A Triangular Processor Array for Computing the Singular Value Decomposition

Franklin T. Luk

TR 84-625
July 1984

Department of Computer Science
Cornell University
Ithaca, New York 14853

*The work of the author was supported in part by the National Science Foundation under grant MCS-8213718.

A Triangular Processor Array for Computing the Singular Value Decomposition

Franklin T. Luk

Department of Computer Science
Cornell University
Ithaca, New York 14853
U.S.A.

ABSTRACT

A triangular processor array for computing a singular value decomposition (SVD) of an $m \times n$ $(m \geq n)$ matrix is proposed. A Jacobi-type algorithm is used to first triangularize the given matrix and then diagonalize the resultant triangular form. The requirements are $O(m)$ time and $\frac{1}{4}\ n^2 + O(n)$ processors.

Keywords and Phrases: Systolic arrays, QR-decomposition, singular value decomposition, Jacobi-type methods, real-time computation, VLSI.

The work of the author was supported in part by the National Science Foundation under grant MCS-8213718.

1. Introduction

Two important factorizations of a given real $m \times n$ $(m \geq n)$ matrix A are its QR-decomposition (QRD):

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix}, \tag{1.1}$$

where the matrix Q $(m \times m)$ is orthogonal and R $(n \times n)$ is upper triangular, and its singular value decomposition (SVD):

$$A = U\Sigma V^T , \tag{1.2}$$

where the matrices U $(m \times m)$ and V $(n \times n)$ are orthogonal, and the matrix Σ $(m \times n)$ is nonnegative diagonal. For details about these decompositions see Golub and Luk[10], Golub and Van Loan[11] and Dongarra et al.[7]. Recently, there has been much interest in their computation using systolic arrays, partially due to the needs of real time signal processing (Bromley and Speiser[5]). QR-arrays are presented in Ahmed, Delosme and Morf[1], Bojanczyk, Brent and Kung[2], Gentleman and Kung[9], Heller and Ipsen[13], Johnsson[14], Luk[16] and Sameh[19]; SVD arrays in Brent and Luk[3], Brent, Luk and Van Loan[4], Finn, Luk and Pottle[8], Heller and Ipsen[12] and Schreiber[20].

The fastest SVD algorithm (effectively linear time) is the Jacobi procedure described in [4]. Jacobi-type methods are natural for matrix computations on processor arrays: they have been proposed for the symmetric eigenvalue decomposition by Brent and Luk[3], for the QR-decomposition by Luk[16], and for the Schur decomposition by Stewart[21]. In addition, the methods used for finding eigenvalues and singular values on the first parallel computer, the ILLIAC IV, were also of the Jacobi type (Luk[15] and Sameh[18]). Unfortunately, Jacobi-SVD algorithms are applicable only to square matrices. For a rectangular matrix A, an obvious strategy is to first compute its QR-decomposition and then apply the SVD procedure to the resultant square matrix R. This approach is particularly suitable for the case where $m >> n$ (cf. Chan[6]). However, the interfacing of different arrays can be a serious problem. In fact, all but one (viz. Luk[16]) of the afore-mentioned QR-arrays are different from the square processor array described in [4]

The purpose of this paper is to develop new QRD and SVD algorithms that can be implemented on one programmable "triangular" processor array. The QRD algorithm is presented in §2 and the SVD algorithm in §3. A distinctive feature of the latter algorithm is that it preserves the triangular structure of R. To compute an $m \times n$ SVD, our two algorithms require $\frac{1}{4} n^2 + O(n)$ processors and $O(m)$ time, which compare favorably with the serial requirement of $O(mn^2)$ time. If singular vectors were desired, we could extend the "triangular" array to a "rectangular" one that is composed of $\frac{1}{2} mn + O(m)$ processors. The required time remains $O(m)$. Section 4 contains a discussion and some numerical results.

