Abstract

A class of parallel scaled Givens rotations, to be applied to weighted multiple linear least squares problems, is discussed.

In comparison to Fast Givens transformations, properly scaled rotations for weighted problems exhibit the same stability, require fewer divisions, and avoid square roots as well as pivoting. Consequently, with a suitable elimination strategy, the algorithm is amenable to parallel linear-time implementation on systolic arrays in VLSI. Round off error and stability analyses are presented, indicating slightly less accumulation of round off error than known sequential methods.

Parallel Scaled Givens Rotations for the Solution of Linear Least Squares Problems

Jesse L. Barlow[1]
Ilse C.F. Ipsen[2]

Research Report YALEU/DCS/RR-310
March 1984

[1]Department of Computer Science, The Pennsylvania State University. This work was supported by the National Science Foundation under contract MCS-8201065 and by the Office of Naval Research under contract N0014-80-0517.

[2]Department of Computer Science, Yale University. This work was supported by the Office of Naval Research under contract N000014-82-K-0184.

1. Introduction

The weighted linear least squares problem is that of finding an $n \times 1$ vector v^* such that

$$\|D(Av^* - s)\| = \min_{v \in R^n} \|D(Av - s)\|, \tag{1.1}$$

where A is an $m \times n$ matrix, s is an $m \times 1$ vector, v is a $n \times 1$ vector, and $D = diag(d_1, \ldots, d_m)$ is the diagonal $m \times m$ matrix wherein, usually, $d_i > 0$, $1 \leq i \leq m$. The symbol $\|\cdot\|$ denotes the Euclidean vector or matrix norm depending on context.

It will be presumed throughout the discussion that $m \geq n$ (usually $m \gg n$) and that A has rank n to machine precision.

A parallel linear time solution to problem (1.1) will be presented. The triangular decomposition of A and corresponding modifications on s are obtained from a VLSI (Very Large Scale Integration) implementation of a systolic array of processors. A systolic device for back substitution can be found in [13].

According to VLSI philosophy , processor interconnections are planar and of nearest neighbour type, while processing elements are primitive, i.e., perform only elementary arithmetic operations. Pipelining replaces global broadcasting of data, so only local exchange of data takes place. Of course, computations are to be evenly distributed over the processors for good hardware utilisation. The above restrictions together with numerical aspects rule out the following methods.

- Normal equations approach : squaring a matrix constitutes an illconditioned task, hence, methods must involve orthogonal transformations.
- Householder-Golub factorisation [4] : its global data communication is not suited to implementation in VLSI, rendering a rather inflexible architecture [12].
- Standard Givens rotations : they are easily adapted to systolic arrays [10], [7], [1], [3], but require a square root computation per transformation. Besides consuming a major part of the chip area, they also constitute a severe bottle neck with regard to computation time.
- Fast Givens rotations : they are faster than standard Givens rotations when solving (1.1) sequentially. As the matrix is considered in factored form, square roots are obviated and the number of multiplications is halved. These advantages carry over to parallel devices, as a systolic array implementation [11] makes clear. Yet, unlike standard Givens rotations, the fast rotations demand 'nearest neighbour pivoting' which greatly increases the complexity and area of processing elements. Furthermore, careful monitoring of element growth, important for the prevention of under- and overflow, is not conveniently achieved [8] p.160.

Consequently, the parallel solution of (1.1) on a VLSI device must rely on the development of scaled rotations which do without square roots and pivoting. Additionally, the scaled rotations to be presented here involve only one division as opposed to two for the standard transformations. A decrease in the amount of multiplications is not really crucial if a good hardware utilisation and a balanced distribution of the computation among the processors is achievable.

The scaled rotations are easily adapted to the systolic structure for the standard [10] and fast [11] rotations by slight reprogramming of the processing elements (in fact, the scaling technique is best used in a parallel context). The resulting decomposition is as stable as any method available, with an error bound at least as good as one based on standard Givens transformations.

Scaled Givens rotations are most useful for forms of problem (1.1) wherein the weight matrix $W = D^2$ is not the identity matrix, because the premultiplication of A by the square root of the weight matrix, D, is unnecessary. Such problems arise in signal processing and least squares filtering when earlier signals are weighted less compared to later ones. Another context in which scaled Givens transformations have advantages over standard rotations occurs when a row of the

decomposed matrix is to be removed. In this case the weight of the row is set to -1 (cf. [14]). It remains to be investigated how practical these techniques are in a parallel implementation.

There are five sections after this introductory one. Section two contains a review of standard Givens rotations, fast Givens rotations and a 2×2 version of Bareiss' G-transformation [2]. Section three, describing and motivating the parallel scaled rotations, is followed by an error analysis in section four. Before some concluding remarks in chapter six, a parallel VLSI implementation of the triangular decomposition with scaled rotations is given.

2. Standard and fast Givens rotations

The Givens (or Jacobi) plane rotation is a device for inserting zeroes into a matrix selectively. Let $M = (m_{ij})$ be a $2 \times n$ matrix. The Givens rotation H that annihilates the $(2,1)$ entry of $M' = HM$ is given by

$$H = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \tag{2.1}$$

where

$$\gamma = m_{11}/r, \quad \sigma = m_{21}/r, \quad r = \sqrt{m_{11}^2 + m_{21}^2}.$$

Note that

$$\gamma^2 + \sigma^2 = 1,$$

and H is orthogonal. The $2 \times n$ matrix $M' = HM$ is computed according to

$$m'_{1j} = \gamma m_{1j} + \sigma m_{2j}, \quad m'_{2j} = -\sigma m_{1j} + \gamma m_{2j}, \qquad j = 2, \ldots n, \tag{2.2a}$$

$$m'_{11} = r, \quad m'_{21} = 0. \tag{2.2b}$$

This computation requires one square root, two divisions, $4n - 2$ multiplications, and $2n - 1$ additions. Even if the square root were implemented in hardware, it would still takes as long as two or three multiplications, thus contributing significantly to the computation time involved in forming H.

Fast Givens rotations [5] [9] require only half the number of multiplications and avoid the formation of square roots. These operations are saved by considering M in the form

$$M = KB \tag{2.3}$$

where $K = diag(k_1, k_2)$ is a diagonal matrix and $B = (b_{ij})$. One then determines a transformation G of the form

$$G = (K')^{-1} HK \tag{2.4}$$

where H is given in (2.1) and $K' = diag(k'_1, k'_2)$. If

$$B' = GB, \tag{2.5}$$

it is easily seen that

$$K'B' = K'GB = K'(K')^{-1}HKB = HM = M'. \tag{2.6}$$

The saving in multiplications and square roots results from the proper choice of K'. A number of such choices is given in [9]. Gentleman [5] gives the following procedure for forming the matrix G in (2.4). K' is chosen implicitly.

Define

$$W = K^2 = diag(w_1, w_2) = diag(k_1^2, k_2^2)$$

and

$$W' = (K')^2 = diag(w_1', w_2') = diag((k_1')^2, (k_2')^2).$$

The matrices M' and K' are not explicitly computed, so M and K are not used. Instead one forms G from W and B and computes only W' and B'. Regarding the first column of B, three cases are distinguished when forming G.

