SIAM J. SCI. STAT. COMPUT.
Vol. 6, No. 1, January 1985

THE SOLUTION OF SINGULAR-VALUE AND SYMMETRIC EIGENVALUE PROBLEMS ON MULTIPROCESSOR ARRAYS*

RICHARD P. BRENT† AND FRANKLIN T. LUK‡

Abstract. Parallel Jacobi-like algorithms are presented for computing a singular-value decomposition of an $m \times n$ matrix ($m \geqq n$) and an eigenvalue decomposition of an $n \times n$ symmetric matrix. A linear array of $O(n)$ processors is proposed for the singular-value problem; the associated algorithm requires time $O(mnS)$, where S is the number of sweeps (typically $S \leqq 10$). A square array of $O(n^2)$ processors with nearest-neighbor communication is proposed for the eigenvalue problem; the associated algorithm requires time $O(nS)$.

Key words. multiprocessor arrays, systolic arrays, singular-value decomposition, eigenvalue decomposition, real symmetric matrices, Hestenes method, Jacobi method, VLSI, real-time computation, parallel algorithms

1. Introduction. A singular-value decomposition (SVD) of a real $m \times n$ ($m \geqq n$) matrix A is its factorization into the product of three matrices:

$$A = U\Sigma V^T, \tag{1.1}$$

where U is an $m \times n$ matrix with orthonormal columns, Σ is an $n \times n$ nonnegative diagonal matrix and the $n \times n$ matrix V is orthogonal. This decomposition has many important scientific and engineering applications (cf. [1], [11], [26], [27]). If the matrix A is square (i.e., $m = n$) and symmetric, we may adjust the sign of the elements of Σ so that $U = V$. We then obtain an eigenvalue decomposition:

$$A = UDU^T, \tag{1.2}$$

wnere U is orthogonal and D diagonal. The advent of massively parallel computer architectures has aroused much interest in parallel singular-value and eigenvalue procedures, e.g. [2], [4], [5], [6], [7], [9], [13], [14], [16], [19], [20], [22], [23], [24], [25]. Such architectures may turn out to be indispensable in settings where real-time computation of the decompositions is required [26], [27]. Speiser and Whitehouse [26] survey parallel processing architectures and conclude that systolic arrays offer the best combination of characteristics for utilizing VLSI/VHSIC technology to do real-time signal processing. (See also [17], [27].)

In this paper we present an array of $O(n)$ linearly-connected processors which computes an SVD in time $O(mnS)$. Here S is a slowly growing function of n which is conjectured to be $O(\log n)$; for practical purposes S may be regarded as a constant (see [21] and the Appendix). Our array implements a one-sided orthogonalization method due to Hestenes [15]. His method is essentially the serial Jacobi procedure for finding an eigenvalue decomposition of the matrix A^TA, and has been used by Luk [20] on the ILLIAC IV computer. We also describe how one may implement a Jacobi method on a two-dimensional array of processors to compute an eigenvalue

* Received by the editors November 12, 1982, and in revised form August 9, 1983.

† Centre for Mathematical Analysis, Australian National University, GPO Box 4, Canberra, ACT 2601, Australia.

‡ Department of Computer Science, Cornell University, Ithaca, New York 14853. The research of this author was supported in part by the U.S. Army Research Office under grant DAAG 29-79-C0124 and the National Science Foundation under grant MCS-8213718, and in part by the Mathematical Sciences Research Centre and the Centre for Mathematical Analysis, Australian National University.

decomposition of a symmetric matrix. Our array requires $O(n^2)$ processors and $O(nS)$ units of time. Assuming that $S = O(\log n)$, this time requirement is within a factor $O(\log n)$ of that necessary for the solution of n linear equations in n unknowns on a systolic array [2], [3], [17], [18].

Results similar to ours have been reported in the literature. For computing the SVD, Sameh [23] describes an implementation of Hestenes' method on a ring of $O(n)$ processors. Suppose n is even (the result is similar for an odd n). At each orthogonalization step $n/2$ column rotations are performed. Sameh's permutation scheme requires $3n-2$ steps to ensure the execution of every possible pairwise rotation at least once; our permutation scheme (presented in § 3) requires only $n-1$ steps.

Parallel Jacobi methods for computing eigenvalues are given in [7], [16], [22]. However, the procedure of Sameh [22] may be unsuitable for multiprocessor arrays. For simplicity, assume again that n is even, so $n/2$ off-diagonal elements can be set to zero at each elimination step. Let us compare the number of permutations necessary for the annihilation of each off-diagonal element at least once. Our procedure (see §§ 3 and 6) requires $n-1$ permutations, which is optimal; that of Chen and Irani [7] requires n permutations. The scheme of Kuck and Sameh [16] is worse. Their basic scheme appears to cycle every $2n-2$ steps and to miss some off-diagonal elements. A modification ("the second row and column are shifted to the nth position after every $(n-1)$ orthogonal transformations") can be made to overcome this problem, but the modified scheme requires $(n-1)^2$ permutations [7].

Let us generalize the notion of a "sweep" and use it to denote a minimum-length sequence of rotations that eliminates each off-diagonal element at least once [7]. It is probably fair to assume that the Jacobi procedures in [7], [16] and in this paper require an equal number (say S) of sweeps for convergence. For the algorithms presented in this paper a sweep always consists of $n(n-1)/2$ rotations (the minimal number possible), but this is not the case for the Chen and Irani or Kuck and Sameh algorithms mentioned above. The architecture proposed in [7] is a linear array of $O(n)$ processors; the associated Jacobi method requires time $O(n^2S)$. The architecture described in [16] is a square array of $O(n)$ processors, with boundary wraparounds and a broadcast unit. The associated algorithm requires time $O(n^3S)$. In comparison, our procedure requires $O(n^2)$ processors and $O(nS)$ units of time.

The principal results of this paper were first reported in [4], [5]. A related (generalized) SVD algorithm is presented by the authors and Van Loan in [6]. It requires $O(n^2)$ processors and $O(nS)$ time to compute the (generalized) singular values of $n \times n$ matrices.

This paper is organized as follows. Sections 2–4 are devoted to the singular-value problem and §§ 5–8 to the eigenvalue problem. Hestenes' method is reviewed in § 2. The new ordering is described in § 3 and the corresponding SVD algorithm in § 4. The serial Jacobi method is outlined in § 5. Details are filled in and some variations and extensions of the basic algorithm are given in §§ 7 and 8. The results of some numerical simulations are presented in the Appendix.

The SVD algorithm described in §§ 3–4 below is being implemented on an experimental 64-processor systolic array by Speiser at the Naval Ocean Systems Center (San Diego).

2. Hestenes' method. We wish to compute an SVD of an $m \times n$ matrix A, where $m \geqq n$. An idea is to generate an orthogonal matrix V such that the transformed matrix $AV = W$ has orthogonal columns. Normalizing the Euclidean length of each nonnull column to unity, we get the relation

$$W = \tilde{U}\Sigma, \tag{2.1}$$

where $\tilde{U}$ is a matrix whose nonnull columns form an orthonormal set of vectors and Σ is a nonnegative diagonal matrix. An SVD of A is given by

$$A = \tilde{U}\Sigma V^T. \tag{1.1'}$$

As a null column of $\tilde{U}$ is always associated with a zero diagonal element of Σ, there is no essential difference between (1.1) and (1.1′).