2. A QRD Algorithm

In this section we propose a parallel algorithm for triangularizing an $m \times n$ matrix A. Our algorithm is similar to that of Gentleman and Kung[9]. A major difference is that we perform 2×2 QRDs, whereas they annihilate individual elements. Our "triangular" processor array is quite alike in overall structure to the "square" Schur decomposition array of Stewart[17,21].

The basic transformation for this (and the next) section is a plane rotation

$$J(\theta) \equiv \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} . \tag{2.1}$$

It is chosen such that

$$J(\theta)\begin{pmatrix} w & x \\ y & z \end{pmatrix} = \begin{pmatrix} p & q \\ 0 & r \end{pmatrix} . \tag{2.2}$$

If $y=0$ we choose $\theta=0$, otherwise we use the formulas:

$$\rho = \frac{w}{y} \equiv \operatorname{ctn}\theta \ , \ \sin\theta = \frac{\operatorname{sign}(\rho)}{\sqrt{1+\rho^2}} \ , \ \cos\theta = \rho \sin\theta \ .$$

Our computational network consists of a "triangular" array of $\frac{1}{4} n^2 + O(n)$ processors and a triangular array of storage cells. Each cell (denoted by ‡) contains a 2×1 matrix, and each processor (denoted by (p,q)) is associated with a 2×2 matrix contained in the two ‡ cells at its two upper corners. Exceptions are processors on the right boundary: they each associates with only one cell. We refer to the stored $2n \times 2n$ "Hessenberg" matrix as H. See Figures 1 and 2 for illustrations of H and the network. Let us denote the ith row of A by a_i^T and define a $(2n-1) \times 2n$ matrix Z by

$$Z = (e_1 , e_2 , \cdots , e_{2n-1} , 0) , \tag{2.3}$$

where e_j denotes the jth coordinate vector. Premultiplying H by Z will thus annihilate the last row of H.

Our algorithm has two stages: the first when the matrix A is fed into into array, and the second when the "Hessenberg" matrix is triangularized. We define one time step as the amount of time required by one processor row to do a QRD. The first stage lasts from step 1 to m, and the second from step $m+1$ to $m+n-1$. The matrix H is initialized to be zero. A typical step

(say i, $1 \leq i \leq m$) of the first stage starts with the ith row of A entering the array, pushing down each row of H by one position and evicting its last row (consisting of all zeros). Step i ends with each processor row computing an appropriate QRD: the diagonal processor determining $J(\theta)$ and the others applying it. A typical step (say $m+j$, $1 \leq j \leq n-1$) of the second stage starts with each storage cell on rows $j, \cdots, n$ passing its bottom element to its southern neighbor. Consequently, each cell on row j contains only one element and is denoted as a $\times$ cell (see Figures 3 and 4). Step $m+j$ ends after processor rows $j+1, \cdots, n$ complete the appropriate QRDs. The computing network completes its task after $m+n-1$ time steps. All cells have become $\times$ cells, and they store the desired triangular matrix R. We may "rigorize" the procedure as follows:

Algorithm QRD.

$H \leftarrow 0$;

for $i = 1, 2, \cdots, m$ do
    begin
    $H \leftarrow \begin{pmatrix} a_i^T \\ ZH \end{pmatrix}$;
    Processors perform QRDs { last row of H become zero }
    end;

for $j = 1, 2, \cdots, n-1$ do
    begin
    Cells on rows $j, \cdots, n$ pass their bottom elements to their southern neighbors;
    Cells on row j become $\times$ cells;
    Processors on rows $j+1, j+2, \cdots, n$ perform QRDs
    end;

Cells on row n eject their bottom elements and become $\times$ cells. □

To avoid the broadcast of rotation parameters, we need to stagger both data entries and processor operations. The scheme, illustrated in Figures 5 and 6, is well known in the study of parallel algorithms (cf. [3,4,16,17,21]). It should be clear why element $a_{i+1,j}$ is annihilated one time step after element a_{ij}. The reason for a two-step delay between the annihilations of $a_{i,j+1}$ and a_{ij} is that, since 2×2 QRDs are performed, the arrival of element $a_{i+1,j+1}$ must be awaited.