Case I : $b_{21} = 0$.

$$W' = W, \quad B' = B, \quad G = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{2.7}$$

Case II : $w_1 b_{11}^2 \geq w_2 b_{21}^2 > 0$. Let $t = \frac{w_2 b_{21}^2}{w_1 b_{11}^2}$, then

$$\begin{aligned} G &= \begin{pmatrix} 1 & \frac{w_2 b_{21}}{w_1 b_{11}} \\ -\frac{b_{21}}{b_{11}} & 1 \end{pmatrix}, \\ B' &= GB, \\ W' &= (1+t)^{-1} W. \end{aligned} \tag{2.8}$$

Note that

$$b_{11}' = b_{11}(1+t), \quad b_{21} = 0.$$

Case III : $w_2 b_{21}^2 \geq w_1 b_{11}^2 \geq 0$. Let $t = \frac{w_1 b_{11}^2}{w_2 b_{21}^2}$, then

$$\begin{aligned} G &= \begin{pmatrix} \frac{w_1 b_{11}}{w_2 b_{21}} & 1 \\ -1 & \frac{b_{21}}{b_{11}} \end{pmatrix}, \\ B' &= GB, \\ W' &= (1+t)^{-1} diag(w_2, w_1). \end{aligned} \tag{2.9}$$

Note that

$$b_{11}' = b_{21}(1+t), \quad b_{21}' = 0.$$

In cases II and III the computation of GB takes three divisions, $2n+5$ multiplications, and $2n$ additions. The distinction between cases II and III is essentially a pivot. This 'pivot' is necessary since for small $w_1 b_{11}^2$ computations with the matrix G in (2.8) are unstable. The same can be said of the matrix G in (2.9) if $w_2 b_{21}^2$ is small. Unfortunately, this 'pivot' adds considerably to the complexity of the processing elements in a parallel device [11].

In a matrix decomposition algorithm based on Fast Givens rotations the matrix to be decomposed must be periodically rescaled. Yet, there is no completely satisfactory method for performing this rescaling.

The scaled rotations to be discussed are based in part upon the 2 × 2 version of Bareiss' G-transformation [2],

$$G = \begin{pmatrix} \frac{w_1 b_{11}}{\rho} & \frac{w_2 b_{21}}{\rho} \\ -\frac{b_{21}}{b_{11}} & 1 \end{pmatrix}, \tag{2.10}$$

where $\rho = r^2$, which can have 'pivoting' and scaling problems similar to those of Fast Givens rotations.

In the next section, a scaled Givens rotation is developed in order to prevent problems of pivoting and scaling during parallel computations.

3. Scaled Givens Rotations for Parallel Computation

Before constructing a scaled Givens rotation, it is necessary to discuss a computational algorithm which, when based on these rotations, solves problem (1.1). Such an algorithm would decompose the $m \times n$ matrix A in (1.1) as follows :

$$A_0 = A, \quad s_0 = s, \tag{3.1a}$$

$$A_{i+1} = G_{i+1}A_i, \quad s_{i+1} = G_{i+1}s_i, \qquad i = 1, \ldots, l-1. \tag{3.1b}$$

$l = mn - (n^2+n)/2$ is the number of rotations necessary to transform A into an upper trapezoidal matrix. Each G_i has the form

$$G_i = D_i^{-1} H_i D_{i-1}, \qquad i = 1, \ldots, l \tag{3.2}$$

with $D_0 = D$ in (1.1), D_i properly chosen diagonal matrices, and H_i Givens rotations, $1 \le i \le l$.

An important concern for algorithms of the form (3.1) is the growth of elements of A; overflows or spurious underflows are to be avoided.

Define the norm

$$\|S\|_{D_0} \stackrel{def}{=} \|D_0 S D_0^{-1}\| \tag{3.3}$$

for an $m \times m$ matrix S. If S is an $m \times n$ matrix where $n < m$ then

$$\|S\|_{D_0} \stackrel{def}{=} \|(S,0)\|_{D_0}, \tag{3.4}$$

where $(S,0)$ is an $m \times m$ matrix whose last $m-n$ columns are zero.

After combining (3.1) and (3.2) one obtains

$$\begin{aligned} A_i &= G_i \ldots G_1 A \\ &= (D_i^{-1} H_i D_{i-1})(D_{i-1}^{-1} H_i D_{i-2}) \ldots (D_1^{-1} H_1 D_0) A \\ &= D_i^{-1} H_i \ldots H_1 D_0 A = D_i^{-1} Q_i D_0 A. \end{aligned} \tag{3.5}$$

The matrix Q_i is orthogonal if the effects of machine rounding are ignored. Classical norm inequalities with the norm in (3.3) yield

$$\begin{aligned} \|A_i\|_{D_0} &\le \|D_i^{-1} Q_i D_0\|_{D_0} \|A\|_{D_0} = \|D_0 D_i^{-1} Q_i\| \|A\|_{D_0} \\ &= \|D_0 D_i^{-1}\| \|A\|_{D_0}. \end{aligned} \tag{3.6}$$

A similar derivation results in the lower bound

$$\|A_i\|_{D_0} \ge \|A\|_{D_0} / \|D_i D_0^{-1}\|. \tag{3.7}$$

Thus if $\|D_0 D_i^{-1}\|$ does not grow large, then overflows in A_i are unlikely (they may still occur if D_0 is chosen badly enough, but such cases seldom arise in practice). Spurious underflows are unlikely if $\|D_i D_0^{-1}\|$ does not become too small. The Fast Givens rotation method guarantees that the elements of A_i will not underflow. However, overflow in the elements of A_i is a strong possibility, as is underflow in the elements of D_i. Since the Euclidean norm of $D_i A_i$ is invariant with respect to i, it suffices to monitor the diagonal matrix D_i for underflows. According procedures are discussed in [9] [14], p.62. Unfortunately, none of them is computationally convenient.

The scaled Givens rotation of this section is designed to be easily implementable in parallel hardware; it rescales subject to the values of $\|D_0 D_i^{-1}\|$ and $\|D_i D_0^{-1}\|$. With the same number of

multiplications as in a standard Givens rotation it avoids the pivoting of a Fast Givens rotation. It requires no square roots and only one division - two less than the Fast rotation.

To observe the effect of *one* rotation, let G be the 2×2 nontrivial portion of the scaled Givens rotation G_i, let K be the portion of D_{i-1} pertaining to the two rows altered by G_i , K' the corresponding portion of D_i , $K^{(0)}$ the corresponding portion of D_0, and B the submatrix of A_{i-1} that is altered by G_i. As in section two, it is the judicious choice of K' that gives the scaled Givens rotation its desirable properties.

The 2×2 matrix G for the scaled Givens rotation is

$$G = (K')^{-1} H K,$$

where H is the standard Givens rotation which inserts a zero in the $(2,1)$ entry of B. The matrix H has the form

$$H = \begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix},$$

where

$$\gamma = \frac{k_1 b_{11}}{r}, \quad \sigma = \frac{k_2 b_{21}}{r}, \quad r = \sqrt{k_1^2 b_{11}^2 + k_2^2 b_{21}^2}.$$

The choice of K' given by

$$K' = \begin{pmatrix} \frac{2^{-\alpha}}{r} & 0 \\ 0 & \frac{2^{-\beta} k_1 k_2}{r} \end{pmatrix}, \tag{3.8}$$

where α and β are integers chosen for scaling purposes, yields the matrix

$$G = (K)^{-1} H K = \begin{pmatrix} r2^{\alpha} & 0 \\ 0 & \frac{r2^{\beta}}{k_1 k_2} \end{pmatrix} \begin{pmatrix} \frac{k_1 b_{11}}{r} & \frac{k_2 b_{21}}{r} \\ -\frac{k_2 b_{21}}{r} & \frac{k_1 b_{11}}{r} \end{pmatrix} \begin{pmatrix} k_1 & 0 \\ 0 & k_2 \end{pmatrix} = \begin{pmatrix} 2^{\alpha} k_1^2 b_{11} & 2^{\alpha} k_2^2 b_{21} \\ -2^{\beta} b_{21} & 2^{\beta} b_{11} \end{pmatrix}.$$

Thus,

$$G = \begin{pmatrix} 2^{\alpha} w_1 b_{11} & 2^{\alpha} w_2 b_{21} \\ -2^{\beta} b_{21} & 2^{\beta} b_{11} \end{pmatrix}. \tag{3.9}$$

If

$$B' = GB,$$

then

$$b'_{11} = 2^{\alpha} r^2, \quad b'_{21} = 0.$$

As with Fast Givens rotations, the matrix K is not used, and the matrix K' is not explicitly computed. Instead $W = K^2 = diag(w_1, w_2) = diag(k_1^2, k_2^2)$ serves to determine

$$W' = (K')^2 = diag(w'_1, w'_2) = \begin{pmatrix} \frac{2^{-2\alpha}}{r^2} & 0 \\ 0 & \frac{2^{-2\beta} k_1^2 k_2^2}{r^2} \end{pmatrix} = \begin{pmatrix} \frac{2^{-2\alpha}}{\rho} & 0 \\ 0 & \frac{2^{-2\beta} w_1 w_2}{\rho} \end{pmatrix} \tag{3.10}$$

and $\rho = r^2 = w_1 b_{11}^2 + w_2 b_{21}^2$. Thus throughout (3.1), D_i^2 rather than D_i is stored.