Hestenes [15] uses plane rotations to construct V. He generates a sequence of matrices $\{A_k\}$ using the relation

$$A_{k+1} = A_k Q_k,$$

where $A_1 = A$ and Q_k is a plane rotation. Let $A_k \equiv (\mathbf{a}_1^{(k)}, \cdots, \mathbf{a}_n^{(k)})$ and $Q_k \equiv (q_{rs}^{(k)})$, and suppose Q_k represents a rotation in the (i, j) plane, with $i < j$, i.e.

$$\begin{aligned} q_{ii}^{(k)} &= \cos\theta, & q_{ij}^{(k)} &= \sin\theta, \\ q_{ji}^{(k)} &= -\sin\theta, & q_{jj}^{(k)} &= \cos\theta. \end{aligned} \tag{2.2}$$

We note that postmultiplication by Q_k affects only two columns:

$$(\mathbf{a}_i^{(k+1)}, \mathbf{a}_j^{(k+1)}) = (\mathbf{a}_i^{(k)}, \mathbf{a}_j^{(k)})\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}. \tag{2.3}$$

The rotation angle θ is chosen so that the two new columns are orthogonal. Adopting the formulas of Rutishauser [21], we let

$$\alpha \equiv \|\mathbf{a}_i^{(k)}\|_2^2, \quad \beta \equiv \|\mathbf{a}_j^{(k)}\|_2^2, \quad \gamma \equiv \mathbf{a}_i^{(k)T}\mathbf{a}_j^{(k)}. \tag{2.4}$$

We set $\theta = 0$ if $\gamma = 0$; otherwise we compute

$$\xi = \frac{\beta - \alpha}{2\gamma}, \quad t = \frac{\operatorname{sign}(\xi)}{|\xi| + \sqrt{1+\xi^2}}, \quad \cos\theta = \frac{1}{\sqrt{1+t^2}}, \tag{2.5}$$

and

$$\sin\theta = t \cdot \cos\theta.$$

The rotation angle always satisfies

$$|\theta| \leq \frac{\pi}{4}. \tag{2.6}$$

However, there remains the problem of choosing (i, j), which is usually done according to some fixed cycle. An objective is to go through all column pairs exactly once in any sequence (a sweep) of $n(n-1)/2$ rotations. A simple sweep consists of a cyclic-by-rows ordering:

$$(1, 2), (1, 3), \cdots, (1, n), (2, 3), \cdots, (2, n), (3, 4), \cdots, (n-1, n). \tag{2.7}$$

Forsythe and Henrici [10] prove that, subject to (2.6), the cyclic-by-rows Jacobi method always converges. Convergence of the cyclic-by-rows Hestenes method thus follows.

Unfortunately, the cyclic-by-rows scheme is apparently not amenable to parallel processing. In § 3 we present an ordering that enables us to do $\lfloor n/2 \rfloor$ rotations simultaneously. The (theoretical) price we pay is the loss of guaranteed convergence. Hansen [12] discusses the convergence properties associated with various orderings for the serial Jacobi method. He defines a certain "preference factor" for comparing different ordering schemes. Our new ordering is in fact quite desirable, for it asymptotically optimizes the preference factor as $n \to \infty$. Thus, although the convergence proof of [10] does not apply, we expect convergence in practice to be faster than for the cyclic-by-rows ordering. Simulation results (presented in the Appendix) support this conclusion.

To enforce convergence, we may choose a threshold approach [29, pp. 277–278]. That is, we associate with each sweep a threshold value, and when making the transformations of that sweep, we omit any rotation based on a normalized inner product

$$\frac{\mathbf{a}_i^{(k)T}\mathbf{a}_j^{(k)}}{\|\mathbf{a}_i^{(k)}\|_2\|\mathbf{a}_j^{(k)}\|_2}$$

which is below the threshold value. Although such a strategy guarantees convergence, we do not know any example for which our new ordering fails to give convergence even without using thresholds. Our method, like the cyclic-by-rows method, is ultimately quadratically convergent [28].

The plane rotations are accumulated if the matrix V is desired. We compute

$$V_{k+1} = V_k Q_k,$$

with $V_1 = I$. Denoting the rth column of V_k (respectively V_{k+1}) by $\mathbf{v}_r^{(k)}$ (respectively $\mathbf{v}_r^{(k+1)}$), we may update both A_k and V_k simultaneously:

$$\begin{pmatrix} \mathbf{a}_i^{(k+1)} & \mathbf{a}_j^{(k+1)} \\ \mathbf{v}_i^{(k+1)} & \mathbf{v}_j^{(k+1)} \end{pmatrix} = \begin{pmatrix} \mathbf{a}_i^{(k)} & \mathbf{a}_j^{(k)} \\ \mathbf{v}_i^{(k)} & \mathbf{v}_j^{(k)} \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}. \tag{2.8}$$

3. Generation of all pairs (i, j). In this section we show how $O(n)$ linearly-connected processors can generate all pairs (i, j), $1 \leqq i < j \leqq n$, in $O(n)$ steps. The application to the computation of the SVD and of the symmetric eigenvalue decomposition is described in § 4 and in §§ 6–8, respectively.

First, suppose n is even. We use $n/2$ processors $P_1, \cdots, P_{n/2}$, where P_k and P_{k+1} communicate ($k = 1, 2, \cdots, n/2-1$). Each processor P_k has registers L_k and R_k, output lines out L_k and out R_k, and input lines in L_k and in R_k, except that out L_1, in L_1, out $R_{n/2}$ and in $R_{n/2}$ are omitted. The output out R_k is connected to the input in L_{k+1} as shown in Fig. 1.

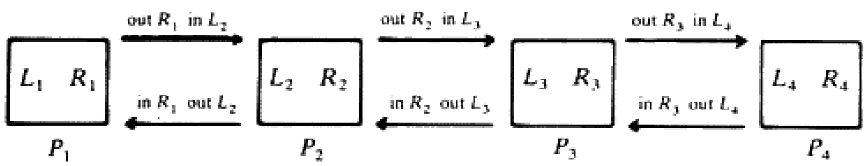

FIG. 1. *Inter-processor connections for $n = 8$.*

Initially $L_k = 2k-1$ and $R_k = 2k$. At each time step processor P_k executes the following program:

if $L_k < R_k$ then process (L_k, R_k) else process (R_k, L_k);
if $k = 1$ then out $R_k \leftarrow R_k$
  else if $k < n/2$ then out $R_k \leftarrow L_k$;
if $k > 1$ then out $L_k \leftarrow R_k$;
{wait for outputs to propagate to inputs of adjacent processors}
if $k < n/2$ then $R_k \leftarrow$ in R_k else $R_k \leftarrow L_k$;
if $k > 1$ then $L_k \leftarrow$ in L_k;

Here "process (i, j)" means perform whatever operations are required on the pair (i, j), $1 \leqq i < j \leqq n$. The operation of the systolic array is illustrated in Fig. 2.