The whole procedure (without broadcast) requires $m+2n-2$ time steps.

Computing Q is a difficult task. Except for Luk[16], it has not been written how this orthogonal matrix may be explicitly determined. Since R results from applying the rotations to A:

$$Q^T A = \begin{bmatrix} R \\ 0 \end{bmatrix},$$

the matrix Q^T is obtainable from applying the same rotations to the $m \times m$ identity matrix I_m. Yet this may be undesirable due to the requirement of $O(m^2)$ processors. Instead, we may extend our "triangular" network to a "rectangular" one, so as to determine an $n \times m$ matrix Q_1^T satisfying

$$A = Q_1 R.$$

That is, Q_1 is composed of the leading m columns of Q. See Figure 7 for an illustration of this "rectangular" array of ½ $mn + O(n)$ processors. Essentially, the matrix I_m enters the array in exactly the same fashion as A, and rotations for A are also applied to I_m. Hence

$$Q^T I_m = Q^T \equiv \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix},$$

but the network accumulates only the top n rows of Q^T.

FIG. 1. *QRD computational network at time step* $t \leq m$ $(n=5)$.

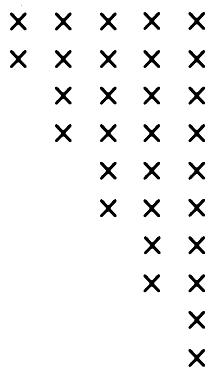

FIG. 2. *Matrix H in network at time step* $t \leq m$ $(n=5)$.

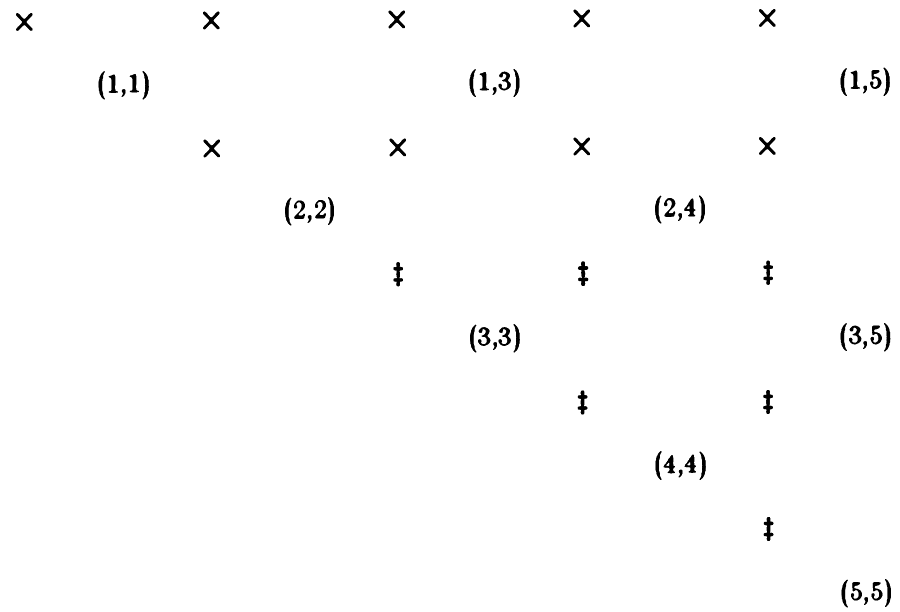

FIG. 3. *QRD computational network at time step* $t = m + 2$ $(n = 5)$.

$$
\begin{array}{ccccc}
\times & \times & \times & \times & \times \\
 & \times & \times & \times & \times \\
 & & \times & \times & \times \\
 & & \times & \times & \times \\
 & & & \times & \times \\
 & & & \times & \times \\
 & & & & \times \\
 & & & & \times
\end{array}
$$