The integers α and β are chosen according to the algorithm below, where

$$W^{(0)} = diag(w_1^{(0)}, w_2^{(0)}) = (K^{(0)})^2 = diag((k_1^{(0)})^2, (k_2^{(0)})^2).$$

Algorithm 3.1

1. Find the integer α such that $w_1^{(0)}\rho 2^{2\alpha} \in [\frac{1}{2}, 2)$ { This insures that $w_1^{(0)}/w_{11}' \in [\frac{1}{2}, 2)$ }.
2. Compute $\varsigma = 1/\rho$ { This is the only division necessary }.
3. Compute $\xi = w_1 w_2 \varsigma$.
4. Subtract the floating point exponent of ξ from that of $w_2^{(0)}$. Call this integer ω.
5. If ω is even then $\beta = \omega/2$, otherwise $\beta = (\omega+1)/2$ { This insures that $w_2^{(0)}/w_2' \in [1/4, 2)$ }.

These values of α and β ascertain

$$\|W^{(0)}(W')^{-1}\| \leq 2$$

and

$$\|W'(W^{(0)})^{-1}\| \leq 4.$$

Therefore,

$$\|K^{(0)}(K')^{-1}\| \leq \sqrt{2} \tag{3.11}$$

and

$$\|K'(K^{(0)})^{-1}\| \leq 2. \tag{3.12}$$

Since

$$\|D_0 D_i^{-1}\| \leq \max\{\|D_0 D_{i-1}^{-1}\|, \|K^{(0)}(K')^{-1}\|\}$$

and

$$\|D_i D_0^{-1}\| \leq \max\{\|D_{i-1} D_0^{-1}\|, \|K'(K^{(0)})^{-1}\|\},$$

a simple induction argument renders the inequalities

$$\|D_0 D_i^{-1}\| \leq \sqrt{2}, \qquad i = 1, 2, \ldots, l \tag{3.13a}$$

$$\|D_i D_0^{-1}\| \leq 2, \qquad i = 1, 2, \ldots, l. \tag{3.13b}$$

Combining (3.13) with (3.6) and (3.7) yields

$$\frac{1}{2}\|A\|_{D_0} \leq \|A_i\|_{D_0} \leq \sqrt{2}\|A\|_{D_0}, \qquad i = 1, 2, \ldots, l.$$

Hence, algorithm (3.1) using the scaled Givens rotations is resistant to underflows and overflows.

In section five it is shown that this kind of scaling is practically free in a parallel architecture. Proper scaling procedures for fast Givens transformations have been a persistent problem in their implementation [8], p.160.

The fact that the rotation in (3.9) takes twice as many multiplications as fast Givens rotation seems to be of disadvantage. However, from the point of designing processing elements it is not. A processing element performs a rotation on a single 2×1 vector at a time. In the case of fast Givens rotations it would have to either distinguish between the cases in equations (2.8) and (2.9) for each matrix-vector multiplication or store all four elements of the matrix G and perform four multiplications, see [11]. The first approach greatly increases the complexity of the processing elements, while the second completely ignores the principal advantage of fast rotations.

In the next section, a parallel algorithm for solving problem (1.1) using scaled rotations is proved to be reasonably stable, and as accurate as extant methods.

4. Error Analysis

The following algorithm will be used to solve (1.1) :

$$X_0 = X = A, \quad f_0 = f = s, \tag{4.1a}$$

$$X_{k+1} = T_{k+1}X_k = A_{l_{k+1}}, \quad f_{k+1} = T_{k+1}f_k = s_{l_{k+1}}, \qquad k = 0, 1, \ldots, \nu - 1, \tag{4.1b}$$

where

$$T_k = G_{l_k} \dots G_{l_{k-1}+1}$$

is the product of the disjoint scaled Givens rotations G_i, $i = l_{k-1}+1, \dots, l_k$ (two Givens rotations are disjoint if they affect disjoint pairs of rows). The indices l_k, $k = 1, 2, \dots, \nu$, denote the number of scaled Givens rotations performed at the kth stage. The indices $l_0 = 0$ and $l_\nu = l$ are the upper and lower limits of equations (3.1). Each G_i has the form (3.9). Equations (4.1) are simply a special version of equations (3.1). Therefore,

$$\begin{aligned} T_k &= (D_{l_k}^{-1} H_{l_k} D_{l_k-1})(D_{l_k-1}^{-1} H_{l_k-1} D_{l_k-2}) \dots (D_{l_{k-1}+1}^{-1} H_{l_{k-1}+1} D_{l_{k-1}}) \\ &= D_{l_k}^{-1} Q_k D_{l_{k-1}}, \qquad k = 1, 2, \dots, \nu, \end{aligned}$$

where $Q_k = H_{l_k} \dots H_{l_{k-1}+1}$ is orthogonal. Define $C_k = D_{l_k}$, $k = 0, 1, \dots, \nu$. Then a simpler form for T_k is given by

$$T_k = C_k^{-1} Q_k C_{k-1}, \qquad k = 1, 2, \dots, \nu. \tag{4.2}$$

Presumably, the final matrix X_ν has the form

$$X_\nu = \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where R is a $n \times n$ nonsingular and upper triangular matrix. Neglecting rounding errors, the algorithm guarantees

$$\|C_0(Xv - f)\| = \|C_\nu (X_\nu v - f_\nu)\|.$$

If f_ν is partitioned according to

$$f_\nu = \begin{pmatrix} f_\nu^{(1)} \\ f_\nu^{(2)} \end{pmatrix},$$

where $f_\nu^{(1)}$ is a $n \times 1$ and $f_\nu^{(2)}$ a $(m-n) \times 1$ vector, then the solution v^* to (1.1) is given by

$$v^* = R^{-1} f_\nu^{(1)}$$

with the residual

$$\|C_\nu (X_\nu v^* - f_\nu)\| = \|C_\nu \left(0, -f_\nu^{(2)}\right)^T\|.$$

In order to estimate the effect of rounding errors in (4.1) on the solution, (1.1) is posed as an unweighted least squares problem and the effect of the algorithm on the solution to that problem will be considered. With

$$Y = Y_0 = C_0 X = C_0 X_0, \quad z = z_0 = C_0 f,$$

and

$$Y_k = C_k X_k, \quad z_k = C_k f_k, \qquad k = 1, 2, \dots, \nu,$$

(4.1), in effect, performs the sequence of operations

$$Y_0 = Y = C_0 X, \quad z_0 = z = C_0 f, \tag{4.3a}$$

$$Y_{k+1} = Q_{k+1} Y_k, \quad z_{k+1} = Q_{k+1} z_k, \qquad k = 1, 2, \dots, \nu \tag{4.3b}$$

to solve the problem of finding v^* such that

$$\|Yv^* - z\| = \min_{v \in R^n} \|Yv - z\|. \tag{4.4}$$

If the error is analysed for the sequence of orthogonal transformations in (4.3), rather than the scaled rotations (4.1), the results on error analyses of orthogonal transformations in [16], [5], and [6] can be directly applied.