We see that the index 1 stays in the register L_1 of processor P_1. Indices $2, \cdots, n$ travel through a cycle of length $n-1$ consisting of the registers $L_2, L_3, \cdots, L_{n/2}, R_{n/2}, R_{n/2-1}, \cdots, R_1$. During any $n-1$ consecutive steps a pair (i, j) or (j, i) can appear in a register pair (L_k, R_k) at most once. A parity argument shows that (i, j) and (j, i) cannot both occur (see Fig. 2). Since there are $n/2$ register pairs at each of $n-1$ time steps, each possible pair (i, j), $1 \leqq i < j \leqq n$, is processed exactly once during a cycle of $n-1$ consecutive steps.

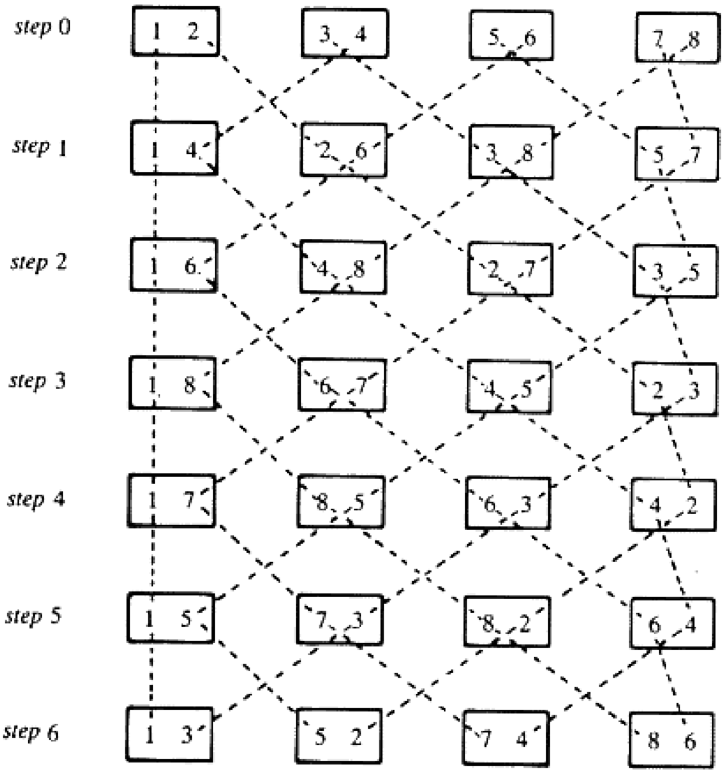

FIG. 2. *Full cycle of the systolic array for $n = 8$.*

If n is odd, we use $\lceil n/2 \rceil$ processors but initialize $L_k = 2k-2$, $R_k = 2k-1$ for $k = 1, \cdots, \lceil n/2 \rceil$ and omit any "process" calls from processor P_1.

It is interesting to note that similar permutations are "well known" for use in chess and bridge tournaments, but have apparently not been applied to parallel computation.

4. A one-dimensional systolic array for SVD computation. Assume that n is even (else we can add a zero column to A or modify the algorithm as described at the end of § 3). We use $n/2$ processors $P_1, \cdots, P_{n/2}$, as described in § 3, except that L_k and R_k are now local memories large enough to store a column of A (i.e., L_k and R_k each has at least m floating-point words). Shift registers or other sequential access memories are sufficient as we do not need random access to the elements of each column.

Suppose processor P_k contains column $\mathbf{a}_i^c$ in L_k and column $\mathbf{a}_j^c$ in R_k. It is clear that P_k can implement the column orthogonalization scheme in time $O(m)$ by making one pass through $\mathbf{a}_i^c$ and $\mathbf{a}_j^c$ to compute the inner products (2.4), and another pass to perform the transformations (2.3) or (2.8). Adjacent processors can then exchange columns in the same way that the processors of § 3 exchange indices. This takes time $O(m)$ if the bandwidth between adjacent processors is one floating-point word. (Alternatively, exchanges can be combined with the transformations (2.3) or (2.8).)

Consequently, we see that $n/2$ processors can perform a full sweep of the Hestenes method in $n-1$ steps of time $O(m)$ each, i.e., in total time $O(mn)$. Initialization requires that the $(2k-1)$th and $2k$th columns of A be stored in the local memory of processor P_k for $k=1,\cdots,n/2$; clearly this can also be performed in time $O(mn)$.

The process is iterative. Suppose S sweeps are required to orthogonalize the columns to full machine accuracy. We then have a systolic array of $n/2$ processors which computes the SVD in time $O(mnS)$. By comparison, the serial Hestenes algorithm takes time $O(mn^2S)$. Our simulation results suggest that S is $O(\log n)$, although for practical purposes we can regard S as a constant in the range six to ten [21].

After an integral number of sweeps the columns of the matrix $W \equiv AV$ (see (2.1)) will be stored in the systolic array (two per processor). If V is required, it can be accumulated at the same time that W is accumulated, at the expense of increasing each processor's local memory (but the computation time remains $O(mnS)$); see (2.8).

5. Serial Jacobi method. We now consider the related problem of diagonalizing a given $n\times n$ symmetric $A=A_1$. The serial Jacobi method generates a sequence of symmetric matrices $\{A_k\}$ via the relation

$$A_{k+1}=Q_k^TA_kQ_k,$$

where Q_k is a plane rotation. Let $A_k\equiv(a_{rs}^{(k)})$ and suppose Q_k represents a rotation through angle θ in the (i,j) plane, with $i<j$ (see (2.2)). We choose the rotation angle to annihilate the (i,j) element of A_k. If $a_{ij}^{(k)}=0$, we do not rotate, i.e., $\theta=0$. Otherwise we use the formulas in [21] to compute $\sin\theta$ and $\cos\theta$:

$$(5.1)\qquad \begin{aligned} \xi&=\frac{a_{jj}^{(k)}-a_{ii}^{(k)}}{2a_{ij}^{(k)}}, &\qquad \cos\theta&=\frac{1}{\sqrt{1+t^2}},\\ t&=\frac{\text{sign}(\xi)}{|\xi|+\sqrt{1+\xi^2}}=\tan\theta, &\qquad \sin\theta&=t\cdot\cos\theta. \end{aligned}$$

Note that the rotation angle θ may be chosen to satisfy

$$|\theta|\leqq\frac{\pi}{4}.$$

The new matrix A_{k+1} differs from A_k only in rows and columns i and j. The modified values are defined by

$$(5.2)\qquad \begin{aligned} a_{ii}^{(k+1)}&=a_{ii}^{(k)}-t\cdot a_{ij}^{(k)},\\ a_{jj}^{(k+1)}&=a_{jj}^{(k)}+t\cdot a_{ij}^{(k)},\\ a_{ij}^{(k+1)}&=a_{ji}^{(k+1)}=0,\\ &\left.\begin{aligned} a_{iq}^{(k+1)}&=a_{qi}^{(k+1)}=\cos\theta\cdot a_{iq}^{(k)}-\sin\theta\cdot a_{jq}^{(k)}\\ a_{jq}^{(k+1)}&=a_{qj}^{(k+1)}=\sin\theta\cdot a_{iq}^{(k)}+\cos\theta\cdot a_{jq}^{(k)} \end{aligned}\right\}\quad (q\neq i,j). \end{aligned}$$

Again we choose (i,j) in accordance to the new ordering introduced in § 3. The comments that were made in § 2 concerning various aspects (convergence proof, convergence rate, threshold approach, etc.) of the Hestenes method apply equally well here to the Jacobi procedure.