FIG. 4. *Matrix (becoming triangular) in network at time step* $t = m + 2$ $(n = 5)$.

$$
\begin{array}{ccccc}
\cdot & \cdot & \cdot & \cdot & \cdot \\
a_{51} & a_{52} & a_{43} & a_{44} & a_{35} \\
a_{41} & a_{42} & a_{33} & a_{34} & a_{25} \\
a_{31} & a_{32} & a_{23} & a_{24} & a_{15} \\
a_{21} & a_{22} & a_{13} & a_{14} & \\
a_{11} & a_{12} & & & \\
\ddagger & \ddagger & \ddagger & \ddagger & \ddagger \\
 & \ddagger & \ddagger & \ddagger & \ddagger \\
 & & \ddagger & \ddagger & \ddagger \\
 & & & \ddagger & \ddagger \\
 & & & & \ddagger
\end{array}
$$

FIG. 5. *Data flow into the array* ($n=5$).

×	×	×	×	×
2	×	×	×	×
3	5	×	×	×
4	6	8	×	×
5	7	9	11	×
6	8	10	12	14
7	9	11	13	15

FIG. 6. *Order of annihilations for a* 7×5 *matrix.*

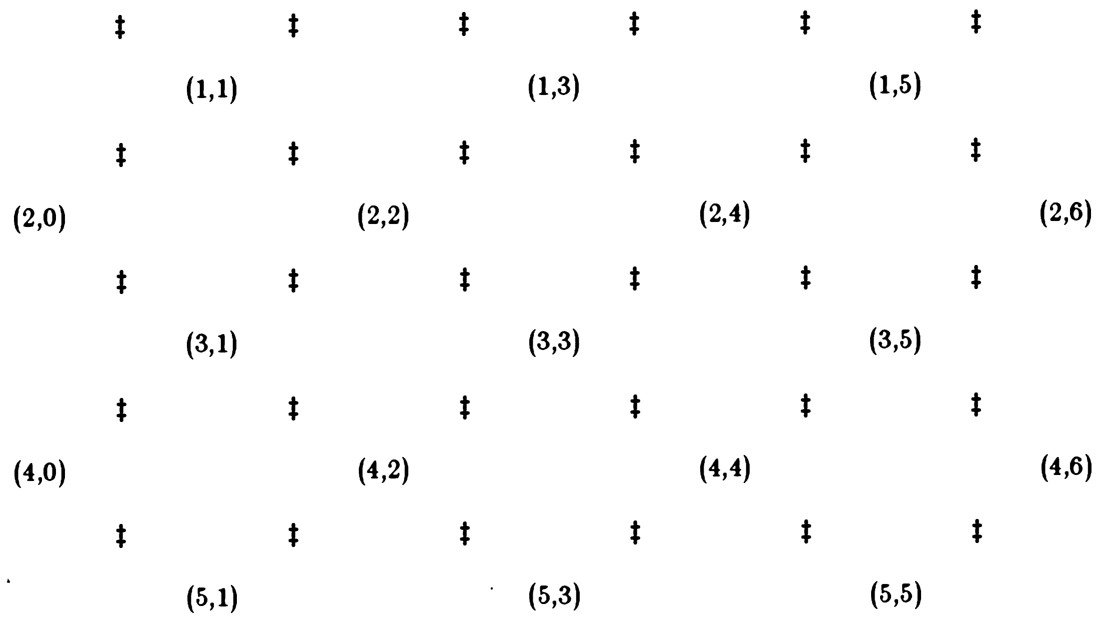

FIG. 7. *QRD computational network at time step* $t \leq m$
($m=6$, $n=5$, Q_1^T *is accumulated*).

3. An SVD Algorithm

In this section we present an SVD algorithm for an $n \times n$ upper triangular matrix R. This problem has been well studied in [4], where it is pointed out that one should merely reduce R to diagonal form and not worry if all the diagonal elements were nonnegative and ordered. We assume the same approach here. Our SVD algorithm has the distinction that it preserves the triangular form of R, an advantage not shared by the algorithm in [4].