The classical techniques of backward error analysis from [16] will be employed to bound $\|u^* - v^*\|$, where u^* satisfies the perturbed problem

$$\|(Y + \delta Y)u^* - (z + \delta z)\| = \min_{u \in R^n} \|(Y + \delta Y)u - (z + \delta z)\|. \tag{4.5}$$

In order to obtain this bound four quantities must be estimated :

- the condition number $\|Y\|\|Y^+\|$ where Y^+ is the Moore-Penrose inverse of Y,
- the norm of the residual $\|Yv^* - z\|$,
- the norm of the backward error in Y, $\|\delta Y\|$,
- the norm of the backward error in z, $\|\delta z\|$.

The first two quantities are dependent on the problem (4.4), not on the algorithm used to solve it. The last two quantities depend upon the algorithm and the precision of the underlying arithmetic. The relation between these four quantities and the forward error $\|u^* - v^*\|$ is given in [14], p.50.

Thus it is enough to bound the quantities $\|\delta Y\|$ and $\|\delta z\|$ to show that the scaled Givens rotations produce an algorithm that is as stable as any available.

The first theorem discusses the rounding error from a single scaled Givens rotation. The symbol $fl(\cdot)$ will denote the floating point computation performed on its argument (for the use of this convention see [16]).

Theorem 4.1. *Let G be the 2×2 scaled Givens rotation defined by the first column of the $2 \times n$ matrix B and the weight matrix $W = diag(w_1, w_2)$; and $M = KB$ where $K = W^{\frac{1}{2}} = diag(k_1, k_2)$. Assume, G is given by*

$$G = (K')^{-1} H K,$$

where H is the standard Givens rotation defined by the first column of M; and $K' = (W')^{\frac{1}{2}}$ is the new weight matrix defined by (3.8). *If $\tilde{B}'$ is the machine computation of GB and $\tilde{M}' = \tilde{K}'\tilde{B}'$, then*

$$\tilde{M}' + \delta M' = \tilde{K}'\tilde{B}' + \delta M' = HKB = HM$$

where $\|\delta M'\|_F \leq 5.5\mu\|M\|_F + O(\mu^2)$ and μ is the machine unit.

Two lemmata are needed for the proof of the theorem. The first bounds the perturbations in the weight matrix K' while the second bounds the perturbations in the Givens rotation matrix H.

Lemma 4.1. *Let K' and W' be defined as in Theorem 4.1 and $\tilde{K}' = (\tilde{W}')^{\frac{1}{2}}$, where $\tilde{W}'$ is the machine computation of W'. Then*

$$\tilde{K}' - K' = (\delta K')K',$$

and $\|\delta K'\| \leq 2.5\mu + O(\mu^2)$.

Proof. The first order Taylor expansion

$$\sqrt{1+\epsilon} = 1 + \frac{\epsilon}{2} + O(\epsilon^2)$$

for ϵ small is used to bound the error in

$$\tilde{K}' = \begin{pmatrix} fl(k'_1) & 0 \\ 0 & fl(k'_2) \end{pmatrix},$$

where

$$fl(k'_1) = \sqrt{fl(w'_1)}, \quad fl(k'_2) = \sqrt{fl(w'_2)}$$

since instead of k'_1 and k'_2, w'_1 and w'_2 are actually computed.

Performing the computations according to algorithm 3.1,

$$fl(w'_1) = 2^{-2\alpha} fl(1/(w_1 b_{11}^2 + w_2 b_{21}^2)) = w'_1(1 + \delta w'_1),$$

where $|\delta w'_1| \leq 3\mu + O(\mu^2)$ and

$$fl(w'_2) = 2^{-2\beta} fl(w_1 w_2/\sigma) = w'_2(1 + \delta w'_2),$$

where $|\delta w'_2| \leq 5\mu + O(\mu^2)$. With the above Taylor expansion of $\sqrt{1+\epsilon}$ one has

$$\begin{aligned} fl(k'_1) &= k'_1(1 + \delta k'_1), \quad |\delta k'_1| \leq 1.5\mu + O(\mu^2), \\ fl(k'_2) &= k'_2(1 + \delta k'_2), \quad |\delta k'_2| \leq 2.5\mu + O(\mu^2). \end{aligned}$$

This gives

$$\tilde{K}' = (I + \delta K')K', \quad \|\delta K'\| \leq 2.5\mu + O(\mu^2)$$

which is the desired result.

∎

Lemma 4.2. *Let G,K, and K' be as defined in Theorem 4.1 and let $\tilde{G}$ be the machine computation of G. Then*

$$\|K'(G - \tilde{G})K^{-1}\|_F \leq \mu.$$

Thus also

$$\|K'\tilde{G}K^{-1}\| \leq 1 + \mu.$$

Proof.

$$G = \begin{pmatrix} 2^\alpha w_1 b_{11} & 2^\alpha w_2 b_{21} \\ -2^\beta b_{21} & 2^\beta b_{11} \end{pmatrix} = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix}$$

gives

$$\tilde{G} = fl(G) = \begin{pmatrix} \tilde{g}_{11} & \tilde{g}_{12} \\ \tilde{g}_{21} & \tilde{g}_{22} \end{pmatrix},$$

where

$$\begin{aligned} \tilde{g}_{11} &= 2^\alpha w_1 b_{11}(1 + \epsilon_{11}), \quad \tilde{g}_{12} = 2^\alpha w_2 b_{21}(1 + \epsilon_{12}), \\ \tilde{g}_{21} &= -2^\beta b_{21} = g_{21}, \quad \tilde{g}_{22} = g_{22}. \end{aligned}$$

The multiplications by 2^α and 2^β are just shifts. They cause no rounding error. Hence $\tilde{g}_{21}$ and $\tilde{g}_{22}$ are exact. Now

$$K'\tilde{G}K^{-1} = \begin{pmatrix} \tilde{\gamma} & \tilde{\sigma} \\ -\sigma & \gamma \end{pmatrix} = \begin{pmatrix} \gamma(1 + \epsilon_{11}) & \sigma(1 + \epsilon_{12}) \\ -\sigma & \gamma \end{pmatrix},$$

where

$$\gamma = \frac{k_1 b_{11}}{r}, \quad \sigma = \frac{k_2 b_{21}}{r}$$

and

$$r = \sqrt{w_1 b_{11}^2 + w_2 b_{21}^2}.$$

Note again that the bottom row of $K'\tilde{G}K^{-1}$ is exact. Thus

$$\|K'(\tilde{G} - G)K^{-1}\|_F = \| \begin{pmatrix} \gamma\epsilon_{11} & \sigma\epsilon_{12} \\ 0 & 0 \end{pmatrix} \|_F \leq \mu,$$

and

$$\|K'\tilde{G}K^{-1}\| \leq \|K'GK^{-1}\| + \|K'(\tilde{G} - G)K^{-1}\| = \|H\| + \|K'(\tilde{G} - G)K^{-1}\| \leq 1 + \mu.$$

∎

It is now possible to prove Theorem 4.1.