6. An idealized systolic architecture. In this section we describe an idealized systolic architecture for implementing the Jacobi method to compute an eigenvalue decomposition of A. The architecture is idealized in that it assumes the ability to broadcast row and column rotation parameters in constant time. In § 8 we show how to avoid this assumption, after showing in § 7 how to take advantage of symmetry, compute eigenvectors, etc.

Assume that the order n is even and that we have a square array of $n/2$ by $n/2$ processors, each processor containing a 2×2 submatrix of $A\equiv(a_{ij})$. Initially processor P_{ij} contains

$$\begin{pmatrix} a_{2i-1,2j-1} & a_{2i-1,2j} \\ a_{2i,2j-1} & a_{2i,2j} \end{pmatrix} \quad \text{for } i,j=1,\cdots,n/2,$$

and P_{ij} is connected to its nearest neighbors $P_{i\pm1,j}$ and $P_{i,j\pm1}$ (see Fig. 3). In general P_{ij} contains four real numbers

$$\begin{pmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{pmatrix},$$

where $\alpha_{ij}=\alpha_{ji}$, $\delta_{ij}=\delta_{ji}$ and $\beta_{ij}=\gamma_{ji}$ by symmetry.

The diagonal processors P_{ii} $(i=1,\cdots,n/2)$ act differently from the off-diagonal processors P_{ij} $(i\neq j,\ 1\leqq i,j\leqq n/2)$. Each time step the diagonal processors P_{ii} compute rotations

$$\begin{pmatrix} c_i & s_i \\ -s_i & c_i \end{pmatrix}$$

to annihilate their off-diagonal elements β_{ii} and γ_{ii}, (actually $\beta_{ii}=\gamma_{ii}$), i.e., so that $c_i^2+s_i^2=1$ and

$$\begin{pmatrix} c_i & -s_i \\ s_i & c_i \end{pmatrix}\begin{pmatrix} \alpha_{ii} & \beta_{ii} \\ \gamma_{ii} & \delta_{ii} \end{pmatrix}\begin{pmatrix} c_i & s_i \\ -s_i & c_i \end{pmatrix}=\begin{pmatrix} \alpha'_{ii} & 0 \\ 0 & \delta'_{ii} \end{pmatrix}$$

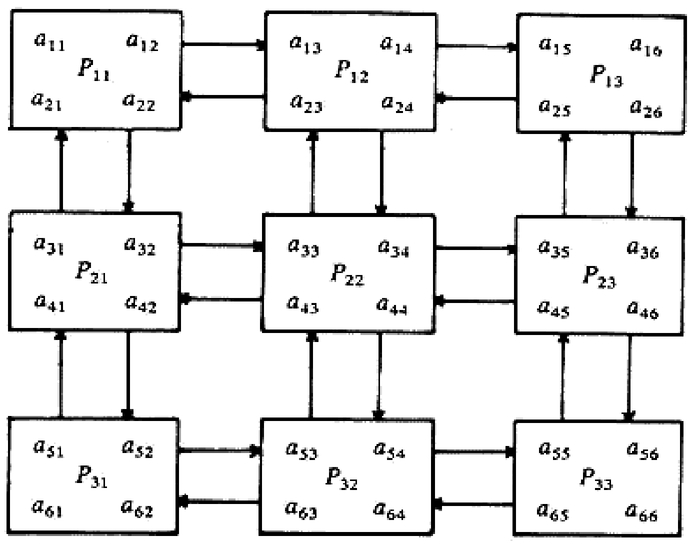

FIG. 3. *Initial configuration (idealized, $n=6$).*

is diagonal. From (5.1) and (5.2) with a change of notation we find that

$$\begin{pmatrix} c_i \\ s_i \end{pmatrix} = \frac{1}{\sqrt{1+t_i^2}} \begin{pmatrix} 1 \\ t_i \end{pmatrix} \tag{6.1}$$

and

$$\begin{pmatrix} \alpha'_{ii} \\ \delta'_{ii} \end{pmatrix} = \begin{pmatrix} \alpha_{ii} \\ \delta_{ii} \end{pmatrix} + t_i\beta_{ii} \begin{pmatrix} -1 \\ 1 \end{pmatrix},$$

where

$$t_i = \begin{cases} 0 & \text{if } \beta_{ii} = 0, \\ \dfrac{\text{sign}\,(\xi_i)}{|\xi_i| + \sqrt{1+\xi_i^2}} & \text{if } \beta_{ii} \neq 0, \end{cases} \tag{6.2}$$

and

$$\xi_i = \frac{\delta_{ii} - \alpha_{ii}}{2\beta_{ii}}.$$

To complete the rotations which annihilate β_{ii} and γ_{ii}, $i = 1, \cdots, n/2$, the off-diagonal processors P_{ij} $(i \neq j)$ must perform the transformations

$$\begin{pmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{pmatrix} \leftarrow \begin{pmatrix} \alpha'_{ij} & \beta'_{ij} \\ \gamma'_{ij} & \delta'_{ij} \end{pmatrix},$$

where

$$\begin{pmatrix} \alpha'_{ij} & \beta'_{ij} \\ \gamma'_{ij} & \delta'_{ij} \end{pmatrix} = \begin{pmatrix} c_i & -s_i \\ s_i & c_i \end{pmatrix} \begin{pmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{pmatrix} \begin{pmatrix} c_j & s_j \\ -s_j & c_j \end{pmatrix}.$$

We assume that the diagonal processor P_{ii} broadcasts the rotation parameters c_i and s_i to processors P_{ij} and P_{ji} $(j = 1, \cdots, n/2)$ in constant time, so that the off-diagonal processor P_{ij} has access to the parameters c_i, s_i, c_j and s_j when required. (This assumption is removed in § 8.)

To complete a step, columns (and corresponding rows) are interchanged between adjacent processors so that a new set of n off-diagonal elements is ready to be annihilated by the diagonal processors during the next time step. This is done in two sub-steps. First, adjacent columns are exchanged as in the SVD algorithm described in §§ 3–4 and as illustrated in Fig. 2. Next, the same permutation is applied to rows, so as to maintain symmetry. Formally, we can specify the operations performed by a processor P_{ij} with outputs out $h\alpha_{ij}, \cdots$, out $h\delta_{ij}$, out $v\alpha_{ij}, \cdots$, out $v\delta_{ij}$, and inputs in $h\alpha_{ij}, \cdots$, in $v\delta_{ij}$ by Program 1. Note that outputs of one processor are connected to inputs of adjacent processors in the obvious way, e.g. out $h\beta_{ij}$ is connected to in $h\alpha_{i,j+1}$ $(1 \leq i \leq n/2, 1 \leq j < n/2)$: see Fig. 4. In Fig. 4 and elsewhere, we have omitted subscripts (i, j) if no ambiguity arises, e.g. in $v\alpha$ is used instead of in $v\alpha_{ij}$.

```
{subscripts (i, j) omitted if no ambiguity results}
{column interchanges}
    if i = 1 then [out hβ ← β; out hδ ← δ]
      else if i < n/2 then [out hβ ← α; out hδ ← γ];
    if i > 1 then [out hα ← β; out hγ ← δ];
    {wait for outputs to propagate to inputs of adjacent processors}
    if i < n/2 then [β ← in hβ; δ ← in hδ]
      else [β ← α; δ ← γ];
    if i > 1 then [α ← in hα; γ ← in hγ];
{row interchanges}
    if j = 1 then [out vγ ← γ; out vδ ← δ]
      else if j < n/2 then [out vγ ← α; out vδ ← β];
    if j > 1 then [out vα ← γ; out vβ ← δ];
    {wait for outputs to propagate to inputs of adjacent processors}
    if j < n/2 then [γ ← in vγ; δ ← in vδ]
      else [γ ← α; δ ← β];
    if j > 1 then [α ← in vα; β ← in vβ];
```