The basic problem again concerns a 2×2 matrix. Plane rotations $J(\theta)$ and $K(\phi)$ are used to diagonalize a given upper triangular matrix:

$$J(\theta)^T \begin{bmatrix} w & x \\ 0 & z \end{bmatrix} K(\phi) = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} . \tag{3.1}$$

The two-stage procedure recommended in [4] is adopted. First, find a rotation $S(\psi)$ that symmetrizes B:

$$S(\psi)^T \begin{bmatrix} w & x \\ 0 & z \end{bmatrix} = \begin{bmatrix} p & q \\ q & r \end{bmatrix} . \tag{3.2}$$

If $x=0$ we choose $\psi=0$, otherwise we compute

$$\rho = \frac{w+z}{x} \equiv \text{ctn}\psi \ , \ \sin\psi = \frac{\text{sign}(\rho)}{\sqrt{1+\rho^2}} \ , \ \cos\psi = \rho \sin\psi \ .$$

Second, diagonalize the result:

$$K(\phi)^T \begin{bmatrix} p & q \\ q & r \end{bmatrix} K(\phi) = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} . \tag{3.3}$$

Suppose $q \neq 0$ (else choose $\phi=\pi/2$). It is well known that $t \equiv \tan\phi$ satisfies the quadratic equation

$$t^2 + 2\rho t - 1 = 0 , \tag{3.4}$$

where

$$\rho = \frac{r-p}{2q} \equiv \text{ctn} 2\phi .$$

The two solutions to (3.4) are

$$t = -\mathrm{sign}(\rho)\,[\ |\rho| + \sqrt{1+\rho^2}\ ]\ ,\ \cos\phi = \frac{1}{\sqrt{1+t^2}}\ ,\ \sin\phi = t\ \cos\phi \tag{3.5}$$

and

$$t = \frac{\mathrm{sign}(\rho)}{|\rho| + \sqrt{1+\rho^2}}\ ,\ \cos\phi = \frac{1}{\sqrt{1+t^2}}\ ,\ \sin\phi = t\ \cos\phi\ . \tag{3.6}$$

The angle ϕ associated with (3.5) is the larger of the two possibilities; it satisfies $\pi/4 \le |\phi| < \pi/2$. We shall choose this larger angle because we want to perform "outer rotations", recommended by Stewart[21] for his Schur decomposition array. If the given matrix were diagonal ($x=0$) then $\psi=0$ and $\phi=\pi/2$, so that (3.1) becomes

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} w & 0 \\ 0 & z \end{pmatrix} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} = \begin{pmatrix} z & 0 \\ 0 & w \end{pmatrix} .$$

Finally, $J(\theta)$ is given by

$$J(\theta)^T = K(\phi)^T S(\psi)^T\ ,$$

i.e., $\theta = \phi + \psi$. The general transformation is

$$T_{ij}\ :\ R \leftarrow J_{ij}^T R\ K_{ij}\ , \tag{3.7}$$

where J_{ij} and K_{ij} are rotations in the (i,j) plane chosen to annihilate the (i,j) and (j,i) elements of R. The pivot block will be restricted to contiguous diagonal elements, so as to preserve the upper triangular structure of R. Defining

$$off(R) \equiv \sum_{i<j} r_{ij}^2\ ,$$

we see that the transformation $T_{i,i+1}$ will produce a triangular matrix $\hat{R}$ satisfying

$$off(\hat{R}) = off(R) - r_{i,i+1}^2\ .$$

Our new algorithm is simply

Algorithm SVD.
do until convergence
begin
for $i = 1, 3, \cdots$ (i odd) do $R \leftarrow J_{i,i+1}^T \ R \ K_{i,i+1}$;
for $i = 2, 4, \cdots$ (i even) do $R \leftarrow J_{i,i+1}^T \ R \ K_{i,i+1}$
end. □

By convergence we mean that the parameter $off(R)$ has fallen below some prechosen tolerance. However, it is difficult to monitor $off(R)$ in the settings of parallel computations. We propose that iterations be stopped after a sufficiently large number, say ten, of sweeps (cf. [4] and Table 1). By one sweep we mean a group of $n(n-1)/2$ transformations.