Proof. By definition,

$$\begin{aligned} \|\delta M'\|_F &= \|\tilde{M}' - HM\|_F = \|\tilde{K}'\tilde{B}' - K'B'\|_F = \|\tilde{K}' fl(\tilde{G}B) - K'GB\|_F \\ &= \|\tilde{K}' fl(\tilde{G}B) - \tilde{K}'\tilde{G}'B + \tilde{K}'\tilde{G}'B - K'\tilde{G}B + K'\tilde{G}B - K'GB\|_F \\ &\leq \|\tilde{K}' fl(\tilde{G}B) - \tilde{K}'\tilde{G}B\|_F + \|(\tilde{K}' - K')\tilde{G}B\|_F + \|K'(\tilde{G} - G)B\|_F. \end{aligned} \tag{4.6}$$

The last two terms of (4.6) can be bounded using Lemmata 4.1 and 4.2. The second term has the bound

$$\begin{aligned} \|(\tilde{K}' - K')GB\|_F &= \|(\delta K')K'G'B\|_F \leq \|\delta K'\|\|K'\tilde{G}B\|_F \\ &= \|\delta K'\|\|K'\tilde{G}K^{-1}M\|_F \leq \|\delta K'\|\|K'\tilde{G}K^{-1}\|\|M\|_F. \end{aligned}$$

With Lemmata 4.1 and 4.2,

$$\|(\tilde{K}' - K')\tilde{G}B\|_F \leq 2.5\mu(1 + \mu)\|M\|_F + O(\mu^2) = 2.5\mu\|M\|_F + O(\mu^2).$$

The third term is bounded by

$$\begin{aligned} \|K'(\tilde{G} - G)B\|_F &= \|K'(\tilde{G} - G)K^{-1}M\|_F \\ &\leq \|K'(\tilde{G} - G)K^{-1}\|_F\|M\|_F = \mu\|M\|_F. \end{aligned}$$

It now suffices to bound the first term of (4.6), whereby the following inequality will be useful,

$$\begin{aligned} \|\tilde{K}'(fl(\tilde{G}B) - GB)\|_F &\leq (\|K'\| + \|\delta K'\|)\|fl(\tilde{G}B) - GB\|_F \\ &\leq (1 + 2.5\mu)\|K' fl(\tilde{G}B) - GB\|_F \\ &= (1 + 2.5\mu)\|K'(fl(\tilde{G}B) - GB)\|_F. \end{aligned} \tag{4.7}$$

Let

$$\begin{aligned} \tilde{M}' &= (\tilde{m}_1', \tilde{m}_2', \ldots, \tilde{m}_n'), \quad M = (m_1, m_2, ..., m_n), \\ \tilde{B}' &= (\tilde{b}_1', \tilde{b}_2', \ldots, \tilde{b}_n'), \quad B = (b_1, b_2, \ldots, b_n) \end{aligned}$$

where $\tilde{m}_i'$, m_i, $\tilde{b}_i'$, and b_i, $1 \leq i \leq n$, are 2×1 column vectors. The first column of $\tilde{B}'$ is computed in a different manner from columns $2, ..., n$ since it defines the transformation G.

Let

$$\tilde{b}'_1 = \begin{pmatrix} \tilde{b}'_{11} \\ 0 \end{pmatrix} = fl(\tilde{G}b_1).$$

Then

$$\tilde{b}'_{11} = fl(2^{\alpha}w_1b_{11}^2 + 2^{\alpha}w_2b_{21}^2) = 2^{\alpha}fl(w_1b_{11}^2 + w_2b_{21}^2).$$

Assume that the computations in $\tilde{b}'_{11}$ are associated as follows :

$$(w_1b_{11})b_{11} + (w_2b_{21})b_{21}.$$

This is a sensible assumption since w_1b_{11} and w_2b_{21} must be computed to form G.

In that case,

$$\tilde{b}'_{11} = fl(\tilde{g}_{11}b_{11} + \tilde{g}_{12}b_{21}) = \tilde{g}_{11}b_{11}(1+\eta_1) + \tilde{g}_{12}b_{21}(1+\eta_2),$$

where $|\eta_1|, |\eta_2| \leq 2\mu + O(\mu^2)$. As

$$0 = (\,\tilde{g}_{21} \quad \tilde{g}_{22}\,)\begin{pmatrix} b_{11} \\ b_{21} \end{pmatrix}$$

is an exact relation, there is no error in the insertion of the zero. Thus,

$$fl(\tilde{G}b_1) = \begin{pmatrix} \tilde{g}_{11}(1+\eta_1) & \tilde{g}_{12}(1+\eta_2) \\ \tilde{g}_{21} & \tilde{g}_{22} \end{pmatrix} b_1$$

which implies

$$K'fl(\tilde{G}K^{-1}m_1) = \begin{pmatrix} \tilde{\gamma}(1+\eta_1) & \tilde{\sigma}(1+\eta_2) \\ -\sigma & \gamma \end{pmatrix} m_1,$$

and the annihilation of the $(2,1)$ entry of M is exact. Hence,

$$K'(fl(\tilde{G}K^{-1}m_1) - \tilde{G}K^{-1}m_1) = \begin{pmatrix} \tilde{\gamma}\eta_1 & \tilde{\sigma}\eta_2 \\ 0 & 0 \end{pmatrix} m_1,$$

from which it can be concluded that

$$\begin{aligned} \|K'(fl(\tilde{G}K^{-1}m_1) - \tilde{G}^{-1}m_1)\| &\leq 2\mu\|K'\tilde{G}K^{-1}\|\|m_1\| + O(\mu^2) \leq 2\mu(1+\mu)\|m_1\| + O(\mu^2) \\ &= 2\mu\|m_1\| + O(\mu^2). \end{aligned}$$

For the remaining columns, $\tilde{G}b_k$, $1 \leq k \leq n$, is just a matrix vector multiplication. Therefore,

$$fl(\tilde{G}b_k) = \begin{pmatrix} \tilde{g}_{11}(1+\eta_{11}) & \tilde{g}_{12}(1+\eta_{12}) \\ \tilde{g}_{21}(1+\eta_{21}) & \tilde{g}_{22}(1+\eta_{22}) \end{pmatrix} b_k,$$

where $|\eta_{ij}| \leq 2\mu + O(\mu^2)$, $1 \leq i,j \leq 2$. The η_{ij} depend on b_k. Consequently,

$$K'fl(\tilde{G}K^{-1}m_k) = \begin{pmatrix} \tilde{\gamma}(1+\eta_{11}) & \tilde{\sigma}(1+\eta_{12}) \\ -\sigma(1+\eta_{21}) & \gamma(1+\eta_{22}) \end{pmatrix} m_k,$$

implying

$$\|K'(fl(\tilde{G}K^{-1}m_k) - \tilde{G}K^{-1}m_k)\| \leq 2\mu\|K'\tilde{G}K^{-1}\|\|m_k\| + O(\mu^2).$$

From

$$\|K'\tilde{G}K^{-1}\| \leq 1+\mu$$

it follows

$$\begin{aligned}\|K'(fl(\tilde{G}K^{-1}m_k) - \tilde{G}K^{-1}m_k)\| &\leq 2\mu(1+\mu)\|m_k\| + O(\mu^2) \\ &= 2\mu\|m_k\| + O(\mu^2), \qquad k = 2,\ldots,n.\end{aligned}$$

Therefore,

$$\begin{aligned}\|K'(fl(\tilde{G}K^{-1}M) - \tilde{G}K^{-1}M)\|_F &= \|K'(fl(\tilde{G}B) - \tilde{G}B)\| \\ &\leq 2\mu\|M\|_F + O(\mu^2)\end{aligned}$$

which along with (4.7) gives

$$\|\tilde{K}'(fl(\tilde{G}B) - \tilde{G}B)\|_F \leq (1+2.5\mu)2\mu\|M\|_F + O(\mu^2) = 2\mu\|M\|_F + O(\mu^2).$$

Employing the inequality in (4.6), one has

$$\|\delta M'\|_F = \|\tilde{M}' - HM\|_F \leq 5.5\mu\|M\|_F + O(\mu^2).$$

∎

The following two lemmata are adapted from [6]; they allow the construction of a bound for the backward error $\|\delta Y\|_F$ in equation (4.5).

Lemma 4.3. *Let H be a Givens transformation rotating a $2 \times n$ matrix M and implemented in such a way that*

$$\|fl(HM) - HM\|_F \leq \tau\|M\|_F + O(\mu^2),$$

where μ is the machine unit and $\tau = O(\mu)$. Then a transformation Q made up of $k \geq 1$ disjoint Givens transformations implemented in the same way and applied to $2k$ rows selected from the $m \times n$ matrix Y, will also satisfy

$$\|fl(QY) - QY\|_F \leq \tau\|Y\|_F + O(\mu^2).$$

In particular, this bound is independent of k and m.