PROGRAM 1. *Column and row interchanges for idealized processor P_{ij}.*

The only difference between the data flow here and that in § 4 is that here rows are permuted as well as columns in order to maintain the symmetry of A and move the elements to be annihilated during the next time step into the diagonal processors. Hence, from § 3 it is clear that a complete sweep is performed every $n-1$ steps, because each off-diagonal element of A is moved into one of the diagonal processors in exactly one of the steps. Each sweep takes time $O(n)$ so, assuming that $O(\log n)$ sweeps are required for convergence, the total time required to diagonalize A is $O(n \log n)$.

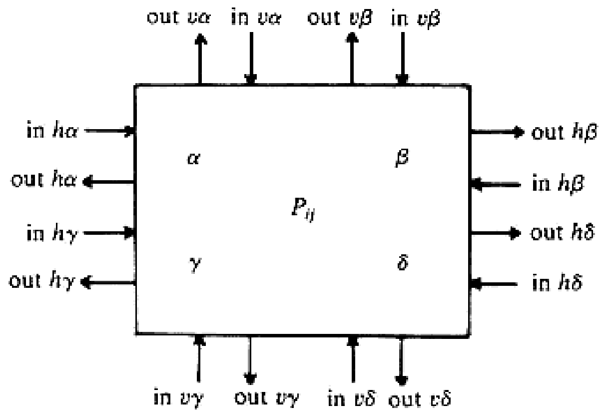

FIG. 4. *Input and output lines for idealized processor P_{ij} with nearest-neighbor connections.*

7. Further details. Several assumptions were made in § 6 to simplify the exposition. In this section we show how to remove these assumptions (except for the broadcast of rotation parameters, discussed in § 8) and we also suggest some practical optimizations.

7.1. Threshold strategy. It is clear that a diagonal processor P_{ii} might omit rotations if its off-diagonal elements $\beta_{ii} = \gamma_{ii}$ were sufficiently small. All that is required is to broadcast

$$\begin{pmatrix} c_i \\ s_i \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

along processor row and column i. As discussed in § 2, a suitable threshold strategy guarantees convergence, although we do not know any example for which our ordering fails to give convergence even without a threshold strategy.

7.2. Computation of eigenvectors. If eigenvectors are required, the matrix U of eigenvectors can be accumulated at the same time as A is being diagonalized. Each systolic processor P_{ij} $(1 \leqq i, j \leqq n/2)$ needs four additional memory cells

$$\begin{pmatrix} \mu_{ij} & \nu_{ij} \\ \sigma_{ij} & \tau_{ij} \end{pmatrix},$$

and during each step sets

$$\begin{pmatrix} \mu_{ij} & \nu_{ij} \\ \sigma_{ij} & \tau_{ij} \end{pmatrix} \leftarrow \begin{pmatrix} \mu_{ij} & \nu_{ij} \\ \sigma_{ij} & \tau_{ij} \end{pmatrix} \begin{pmatrix} c_j & s_j \\ -s_j & c_j \end{pmatrix}.$$

Each processor transmits its

$$\begin{pmatrix} \mu & \nu \\ \sigma & \tau \end{pmatrix}$$

values to adjacent processors in the same way as its

$$\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$$

values (see Program 1). Initially $\mu_{ij} = \nu_{ij} = \sigma_{ij} = \tau_{ij} = 0$ if $i \neq j$, and $\mu_{ii} = \tau_{ii} = 1$, $\sigma_{ii} = \nu_{ii} = 0$. After a sufficiently large (integral) number of sweeps, we have U defined to working accuracy by

$$\begin{pmatrix} u_{2i-1,2j-1} & u_{2i-1,2j} \\ u_{2i,2j-1} & u_{2i,2j} \end{pmatrix} = \begin{pmatrix} \mu_{ij} & \nu_{ij} \\ \sigma_{ij} & \tau_{ij} \end{pmatrix}.$$

7.3. Diagonal connections. In Program 1 we assumed that only horizontal and vertical nearest-neighbor connections were available. Except at the boundaries, diagonal connections are more convenient. This is illustrated in Figs. 5 and 6 (with subscripts (i, j) omitted).

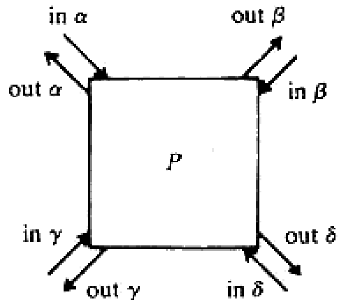

FIG. 5. *Diagonal input and output lines for processor P_{ij}.*

Diagonal outputs and inputs are connected in the obvious way, as shown in Fig. 6,

$$\text{e.g. out } \beta_{ij} \text{ is connected to} \begin{cases} \text{in } \gamma_{i-1,j+1} & \text{if } i > 1, j < n/2, \\ \text{in } \alpha_{i,j+1} & \text{if } i = 1, j < n/2, \\ \text{in } \delta_{i-1,j} & \text{if } i > 1, j = n/2, \\ \text{in } \beta_{i,j} & \text{if } i = 1, j = n/2. \end{cases}$$

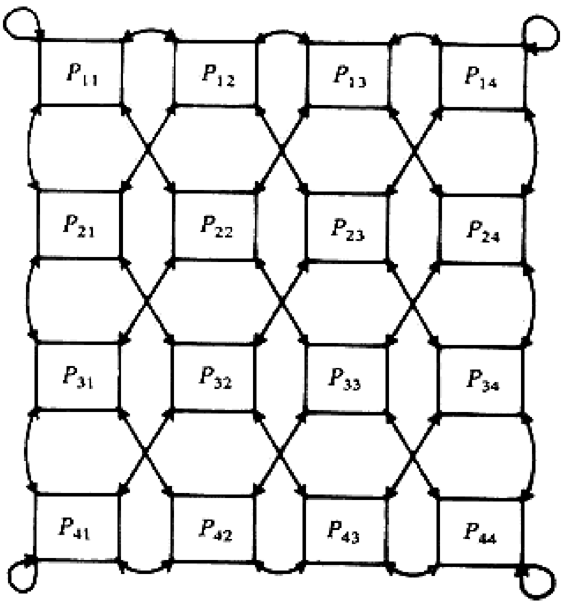

FIG. 6. *"Diagonal" connections, $n=8$ (here and below $\leftrightarrow$ stands for $\rightleftarrows$).*

Program 2 is equivalent to Program 1 but assumes a diagonal connection pattern as illustrated in Figs. 5 and 6. Subsequently we assume the diagonal connection pattern for convenience, although it can easily be simulated if only horizontal and vertical connections are available.

