k - i \)
7: Compute the GQRD of \( \begin{pmatrix} X_t & \tilde{y}^{(j)}_t \\ \tilde{X}_{t+1} & \tilde{y}^{(j)}_{t+1} \end{pmatrix} \) and \( \begin{pmatrix} C_{t,t} & \tilde{C}_{t,t+1} \\ 0 & \tilde{C}_{t+1,t+1} \end{pmatrix} \) as in (9) and (10).
8: Compute \( y_{k-i}^{(i+1)} = \hat{y}_k^{(j)} - \eta_i \delta r_k^{(j)} - i \).
9: if \( i \neq k - 1 \) then
10: Set \( q := k - i - 1 \)
11: Compute \( \gamma^{(j)}_{1n[q/p],nq+1:nq+2n} = C^{(j)}_{1n[q/p],nq+1:nq+2n} P_i \).
12: Compute \( \gamma^{(j)}_{1n[q/p],nq+1:nq+n+1} = \gamma^{(j)}_{1n[q/p],nq+1:nq+n+1} - \theta_i C^{(j)}_{1n[q/p],nq+1:nq+n+1} \).
13: if \( j = (i - 1) \mod p + 1 \) then
14: Send \( y_{t-1}^{(i+1)} \) and \( \tilde{C}_{t-1,t} \) to \( P_r \), where \( r = 1, \ldots, p \) and \( r \neq j \).
15: else
16: Receive \( y_{t-1}^{(i+1)} \) and \( \tilde{C}_{t-1,t} \) from \( P_r \), where \( r = (i - 1) \mod p + 1 \).
17: end if
18: end if
19: end for
20: Solve \( \tilde{X}_1 \beta = y_1^{(k)} \)

\[ \text{Fig. 1. The row-block cyclic distribution of the matrices on 4 processors, when } k = 18. \]
Table 2
Execution times (s) and efficiency of Algorithm 2

<table>
<thead>
<tr>
<th>µ = m/n − 2</th>
<th>2 Processors</th>
<th>4 Processors</th>
<th>8 Processors</th>
<th>16 Processors</th>
<th>32 Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>µ</td>
<td></td>
<td>Serial Time</td>
<td>2 Processors</td>
<td>4 Processors</td>
<td>8 Processors</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>25</td>
<td>0.76</td>
<td>0.41</td>
<td>.93 (.95)</td>
<td>0.24</td>
</tr>
<tr>
<td>256</td>
<td>25</td>
<td>2.88</td>
<td>1.54</td>
<td>.94 (.97)</td>
<td>0.82</td>
</tr>
<tr>
<td>384</td>
<td>25</td>
<td>6.59</td>
<td>3.47</td>
<td>.95 (.98)</td>
<td>1.75</td>
</tr>
<tr>
<td>512</td>
<td>25</td>
<td>11.95</td>
<td>6.09</td>
<td>.98 (.99)</td>
<td>3.17</td>
</tr>
<tr>
<td>640</td>
<td>25</td>
<td>18.77</td>
<td>9.51</td>
<td>.99 (.99)</td>
<td>4.82</td>
</tr>
<tr>
<td>768</td>
<td>25</td>
<td>26.49</td>
<td>13.54</td>
<td>.98 (.99)</td>
<td>6.82</td>
</tr>
<tr>
<td>896</td>
<td>25</td>
<td>36.23</td>
<td>18.51</td>
<td>.98 (.99)</td>
<td>9.35</td>
</tr>
<tr>
<td>128</td>
<td>50</td>
<td>1.44</td>
<td>0.79</td>
<td>.91 (.90)</td>
<td>0.49</td>
</tr>
<tr>
<td>192</td>
<td>50</td>
<td>5.32</td>
<td>2.83</td>
<td>.94 (.95)</td>
<td>1.58</td>
</tr>
<tr>
<td>256</td>
<td>50</td>
<td>9.71</td>
<td>5.18</td>
<td>.94 (.96)</td>
<td>2.76</td>
</tr>
<tr>
<td>320</td>
<td>50</td>
<td>17.47</td>
<td>9.09</td>
<td>.96 (.97)</td>
<td>4.80</td>
</tr>
<tr>
<td>384</td>
<td>50</td>
<td>27.13</td>
<td>14.14</td>
<td>.96 (.98)</td>
<td>7.45</td>
</tr>
<tr>
<td>448</td>
<td>50</td>
<td>53.71</td>
<td>27.38</td>
<td>.98 (.98)</td>
<td>14.34</td>
</tr>
<tr>
<td>512</td>
<td>100</td>
<td>2.62</td>
<td>1.61</td>
<td>.81 (.84)</td>
<td>1.11</td>
</tr>
<tr>
<td>640</td>
<td>100</td>
<td>10.56</td>
<td>5.94</td>
<td>.89 (.90)</td>
<td>3.61</td>
</tr>
<tr>
<td>768</td>
<td>100</td>
<td>19.04</td>
<td>10.39</td>
<td>.92 (.93)</td>
<td>5.92</td>
</tr>
<tr>
<td>896</td>
<td>100</td>
<td>26.67</td>
<td>14.27</td>
<td>.93 (.95)</td>
<td>8.06</td>
</tr>
<tr>
<td>128</td>
<td>100</td>
<td>58.91</td>
<td>30.93</td>
<td>.95 (.96)</td>
<td>16.76</td>
</tr>
<tr>
<td>192</td>
<td>100</td>
<td>79.85</td>
<td>41.65</td>
<td>.96 (.97)</td>
<td>22.40</td>
</tr>
</tbody>
</table>