Lemma 4.4. *If a sequence of Givens transformations can be written as a sequence of ν stages, where each stage consists of the simultaneous application of disjoint Givens transformations, then the final computed matrix obtained when this sequence of Givens transformations is applied to a given matrix Y will be the exact transformation of $Y + \delta Y$ where*

$$\|\delta Y\|_F \leq \tau\nu(1+\tau)^{\nu-1}\|Y\|_F + O(\mu^2).$$

Theorem 4.2. *The algorithm specified by equations (4.1) obtains an approximate solution to problem (4.4) which solves (4.5) exactly for some backward error matrix δY and backward error vector δz such that*

$$\|\delta Y\| \leq \|\delta Y\|_F \leq 5.5\mu\nu(1+5.5\mu)^{\nu-1}\|Y\|_F + O(\mu^2), \tag{4.8}$$

$$\|\delta z\| \leq 5.5\mu\nu(1+5.5\mu)^{\nu-1}\|z\| + O(\mu^2). \tag{4.9}$$

Proof. Simply combine the results of Theorem 4.1 and Lemmata 4.3 and 4.4.

∎

The factor $(1+5.5\mu)^{\nu-1}$ results from the fact that the computed transformations Q_k may not be exactly orthogonal. However, as pointed out by Wilkinson [16] p.138 this factor is seldom important enough to affect the stability.

5. Parallel Implementation

The systolic processor array will be designed for multiple linear least squares problems, where $1 \leq t$ different right side vectors are given,

$$\|D(Av_h^* - s_h)\| = \min_{v_h \in R^n} \|D(Av_h - s_h)\|, \qquad 1 \leq h \leq t.$$

The discussion will omit the backsubstitution, as a systolic array for it can be found in [13].

The following paragraphs will illustrate a systolic array for the triangular decomposition with corresponding modifications of the right hand side vectors, to be implemented in silicon with VLSI (Very Large Scale Integration), for instance.

Constraints imposed by technology and fabrication demand regular (and if possible planar) processor interconnections as well as processor communication on a nearest neighbour basis. The obvious choice of structure is a rectangular array of synchronously operating processors. Furthermore, to keep the chip area minimal, processors should perform no other than the elementary operations, addition, multiplication and divsion.

The constraint of *local* data exchange suggests a combination of pipelining and multiprocessing to achieve good processor utilisation and speed up in computation time. Pipelining is efficient for long matrices, which often possess a narrow dense band. The hardware should reflect the features of the matrix : a processor count proportional to the bandwidth, not the order of the matrix. Consequently, I/O occurs by codiagonals rather than rows or columns.

Matrices to be considered have a presumably narrow and dense band of width $w = p + q + 1$, $q \geq 0$ being the number of subdiagonals and $p \geq 0$ the number of superdiagonals.

Sameh and Kuck [15] proposed a simple startegy for elimination of subdiagonal elements which preserves the bandwidth w (i.e., annihilation of $a_{q+i,i}$ causes fill-in $a_{q+i-1,i+w-1}$) and removes elements by rotations in adjacent planes (that is, the elimination of $a_{q+i,i}$ takes place by rotating planes $q+i-1$ and $q+i$). Therefore, it adheres to regular data flow and local communication.

Banded matrices are reduced to (banded) triangular form through removal of elements by subdiagonals, starting from without toward the main diagonal while proceeding from top to bottom within a subdiagonal. Formally. if $a_{q+1,1}$ is removed at time $t = 1$ then $a_{q+i-2j,i-j}$ is removed at time $t = i$, $0 \leq i \leq n+q-1$, $\max\{0, i-n, \lceil (q+i-m)/2 \rceil\} \leq j \leq \min\{i-1, q-1\}$ (note that during the removal of a subdiagonal each of its rows, except the first and the last, is modified by two successive rotations). In the example below for $q = 3$, $p = 2$, $m = 9$ and $n = 8$, matrix entry $(q+i-2j, i-j)$ contains the time of removal of subdiagonal element $a_{q+i-2j,i-j}$,

$$\begin{bmatrix}
x & x & x & & & & & \\
3 & x & x & x & & & & \\
2 & 4 & x & x & x & & & \\
1 & 3 & 5 & x & x & x & & \\
 & 2 & 4 & 6 & x & x & x & \\
 & & 3 & 5 & 7 & x & x & x \\
 & & & 4 & 6 & 8 & x & x \\
 & & & & 5 & 7 & 9 & x \\
 & & & & & 6 & 8 & 10
\end{bmatrix}$$

Algorithm 5.1 below, which is by and large the same as in [11] for Fast Givens transformations, describes a parallel implementation of equations (4.1) for $m > n$. The matrix A is overwritten

with its upper triangular factor R and the original weight matrix $W = W^{(0)}$ with W'. For each subdiagonal $q-j+1$, $1 \le j \le q$, the auxiliary variable Δ_j represents a weight matrix element after the first of two plane rotations during its elimination. $\Delta_{h,j}$ has a similar function for the right hand side vectors s_h, $1 \le h \le t$, since those are input by columns - as opposed to elements of A which are addressed by diagonals (let $s_{h,i}$ denote the jth element of s_h). $\Delta_j^{(0)}$ holds elements of the original weight matrix $W^{(0)}$ which are needed for automatic rescaling. In a hardware implementation, Δ_j, $\Delta_j^{(0)}$ and $\Delta_{h,j}$, $1 \le j \le q$, $1 \le h \le t$, correspond to registers.

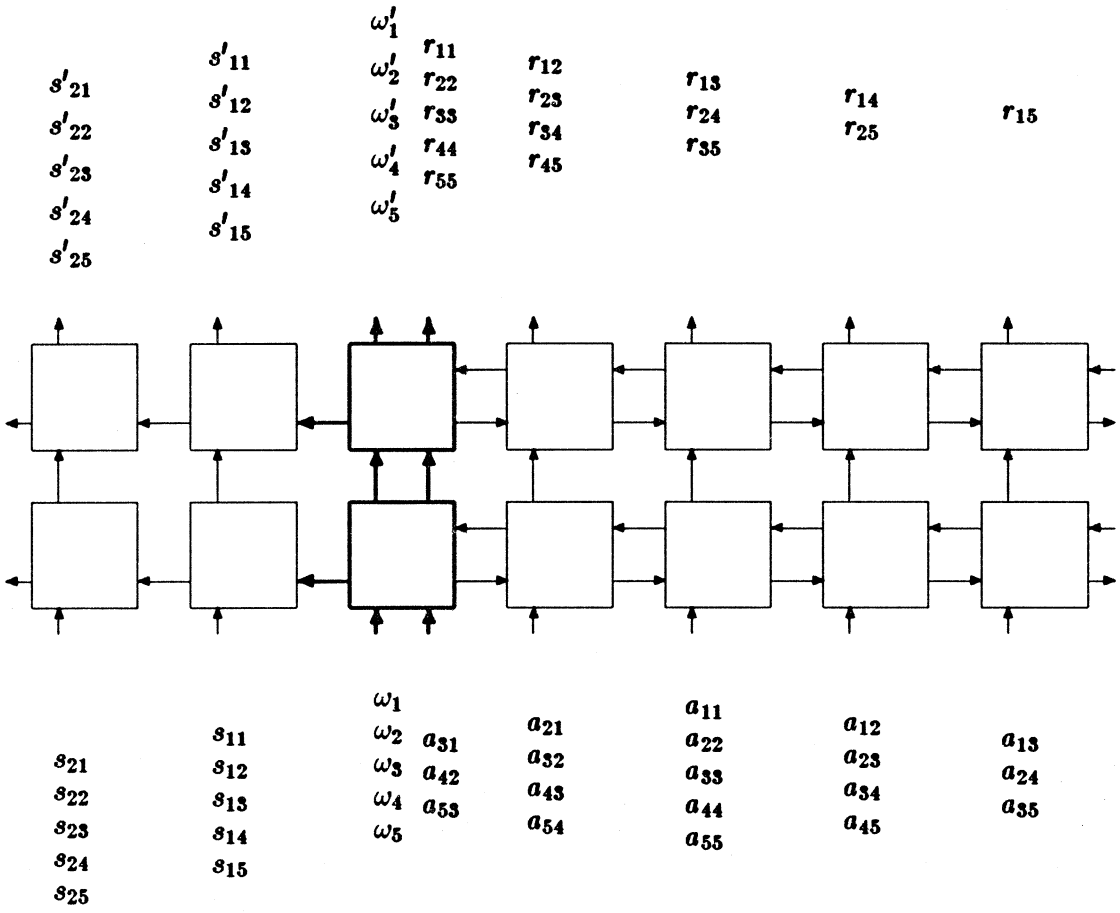
Algorithm 5.1.