{subscripts (i, j) omitted for clarity}

if $(i=1)$ and $(j=1)$ then $\begin{bmatrix} \text{out } \alpha \leftarrow \alpha; \text{ out } \beta \leftarrow \beta; \\ \text{out } \gamma \leftarrow \gamma; \text{ out } \delta \leftarrow \delta; \end{bmatrix}$

else if $i=1$ then $\begin{bmatrix} \text{out } \alpha \leftarrow \beta; \text{ out } \beta \leftarrow \alpha; \\ \text{out } \gamma \leftarrow \delta; \text{ out } \delta \leftarrow \gamma; \end{bmatrix}$

else if $j=1$ then $\begin{bmatrix} \text{out } \alpha \leftarrow \gamma; \text{ out } \beta \leftarrow \delta; \\ \text{out } \gamma \leftarrow \alpha; \text{ out } \delta \leftarrow \beta; \end{bmatrix}$

else $\begin{bmatrix} \text{out } \alpha \leftarrow \delta; \text{ out } \beta \leftarrow \gamma; \\ \text{out } \gamma \leftarrow \beta; \text{ out } \delta \leftarrow \alpha; \end{bmatrix}$

{wait for outputs to propagate to inputs of adjacent processors};

$\alpha \leftarrow \text{in } \alpha; \ \beta \leftarrow \text{in } \beta;$

$\gamma \leftarrow \text{in } \gamma; \ \delta \leftarrow \text{in } \delta.$

PROGRAM 2. *Diagonal interchanges for processor P_{ij}.*

7.4. Taking full advantage of symmetry. Because A is symmetric and our transformations preserve symmetry, only a triangular array of $(1/2)(n/2)(n/2+1) = n(n+2)/8$ systolic processors is necessary for the eigenvalue computation. In the description above, simply replace any reference to a below-diagonal element a_{ij} (or processor P_{ij}) with $i>j$ by a reference to the corresponding above-diagonal element a_{ji} (or processor P_{ji}). Note, however, that this idea complicates the programs, and cannot be used if eigenvectors as well as eigenvalues are to be computed. Hence, for clarity of exposition we do not take advantage of symmetry in what follows, although only straightforward modifications would be required to do so.

7.5. Odd n. So far we assumed n to be even. For odd n we can modify the program for processors P_{1i} and P_{i1} $(i = 1, \cdots, \lceil n/2 \rceil)$ in a manner analogous to that used in § 3, or simply border A by a zero row and column. For simplicity we continue to assume that n is even.

7.6. Rotation parameters. In § 6 we assumed that the diagonal processor P_{ii} would compute c_i and s_i according to (6.1), and then broadcast both c_i and s_i along processor row and column i. It may be preferable to broadcast only t_i (given by (6.2)) and let each off-diagonal processor P_{ij} compute c_i, s_i, c_j and s_j from t_i and t_j. Thus communication costs are reduced at the expense of requiring off-diagonal processors to compute two square roots per time step (but this may not be significant since the diagonal processors must compute one or two square roots per step in any case). In what follows a "rotation parameter" may mean either t_i or the pair (c_i, s_i).

8. Avoiding broadcast of rotation parameters. The most serious assumption of § 6 is that rotation parameters computed by diagonal processors can be broadcast along rows and columns in constant time. We now show how to avoid this assumption, and merely transmit rotation parameters at constant speed between adjacent processors, while retaining total time $O(n)$ for the algorithm. We use a special case of a general procedure (due to Leiserson and Saxe) for the elimination of broadcasting.

Let $\Delta_{ij} = |i-j|$ denote the distance of processor P_{ij} from the diagonal. The operation of processor P_{ij} will be delayed by Δ_{ij} time units relative to the operation of the diagonal processors, in order to allow time for rotation parameters to be propagated at unit speed along each row and column of the processor array.

A processor cannot commence a rotation until data from earlier rotations is available on all its diagonal input lines. Thus, processor P_{ij} needs data from processors $P_{i-1,j-1}$, $P_{i-1,j+1}$, $P_{i+1,j-1}$ and $P_{i+1,j+1}$ if $1 < i < n/2$, $1 < j < n/2$ (for the other cases see § 7.3). Since

$$|\Delta_{ij} - \Delta_{i\pm1,j\pm1}| \leqq 2$$

it is sufficient for processor P_{ij} to be idle for two time steps while waiting for the processors $P_{i\pm1,j\pm1}$ to complete their (possibly delayed) steps. Thus, the price paid to avoid broadcasting rotation parameters is that each processor is active for only one third of the total computation time. A similar inefficiency occurs with many other systolic algorithms, [2], [3], [17], [18]. (The fraction one-third can be increased almost to unity if rotation parameters are propagated at greater than unit speed.)

A typical processor P_{ij} $(1 < j \leqq i < n/2)$ has input and output lines as shown in Fig. 7 (with subscripts (i, j) or (i, i) omitted). Figure 7 differs from Fig. 5 in that it

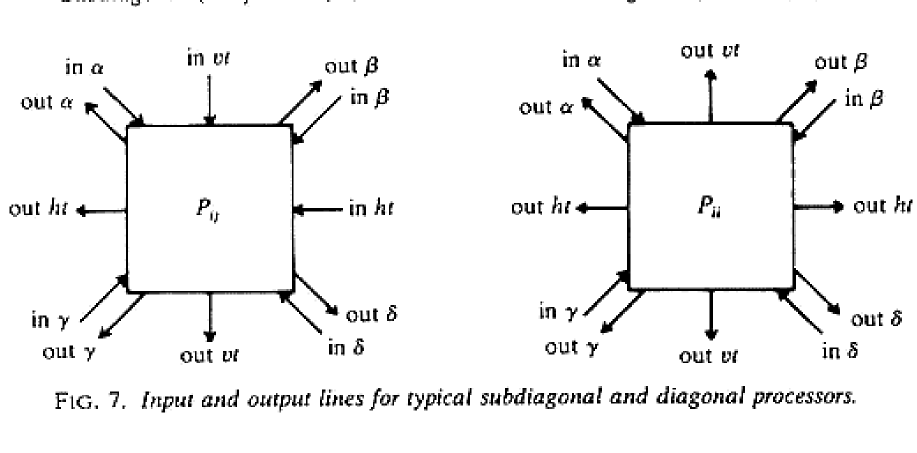

FIG. 7. *Input and output lines for typical subdiagonal and diagonal processors.*