Our Jacobi-SVD algorithm and related "triangular" computational network are taken from Stewart's Jacobi-Schur algorithm and related "square" network[21] (the "lower triangular" part of the network is discarded). Each processor, denoted by (p,q), is associated with a 2×2 matrix stored in its four neighboring cells (denoted by $\times$); the boundary processors are associated with fewer than four nonzero elements. The network is illustrated in Figure 8, and the reader is referred to [17,21] for details on the computing architecture, on the significance of "outer rotations", and on the staggering of computations to avoid broadcasting. An important point is that a sweep can be completed in $O(n)$ time. Unlike Stewart's algorithm, no example of nonconvergence is known of Algorithm SVD. Yet we must point out that computing the Schur decomposition is a much harder task than computing the SVD.

The $n \times n$ matrices U_R and V_R of singular vectors of R are easily computable (if desired) by accumulating the Jacobi rotations. We need to extend the "triangular" processor array to the "square" array of Stewart[21]. Both matrices are initialized to be the identity, and updated by the formulae:

$$U_R \leftarrow U_R \ J_{i,i+1} \quad \text{and} \quad V_R \leftarrow V_R \ K_{i,i+1} .$$

With $A = Q_1 R$ and $R = U_R \Sigma V_R^T$, an SVD of A is given by

$$A = (Q_1 U_R) \Sigma V_R^T .$$

We may wish to determine the explicit product $U_A \equiv Q_1 U_R$, which can be computed as follows:

$$U_A^T \leftarrow J_{i,i+1}^T U_A^T ,$$

with U_A^T initialized as Q_1^T. This computation can be performed on a "rectangular" array that is a natural extension of the "square" one[21]. See Figure 9 for an illustration and note the similarities between the processor arrays in Figures 7 and 9. The "rectangular" network requires ½ $mn + O(m)$ processors.

FIG. 8. *SVD computational network* (n=5).