$$
\begin{aligned}
&\textit{Initially, } \Delta_j = \Delta_j^{(0)} = w_j^{(0)}; \quad \Delta_{h,j} = s_{h,j};\\
&\quad \textit{for } i = 1 \ldots n+q-1 \textit{ do sequentially,}\\
&\quad\quad \textit{for } j = \max\{0, i-n, \lceil (q+i-m)/2 \rceil\} \ldots \min\{i-1, q-1\} \textit{ do in parallel,}\\
&\quad\quad\quad \rho_j = \Delta_j \left(a_{q+i-1-2j,i-j}\right)^2 + w_{q+i-2j} \left(a_{q+i-2j,i-j}\right)^2,\\
&\quad\quad\quad \textit{Compute } \alpha \textit{ and } \beta \textit{ according to algorithm } 3.1 \textit{ where}\\
&\quad\quad\quad\quad \varsigma_j = 1/\rho_j, \quad \xi_j = \Delta_j w_{q+i-2j} \varsigma_j, \qquad [\mathbf{i-2j, 1}]\\
&\quad\quad\quad\quad G_{i,j} = \begin{pmatrix} 2^{\alpha} \Delta_j a_{q+i-1-2j,i-j} & 2^{\alpha} w_{q+i-2j} a_{q+i-2j,i-j},\\ -2^{\beta} a_{q+i-2j,i-j} & 2^{\beta} a_{q+i-1-2j,i-j} \end{pmatrix},\\
&\quad\quad\quad\quad a_{q+i-1-2j,i-j} = 2^{\alpha} \rho_j, \quad a_{q+i-2j,i-j} = 0,\\
&\quad\quad\quad\quad w_{q+i-1-2j} = 2^{-2\alpha} \varsigma_j, \quad \Delta_j = 2^{-2\beta} \xi_j, \quad \Delta_j^{(0)} = w_{q+i-2j}^{(0)},\\
&\quad\quad\quad\quad \textit{for } l = 1 \ldots w-1 \textit{ do sequentially by pipelining } G_{i,j}, \qquad [\mathbf{i-2j, l+1}]\\
&\quad\quad\quad\quad\quad \begin{pmatrix} a_{q+i-1-2j,i-j+l} \\ a_{q+i-2j,i-j+l} \end{pmatrix} = G_{i,j} \begin{pmatrix} a_{q+i-1-2j,i-j+l} \\ a_{q+i-2j,i-j+l} \end{pmatrix};\\
&\quad\quad\quad\quad \textit{for } h = 1 \ldots t \textit{ do sequentially by pipelining } G_{i,j}, \qquad [\mathbf{h, i-2j}]\\
&\quad\quad\quad\quad\quad \begin{pmatrix} s_{h,q+i-1-2j} \\ \Delta_{h,j} \end{pmatrix} = G_{i,j} \begin{pmatrix} \Delta_{h,j} \\ s_{h,q+i-2j} \end{pmatrix}.
\end{aligned}
$$

The systolic array that implements algorithm 5.1 consists of q processor vectors of $w+t$ processors each. Counting fom top to bottom, vector j is responsible for eliminating subdiagonal j, $1 \le j \le q$. A, W and s_h are input to the bottom of the array while R, W' and s_h' are available at the top, see Figure 1. Within vector j processors are numbered from left to right, $(j,-h), \ldots, (j,-1), (j,1), \ldots (j,w)$ of which the first h apply $G_{i,j}$ to elements of s_h and the remaining w to elements of A. Consequently, $G_{i,j}$ is computed in processors $(j,1)$ and thereafter pipelined to the left across processors $(j,-1), \ldots (j,-h)$ and at the same time to the right across $(j,2), \ldots (j,w)$.

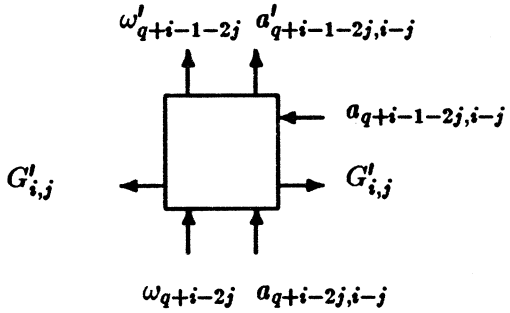
Subject to the three different sets of equations, three different types of processors, sketched in Figure 2, accomplish the triangularisation. For $1 \le j \le q$, processors $(j,1)$ compute equations $[i-2j,1]$, processors $(j,l+1)$ compute $[i-2j,l+1]$, $1 \le l \le w-1$, while processors $(j,-h)$ determine $[h,i-2j]$, $1 \le h \le t$. When idle or receiving no input, processors $(j,1)$ generate identity rotations. The default input for matrix elements is 0.

Data flow in and out of processor (j,l), $1 \le l \le w$ occurs by codiagonals so that succeeding codiagonal elements enter and leave a processor every other time step. Input and output for processors $(j,-h)$, $1 \le h \le t$, is by columns, again with successive column elements entering and exiting every other time step. A partial execution trace for the example $p = 3$, $q = t = 2$ can be found in Figure 3.



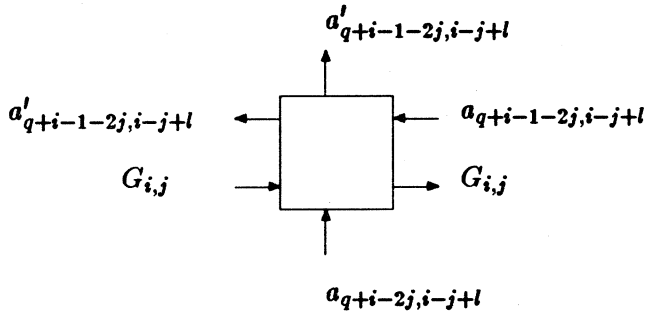
$$\omega_i = (w_i^{(0)}, w_i), \quad \omega'_i = (w_i^{(0)}, w'_i)$$

Figure 1: Processor Array for $q = p = t = 2$.

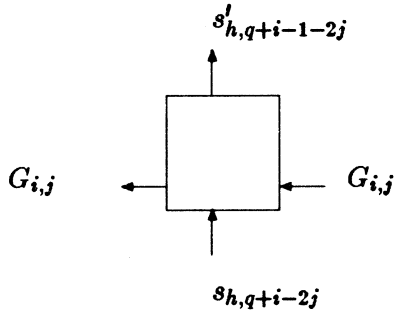


$$\omega_{q+i-1-2j} = (w^{(0)}_{q+i-1-2j}, w_{q+i-1-2j}), \quad \omega_{q+i-2j} = (w^{(0)}_{q+i-2j}, w_{q+i-2j}).$$

(a) Equations $[i-2j, 1]$.



(b) Equations $[i-2j, l]$.



(c) Equations $[l, i-2j]$.

Primes indicate quantities computed or altered by a processor.

Figure 2: Processors.