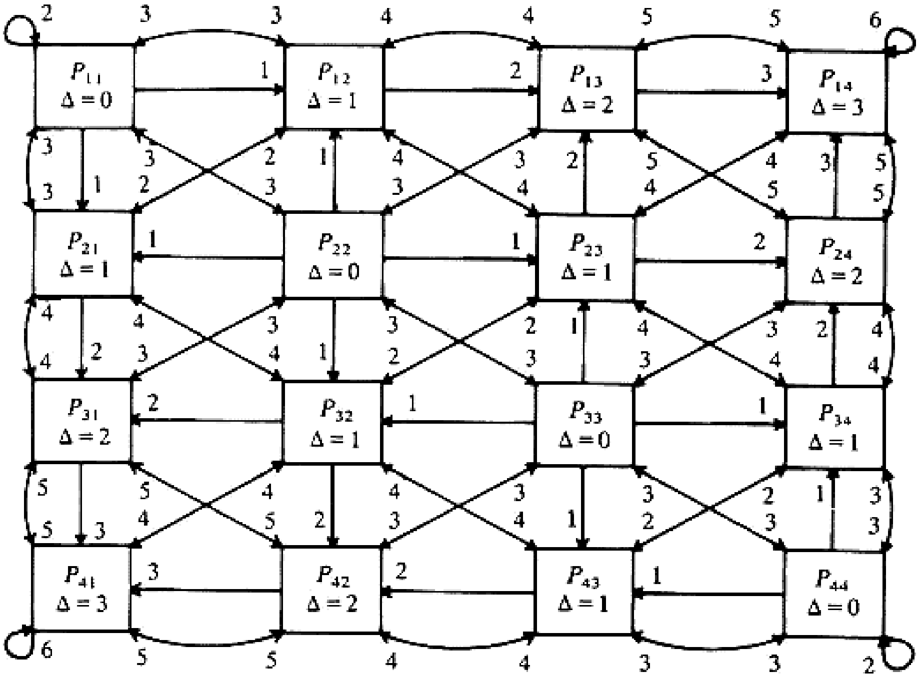

FIG. 8. *Interprocessor connections* ($n=8$). (*The first times at which inputs are available are indicated.*)

shows the horizontal and vertical lines in *ht*, out *ht*, in *vt*, out *vt* for transmission of rotation parameters. Processors interconnect as shown in Fig. 8.

Assuming that the array $(a_{ij})_{1\leqq i,j\leqq n}$ is available in the systolic array at time $T=0$, the operation of processor P_{ij} proceeds as described by Program 3. We assume that each time step has nonoverlapping read and write phases; the result of a write at step T should be available at the read phase of steps $T+1$, $T+2$ and $T+3$ in a neighbouring processor, but should not interfere with a read at step T in a neighbouring processor. The first time steps at which data are available on various processors' input lines are indicated in Fig. 8.

Program 3 does not compute eigenvectors, but may easily be modified to do so (as outlined in § 7). We have also omitted a termination criterion. The simplest is to perform a fixed number S (say conservatively 10) sweeps; then processor P_{ij} halts when $T=3S(n-1)+\Delta_{ij}+3$, since a sweep takes $3(n-1)$ time steps. A more sophisticated criterion is to stop if no nontrivial rotations were performed during the previous sweep. This requires communication along the diagonal, which can be done in $n/2$ time steps.

if $(T\geqq\Delta)$ and $(T-\Delta\equiv 0 \pmod 3)$ then
    begin

    if $T\neq\Delta$ then $\begin{bmatrix}\alpha & \beta\\ \gamma & \delta\end{bmatrix}\leftarrow\begin{bmatrix}\text{in }\alpha & \text{in }\beta\\ \text{in }\gamma & \text{in }\delta\end{bmatrix}$;

    if $\Delta=0$ then {diagonal processor}
      begin
      if $\beta=0$ then $\xi\leftarrow 0$ else $\xi\leftarrow(\delta-\alpha)/(2*\beta)$;

      if $\xi=0$ then $t\leftarrow 0$ else $t\leftarrow\dfrac{\text{sign}(\xi)}{|\xi|+\sqrt{1+\xi^2}}$;

      $t'\leftarrow t$;

$\alpha \leftarrow \alpha - t * \beta$; $\delta \leftarrow \delta + t * \beta$;

$\beta \leftarrow 0$; $\gamma \leftarrow 0$

end

else {off-diagonal processor}

begin

$t \leftarrow$ in ht; $t' \leftarrow$ in vt;

$c \leftarrow 1/\sqrt{1+t^2}$; $c' \leftarrow 1/\sqrt{1+t'^2}$;

$s \leftarrow t * c$; $s' \leftarrow t' * c'$;

$$\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \leftarrow \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \begin{pmatrix} c' & s' \\ -s' & c' \end{pmatrix}$$

end;

out $ht \leftarrow t$; out $vt \leftarrow t'$;
if $i > j$ then set out β as in Program 2;
if $i < j$ then set out γ as in Program 2
end
else if $(T \geqq \Delta)$ and $(T - \Delta \equiv 1 \pmod 3)$ then
begin
if $(i = 1)$ or $(j = 1)$ then set out α as in Program 2;
if $(i = n/2)$ or $(j = n/2)$ then set out δ as in Program 2
end
else if $(T \geqq \Delta)$ and $(T - \Delta \equiv 2 \pmod 3)$ then
begin
if $(i > 1)$ and $(j > 1)$ then set out α as in Program 2;
if $i \leqq j$ then set out β as in Program 2;
if $i \geqq j$ then set out γ as in Program 2;
if $(i < n/2)$ and $(j < n/2)$ then set out δ as in Program 2
end
else {do nothing this time step}.

PROGRAM 3. *Program for one time step of processor P_{ij}.*

9. Conclusion. We have presented a linear array of $\lceil n/2 \rceil$ processors, each able to perform floating-point operations (including square roots) and having $O(m)$ local storage, for computing the SVD of a real $m \times n$ matrix in time $O(mn \log n)$. We have also described how a square array of $\lceil n/2 \rceil$ by $\lceil n/2 \rceil$ processors, each with similar arithmetical capabilities but with only $O(1)$ local storage, and having connections to nearest horizontal and vertical (and preferably also diagonal) neighbors, can compute the eigenvalues and eigenvectors of a real symmetric matrix in time $O(n \log n)$. The constant is sufficiently small that the method is competitive with the usual $O(n^3)$ serial algorithms, e.g., tridiagonalization followed by the QR iteration, for quite small n. The speedup should be significant for real-time computations with moderate or large n.

The problem of computing eigenvalues and eigenvectors of an unsymmetric real matrix on a systolic array is currently being investigated; unfortunately, the ideas used for symmetric matrices do not all appear to carry over to Eberlein's methods [8] in an obvious way. However, everything that we have said concerning real symmetric matrices goes over with the obvious changes to complex Hermitian matrices.

Appendix. Simulation results. We have compared the ordering described in § 3 with the cyclic-by-rows ordering (2.7) by applying the Jacobi method with each ordering to random $n \times n$ symmetric matrices (a_{ij}), where the elements a_{ij} for $1 \leqq i \leqq j \leqq n$ were uniformly and independently distributed in $[-1, 1]$. (Other distributions were also tried, and similar results were obtained.) The stopping criterion was that the sum $\sum_{i \neq j} a_{ij}^2$ of squares of off-diagonal elements was reduced to 10^{-12} times its initial value. Table 1 gives the mean number of sweeps S_{row} and S_{new} for the cyclic-by-rows ordering and the ordering of § 3, respectively, where a "sweep" is $n(n-1)/2$ rotations. The maximum number of sweeps required for each ordering is given in parentheses in the Table.