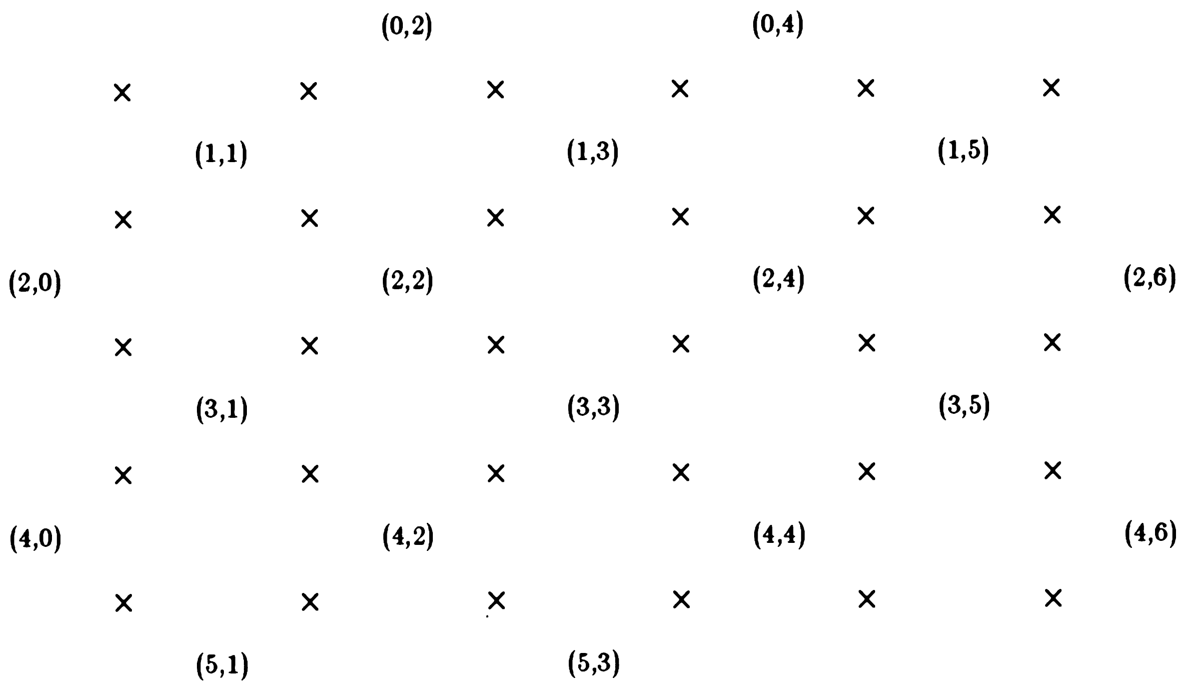

FIG. 9. *SVD computational network*
(m=6, n=5, U_A^T *and* V_R *are accumulated*).

4. Discussion

It is obvious from foregoing descriptions (§§ 2, 3 and Figures 1, 3, 8) that Algorithms QRD and SVD can both be implemented on one programmable "triangular" computational network. Compared to previous SVD works, our network has the advantage that no complex array interfacings are necessary, and its requirements of $O(m)$ time and $\frac{1}{4}\, n^2 + O(n)$ processors are unsurpassed. If singular vectors were desired, we would extend the network to a "rectangular" one composed of $\frac{1}{2}\, mn + O(m)$ processors (see §§ 2, 3 and Figures 7, 9). The required time remains $O(m)$. We know of no other network that can compute these vectors without using $O(m^2)$ processors.

Algorithm SVD was programmed on a VAX-11/780 at Cornell University. Double floating data types were used: each number was binary normalized, with an 8-bit signed exponent and a 57-bit signed fraction whose most significant bit was not represented. The machine precision was thus given by

$$\epsilon \approx 1.4 \times 10^{-17} .$$

Results of our experiments are presented in Table 1. The starting matrices R were $n \times n$ upper triangular, and their elements were chosen from a uniform distribution in the interval $(-1,1)$. The algorithm converged quadratically, confirming theoretical predictions, and only six or fewer sweeps were required for $n \leq 20$.

Acknowledgements. The author would like to thank W.M. Gentleman for a valuable suggestion.

Table 1. *Off(R) After Each Sweep of Algorithm SVD.*

n	Sweep					
	0	1	2	3	4	5
4	3.1e+00	4.4e-01	1.8e-03	5.5e-09	$< \epsilon$	
6	6.8e+00	1.6e-01	2.5e-04	7.2e-11	$< \epsilon$	
8	1.6e+01	3.3e+00	7.6e-03	3.2e-08	$< \epsilon$	
10	2.6e+01	4.3e+00	4.2e-01	1.1e-06	$< \epsilon$	
12	3.5e+01	1.4e+00	9.3e-02	1.6e-03	3.5e-10	$< \epsilon$
14	4.1e+01	2.4e+00	8.5e-02	6.7e-04	4.6e-09	$< \epsilon$
16	6.9e+01	1.6e+00	6.4e-02	3.5e-04	1.9e-09	$< \epsilon$
18	7.9e+01	7.3e+00	1.4e-01	3.2e-04	1.5e-08	$< \epsilon$
20	1.0e+02	8.6e+00	5.2e-01	1.9e-02	2.2e-05	1.2e-11

REFERENCES

[1] H.M. Ahmed, J.-M. Delosme and M. Morf, *Highly concurrent computing structures for matrix arithmetic and signal processing,* Computer, 15 no. 1 (Jan. 1982), pp. 65-82.