In particular, $s_{h,1}$ enters processor $(q,-h)$ $q-h$ steps before a_{11} enters $(q,q+1)$ and leaves processor $(1,-h)$ h steps after a_{11} leaves $(1,1)$. The weight matrix elements $\omega_1 = (w_1^{(0)}, w_1)$ enter processor $(q,1)$ $q-h+1$ steps before a_{11} enters the array while they exit $(1,1)$ together with a_{11}. Starting the input with ω_1 at $t=1$ the computation time comes to

$$2m + 3(q-1) + t + 1.$$

6. Conclusions

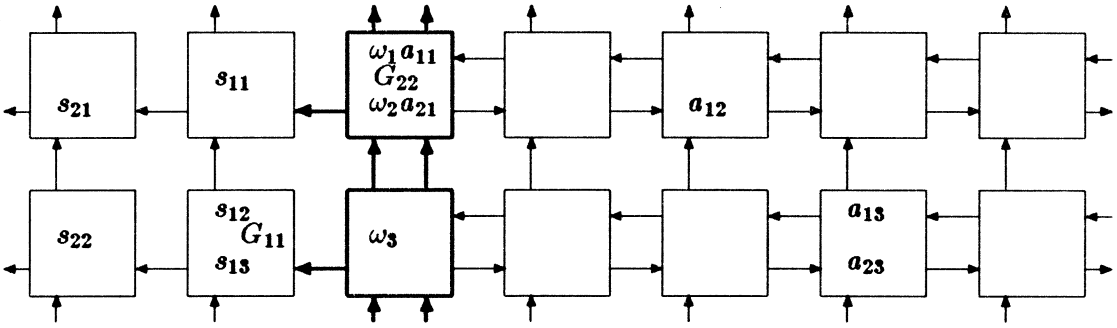
A class of scaled rotations for the parallel solution to weighted multiple linear least squares problems of finding v_i^* such that

$$\|D(Av_i^* - s_i)\| = \min_{v_i \in R^n} \|D(Av_i - s_i)\|, \qquad 1 \le i \le t,$$

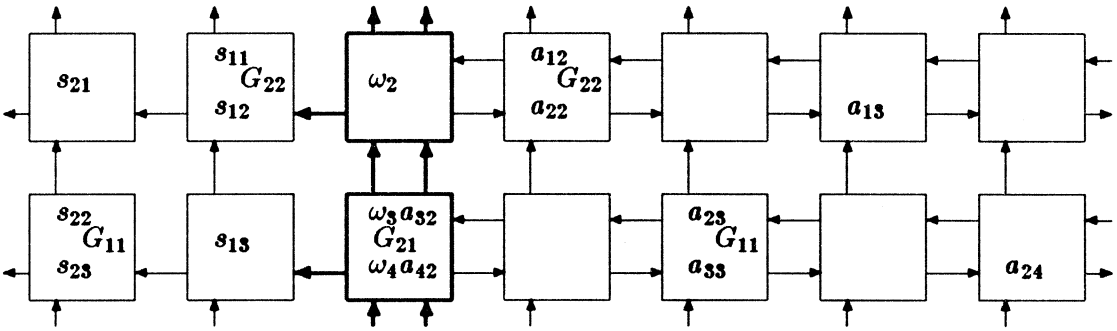
for $m \times n$, w-banded matrices, $m > n$, in time $O(m+t)$ has been presented.

The planar orthogonally connected systolic array for this problem consists of $O(w^2 + tw)$ processing elements which perform only the elementary operations $\{+, *, /\}$. To satisfy VLSI requirements, data exchange among processors proceeds on a nearest neighbour basis, and square root computations are avoided.

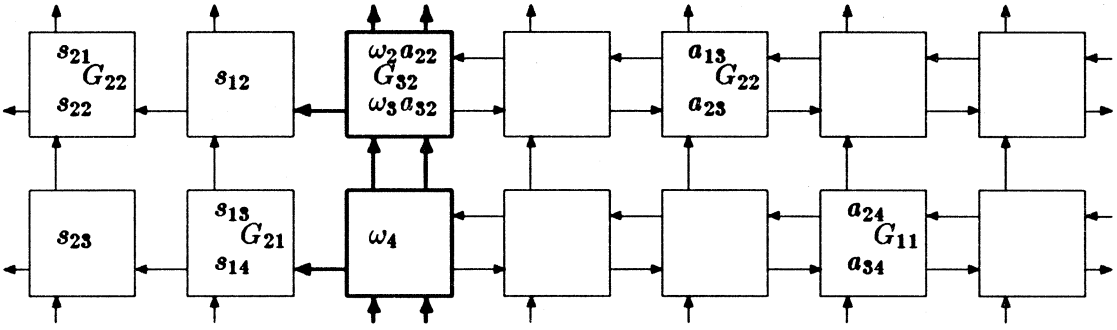
Proper scaling, which has been a persistent problem in implementations of fast scaled Givens rotations, prevents under- and overflow, obviating the need to pivot. A backward error analysis has shown the algorithm to be as stable as any available.



(a) $t = 6$



(b) $t = 7$



(c) $t = 8$

$$\omega_i = (w_i^{(0)}, w_i)$$

Figure 3: Partial Execution Trace for $p = 3$, $q = t = 2$.

References

[1] Ahmed, H.M., Delosme, J.M. and Morf, M., *Highly Concurrent Structures for Matrix Arithmetic and SignalProcessing,* Computer, 15(1982), pp. 65–82.

[2] Bareiss, E.H., *Numerical Solution of the Weighted Least Squares Problem byG-Transformations,* Tech. Report 82-03-NAM-03 , Department of Electrical Engineering and Computer Science,Northwestern University, Evanston, IL 60201, 1982.

[3] Bojanczyk, A., Brent, R.P. and Kung, H.T., *Numerically Stable Solution of Dense Systems of Linear EquationsUsing Mesh-Connected Processors,* SIAM J. Sci. Stat. Comp., 5(1984), pp. 95–104.

[4] Businger, P.A. and Golub, G.H., *Linear Least Squares Solutions by Householder Transformations,* Num. Math., 7(1965), pp. 269–78.

[5] Gentleman, W.M., *Least Squares Computations by Givens Rotations without Square Roots,* J. Inst. Math. Appl., 12(1973), pp. 329–36.

[6] ————, *Error Analysis of QR Decompositions by Givens Transformations,* Lin. Alg. Appl., 10(1975), pp. 189–97.

[7] Gentleman, W.M., and Kung, H.T., On Stable Parallel Linear System Solvers, *Proc. SPIE Real Time Signal Processing IV,* pp. 19–26, SPIE, San Diego, CA, 1981.

[8] Golub, G.H. and van Loan, C.F, *Matrix Computations,* The Johns Hopkins Press, Baltimore, MD, 1973.

[9] Hammarling, S., *A Note on Modifications to the Givens Plane Rotation,* J. Inst. Math Appl., 13(1974), pp. 215–8.

[10] Heller, D.E. and Ipsen, I.C.F., *Systolic Networks for Orthogonal Decompositions,* SIAM J. Stat. Sci. Comp., 4(1983), pp. 261–9.

[11] Ipsen, I.C.F., *A Parallel QR Method Using Fast Givens' Rotations,* Research Report 299 , Department of Computer Science, Yale University,New Haven, CT 06520, 1984.

[12] Johnsson, L., A Computational Array for the QR-Method, *Proc. Conf. Advanced Research in VLSI,* pp. 123–129, MIT, 1982.

[13] Kung, H.T. and Leiserson, C.E., Systolic Arrays (for VLSI), *Sparse Matrix Proceedings,* pp. 256–82, SIAM, Philadelphia, PA, 1978.

[14] Lawson, C.L. and Hanson, R.J., *Solving Least Squares Problems,* Prentice Hall, Englewood Cliffs, NJ, 1974.

[15] Sameh, A.H. and Kuck, D.J., *On Stable Parallel Linear System Solvers,* JACM, 25(1978), pp. 81–91.

[16] Wilkinson, J.H., *The Algebraic Eigenvalue Problem,* Oxford University Press, London, 1965.