TABLE 1
Simulation results for row and new orderings.

n	trials	S_{row}	S_{new}
4	5,000	2.96 (4.17)	2.64 (4.00)
6	5,000	3.63 (4.87)	3.37 (4.40)
8	2,000	4.07 (5.04)	3.79 (4.75)
10	2,000	4.39 (5.56)	4.09 (5.47)
20	1,000	5.23 (5.93)	4.94 (5.81)
30	1,000	5.67 (6.62)	5.41 (6.49)
40	1,000	5.92 (6.76)	5.74 (6.54)
50	1,000	6.17 (7.13)	5.99 (6.78)
100	500	6.81 (7.42)	6.78 (7.32)

From Table 1 we see that our new ordering is better than the cyclic-by-rows ordering, perhaps for the reason suggested in § 2, although the difference between the two orderings becomes less marked as n increases. For both ordering, the number of sweeps S grows slowly with n. Empirically we find that $S = O(\log n)$, and there are theoretical reasons for believing this, although it has not been proved rigorously. In practice S can be regarded as a constant (say 10) for all realistic values of n (say $n \leqq 1{,}000$); see [21]. More extensive simulation results for six different classes of orderings will be reported elsewhere.

Acknowledgment. We thank the referees and the editor for their comments, which helped to improve the presentation and make the list of references more complete.

REFERENCES

[1] H. C. ANDREWS AND C. L. PATTERSON, *Singular value decomposition and digital image processing*, IEEE Trans. Acoustics, Speech and Signal Processing ASSP-24 (1976), pp. 26–53.

[2] A. BOJANCZYK, R. P. BRENT AND H. T. KUNG, *Numerically stable solution of dense systems of linear equations using mesh-connected processors*, this Journal, 5 (1984), pp. 95–104. Also available as Tech. Report. TR-CS-81-01, Dept. Computer Science, Australian National Univ., 1981.

[3] R. P. BRENT AND F. T. LUK, *Computing the Cholesky factorization using a systolic architecture*, Proc. 6th Australian Computer Science Conference (1983), pp. 295–302.

[4] ———, *A systolic architecture for the singular value decomposition*, Tech. Report TR-CS-82-09, Dept. Computer Science, Australian National Univ., August, 1982.

[5] ———, *A systolic architecture for almost linear-time solution of the symmetric eigenvalue problem*, Tech. Report TR-CS-82-10, Dept. Computer Science, Australian National Univ., 1982.

[6] R. P. BRENT, F. T. LUK AND C. VAN LOAN, *Computation of the generalized singular value decomposition using mesh-connected processors*, Proc. SPIE Vol. 431, Real Time Signal Processing VI, SPIE, Bellingham, WA, 1983, pp. 66–71.

[7] K-W. CHEN AND K. B. IRANI, *A Jacobi algorithm and its implementation on parallel computers*, Proc. 18th Annual Allerton Conference on Communication, Control and Computing, 1980, pp. 564–573.

[8] P. J. EBERLEIN AND J. BOOTHROYD, *Solution to the eigenproblem by a norm reducing Jacobi type method*, in [30], pp. 327–338.

[9] A. M. Finn, F. T. Luk and C. Pottle, *Systolic array computation of the singular value decomposition*, Proc. SPIE Symp. East 1982, Vol. 341, Real Time Signal Processing V, 1982, pp. 35–43.
[10] G. E. Forsythe and P. Henrici, *The cyclic Jacobi method for computing the principal values of a complex matric*, Trans. Amer. Math. Soc., 94 (1960), pp. 1–23.
[11] G. H. Golub and F. T. Luk, *Singular value decomposition: applications and computations*, ARO Report 77-1, Trans. 22nd Conference of Army Mathematicians (1977), pp. 577–605.
[12] E. R. Hansen, *On cyclic Jacobi methods*, J. Soc. Indust. Appl. Math., 11, 1963, pp. 448–459.
[13] D. E. Heller and I. C. F. Ipsen, *Systolic networks for orthogonal equivalence transformations and their applications*, Proc. 1982 Conf. on Advanced Research on VLSI, Massachusetts Institute of Technology, 1982, pp. 113–122.
[14] ———, *Systolic networks for orthogonal decompositions*, this Journal, 4 (1983), pp. 261–269.
[15] M. R. Hestenes, *Inversion of matrices by biorthogonalization and related results*, J. Soc. Indust. Appl. Math., 6 (1958), pp. 51–90.
[16] D. J. Kuck and A. H. Sameh, *Parallel computation of eigenvalues of real matrices*, Information Processing 1971, North-Holland, Amsterdam, 1972, pp. 1266–1272.
[17] H. T. Kung, *Why systolic architectures*, IEEE J. Comput., (1982), pp. 37–46.
[18] H. T. Kung and C. E. Leiserson, *Algorithms for* VLSI *processor arrays*, in Introduction to VLSI Systems, C. Mead and L. Conway, eds. Addison-Wesley, Reading, MA, 1980, pp. 271–292.
[19] S. Y. Kung and R. J. Gal-Ezer, *Linear or square array fo. eigenvalue and singular value decompositions?*, Proc. USC Workshop on VLSI and Modern Signal Processing, Los Angeles, California (Nov. 1982), pp. 89–98.
[20] F. T. Luk, *Computing the singular-value decomposition on the* ILLIAC IV, ACM Trans. Math. Software, 6 (1980), pp. 524–539.
[21] H. Rutishauser, *The Jacobi method for real symmetric matrices*, in [30], pp. 202–211.
[22] A. H. Sameh, *On Jacobi and Jacobi-like algorithms for a parallel computer*, Math. Comput., 25 (1971), pp. 579–590.
[23] ———, *Solving the linear least squares problem on a linear array of processors*, Proc. Purdue Workshop on Algorithmically-specialized Computer Organizations, 1982.
[24] R. Schreiber, *Systolic arrays for eigenvalue computation*, Proc. SPIE Symp. East 1982, Vol. 341, Real-Time Signal Processing V, 1982.
[25] ———, *A systolic architecture for singular value decomposition*, Proc. 1st International Colloquium on Vector and Parallel Computing in Scientific Applications, Paris, Mar., 1983.
[26] J. M. Speiser and H. J. Whitehouse, *Architecture for real-time matrix operations*, Proc. 1980 Government Microcircuits Applications Conference, Houston, TX, Nov., 1980.
[27] H. J. Whitehouse, J. M. Speiser and K. Bromley, *Signal processing applications of systolic array technology*, Proc. USC Workshop on VLSI and Modern Signal Processing, Los Angeles, CA, Nov. 1982, pp. 5–10.
[28] J. H. Wilkinson, *Note on the quadratic convergence of the cyclic Jacobi process*, Numer. Math., 4 (1962), pp. 296–300.
[29] ———, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.
[30] J. H. Wilkinson and C. Reinsch, eds., *Handbook for Automatic Computation, Vol.* 2 (*Linear Algebra*), Springer-Verlag, Berlin, 1971.