[2] A. Bojanczyk, R.P. Brent and H.T. Kung, *Numerically stable solution of dense systems of linear equations using mesh-connected processors,* SIAM J. Sci. Statist. Comput., 5 (1984), pp. 95-104.

[3] R.P. Brent and F.T. Luk, *The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays,* SIAM J. Sci. Statist. Comput., 5 (1984), to appear.

[4] R.P. Brent, F.T. Luk and C. Van Loan, *Computation of the singular value decomposition using mesh-connected processors,* J. VLSI Computer Systems, 1 (1984), to appear.

[5] K. Bromley and J.M. Speiser, *Signal Processing Algorithms, Architectures, and Applications,* Tutorial 31, SPIE 27th Annual Internat. Tech. Symp., San Diego, Calif., Aug. 1983.

[6] T.F. Chan, *An improved algorithm for computing the singular value decomposition,* ACM Trans. Math. Softw., 8 (1982), pp. 72-83.

[7] J.J. Dongarra, C.B. Moler, J.R. Bunch and G.W. Stewart, *LINPACK Users' Guide,* SIAM, Philadelphia, 1979.

[8] A.M. Finn, F.T. Luk, and C. Pottle, *Systolic array computation of the singular value decomposition,* Proc. SPIE Vol. 341, Real-Time Signal Processing V (1982), pp. 35-43.

[9] W.M. Gentleman and H.T. Kung, *Matrix triangularization by systolic arrays,* Proc. SPIE Vol. 298, Real Time Signal Processing IV (1981), pp. 19-26.

[10] G.H. Golub and F.T. Luk, *Singular value decomposition: applications and computations,* Trans. 22nd Conf. Army Mathematicians, ARO Report 77-1 (1977), pp. 577-605.

[11] G.H. Golub and C. Van Loan, *Advanced Matrix Computations,* The Johns Hopkins Press, Baltimore, 1984.

[12] D.E. Heller and I.C.F. Ipsen, *Systolic networks for orthogonal equivalence transformations and their applications,* Proc. 1982 Conf. on Advanced Research in VSLI, MIT, Cambridge, Mass., Jan. 1982, pp. 113-122.

[13] D.E. Heller and I.C.F. Ipsen, *Systolic networks for orthogonal decompositions,* SIAM J. Sci. Statist. Comput., 4 (1983), pp. 261-269.

[14] L. Johnsson, *A computational array for the QR-method,* Proc. 1982 Conf. on Advanced Research in VSLI, MIT, Cambridge, Mass., Jan. 1982, pp. 123-129.

[15] F.T. Luk, *Computing the singular-value decomposition on the ILLIAC IV,* ACM Trans. Math. Softw., 6 (1980), pp. 524-539.

[16] F.T. Luk, *A Jacobi-like algorithm for computing the QR-decomposition,* Tech. Report 84-612, Computer Science Dept., Cornell Univ., 1984.

[17] D.P. O'Leary and G.W. Stewart, *Data-flow algorithms for parallel matrix computations,* Tech. Report 1366, Computer Science Dept., Univ. of Maryland, 1984.

[18] A.H. Sameh, *On Jacobi and Jacobi-like algorithms for a parallel computer,* Math. Comput., 25 (1971), pp. 579-590.

[19] A.H. Sameh, *Solving the linear least squares problem on a linear array of processors,* Proc. Purdue Workshop Algorithmically-Specialized Computer Organ., W. Lafayette, Indiana, Sept. 1982.

[20] R. Schreiber, *A systolic architecture for singular value decomposition,* Proc. 1st Internat. Coll. on Vector and Parallel Comput. in Sci. Applic., Paris, France, Mar. 1983, pp. 143-148.

[21] G.W. Stewart, *A Jacobi-like algorithm for computing the Schur decomposition of a non-Hermitian matrix,* Tech. Report 1321, Computer Science Dept., Univ. of Maryland, 1983.