The theoretical computational complexity (see Appendix) of this algorithm is given by:

\[ T_p(m, n, p) \approx \left(4m^2n - 16mn^2 + 16n^3\right)/p + 30mn^2 - 34n^3. \]  

From (13) and (17) it follows that the computational efficiency of Algorithm 2 approaches one for very large \( m \), i.e. \( \lim_{m \to \infty} T_{BG}(m,n)/ (p \times T_p(m,n,p)) \approx 1 \). This does not take into account, however, the inter-processor communication. Table 2 shows the execution times and actual (and in brackets the theoretical) efficiency of Algorithm 2 for some \( \mu \) and \( n \), where \( \mu = m/n - 2 \). The experimental results confirm the theoretical complexities. Note that, the communication time increases with the number of the processors which affects the efficiency of the algorithm for small size problems. Furthermore, Algorithm 2 is scalable in the sense that the efficiency remains constant when the size of the problem \( m \), and consequently \( \mu \), is multiplied by \( 2\sqrt{2} \) and the number of the processors \( p \) is doubled. Fig. 2 shows the scalability plots based on the theoretical and computational results presented in Table 2. The model size indicates the \((m/n - 2)\) number of blocks, where each block has dimension \( n \times n \).

4. Conclusion

Computationally efficient sequential and parallel algorithms for computing the best linear unbiased estimator of the general linear model (1) have been proposed. The sequential algorithm is a block version of an efficient serial approach that employs as a main computational component the Generalized QR Decomposition [12]. The new block Givens algorithm exploits the triangular structure of the Cholesky factor \( C \) of the dispersion matrix \( \Omega \) and is rich in BLAS-3 operations. It is found to be \( m/3n \) times faster than the corresponding LAPACK routine DGGLML for estimating the GLM [1].

The parallel approach is based on the new sequential strategy. The parallel algorithm copies the augmented matrix \( \tilde{X} \) and the main, sub- and super-block diagonals of \( C \) to all processors. The rest of the matrix \( C \) is evenly distributed across the processors. The algorithm duplicates parts of the computation. However, this is compensated for the load balanced distribution of the computationally expensive matrix \( C \) resulting in minimal inter-processor communication. The algorithms have been implemented on a parallel computer with distributed memory. The theoretical complexities of both algorithms are stated and experimental results are
The parallel algorithm is found to be scalable and capable of solving large scale GLM estimation problems, where $m \gg n$. The minimal interprocessor communication of the algorithm makes it applicable to other parallel computers (shared memory or distributed memory) with different system specifications. That is, the algorithm is expected to have a similar efficiency on other distributed systems due to the high ratio of local computation to communication cost.

Fig. 2. Theoretical and computational scalability plots based on the results in Table 2. (a) Theoretical efficiency, $n = 25$ (b) Comput. efficiency, $n = 25$ (c) Theoretical efficiency, $n = 50$ (d) Comput. efficiency, $n = 50$ (e) Theoretical efficiency, $n = 100$ (f) Comput. efficiency, $n = 100$. 

Presented and analyzed. Overall, the parallel algorithm is found to be scalable and capable of solving large scale GLM estimation problems, where $m \gg n$. The minimal interprocessor communication of the algorithm makes it applicable to other parallel computers (shared memory or distributed memory) with different system specifications. That is, the algorithm is expected to have a similar efficiency on other distributed systems due to the high ratio of local computation to communication cost.
Currently, an adaptation of the parallel algorithm to estimate Seemingly Unrelated Regressions – a special class of a GLM which involving Kronecker structures – is being investigated [3,6,9,20,24].

Acknowledgements

The authors are grateful to Maurice Clint and the three anonymous referees for their valuable comments and suggestions.

Appendix

The theoretical complexities of computing the QRD of an \( m \times n \) matrix and of applying the orthogonal matrix \( Q \) to an \( m \times q \) matrix, using the corresponding LAPACK routines are given, respectively, by

\[
QR(m, n) = 2n^2(m - n/3)
\]

and

\[
\text{AppQ}(m, n, q) = 2qn(2m - n + 1).
\]

In order to solve the GLM, the LAPACK routine computes the GQRD (4), i.e. it computes the QRD of the \( m \times n \) matrix \( \bar{X} \), it then applies the orthogonal \( Q^T \) to the matrix \( C \in \mathbb{R}^{m \times m} \) and finally it computes the RQD of \( (Q^T C) \). Thus, the theoretical complexity of LAPACK is given by

\[
T_{LP}(m, n) = QR(m, n) + \text{AppQ}(m, n, m) + QR(m, m)
= 2n^2(m - n/3) + 2mn(2m - n + 1) + 2m^2(m - m/3) = (4m^3 + 12m^2n - 2n^3 + 6mn)/3.
\]

This corresponds to the complexity given in (14).

The sequential block Givens algorithm (Algorithm 1) takes advantage of the upper-triangular structure of \( C \) and solves the GLM in \((k - 1)\) steps, where \( m = kn \) at the \( i \)th step \((i = 1, \ldots, k - 1)\) a GQRD of \( 2n \times n \) and \( 2n \times 2n \) matrices is computed (line 5, Algorithm 1). This computation takes \( T_{LP}(2n, n) \) flops. Then, the orthogonal matrix \( \Pi_i \) is applied from the right to a \((m - i + 1)n \times 2n\) matrix (line 7, Algorithm 1). This computation takes \( \text{AppQ}(2n, 2n, m - (i + 1)n) \) flops. The computations for updating the vector \( y \) (line 9, Algorithm 1) is negligible and thus, it is ignored. Finally, the total theoretical complexity of Algorithm 1 shown in (13) is calculated by:

\[
T_{BG}(m, n) = \sum_{i=1}^{m/n-1} (T_{LP}(2n, n) + \text{AppQ}(2n, 2n, m - (i+1)n))
= (m - n)(32n^3 + 48n^3 - 2n^3 + 12n^2)/3n + \sum_{i=1}^{m/n-1} 4n(m - (i+1)n)(2n + 1)
= (m - n)(26n^3 + 4n) + 2(m - n)(m - 2n)(2n + 1) = 4m^2n + 14mn^2 - 18n^3 + 2m^2 - 2mn.
\]

The computations of the parallel algorithm (Algorithm 2) are dominated by the last processor (e.g. \( P4 \) in Fig. 1). Similarly to the sequential algorithm, it solves the GLM in \((k - 1)\) steps, where \( m = kn \). At the \( i \)th step \((i = 1, \ldots, k - 1)\) a GQRD of \( 2n \times n \) and \( 2n \times 2n \) matrices is computed (line 7, Algorithm 2). This computation takes \( T_{LP}(2n, n) \) flops. Then, the orthogonal \( \Pi_i \) is applied from the right of a \((n[(m-(i+1)n)/np] \times 2n)\) matrix (line 11, Algorithm 2). This computation requires \( \text{AppQ}(2n, 2n, n[(m - (i + 1)n)/np]) \) flops. The computations for updating the vector \( y \) (line 12, Algorithm 2) are very small and are neglected. Thus, the total computational theoretical complexity of Algorithm 2 in (17) has been calculated by:

\[
T_p(m, n, p) = \sum_{i=1}^{m/n-1} (T_{LP}(2n, n)) + \text{AppQ}(2n, 2n, n[(m - (i + 1)n)/np])
= (m - n)(26n^2 + 4n) + 2(m - 2n)(m - 2n + pn)(2n + 1)/p
= (4m^2n - 16mn^2 + 16n^3 + 2m^2 - 8mn + 8n^2)/p + 30mn^2 - 34n^3 + 6mn - 8n^2.
\]
References


