Multiphase Systolic Algorithms for Spectral Decomposition

KuoJuey R. Liu, *Member, IEEE,* and Kung Yao, *Senior Member, IEEE*

***Abstract*—In this paper, we propose two multiphase systolic algorithms to solve the spectral decomposition problem based on the *QR* algorithm. The spectral decomposition is one of the most computationally intensive modern signal processing operations. While the *QR* algorithm is well known to be an effective method to solve the eigenvalue problem, there is still no single systolic array architecture that can compute the unitary *Q* matrix readily and perform the *QR* algorithm efficiently. Previous methods using the *QR* algorithm had communication problems among different architectures. In this paper, two arrays, a triangular and a rectangular, are presented to implement the multiphase algorithms. Details on these multiphase operations of the *QR* algorithm as well as architectural consequences and performance evaluation are discussed in the paper. Efficient fault-tolerant schemes for these multiphase operations are also considered.**

I. Introduction

COMPUTING the spectral decomposition of a matrix is an important issue in many modern signal processing and system applications. The feasibility of real-time processing for sophisticated modern signal processing systems depends crucially on efficient implementation of parallel processing of the algorithms and associated architectures needed to perform these operations [4], [21]. While many variations exist in the literature for solving these matrix problems, all these iterative methods are based either on the Jacobi-like method or the *QR* algorithm [10], [40], [43]. While there are some fundamental differences between these two approaches, both algorithms have good numerical stability and convergence rate properties and thus are desirable for possible implementation. Since present VLSI technology is capable of building a multiprocessor system on a chip, many researchers have proposed different parallel processing architectures to solve eigenvalue and singular value decomposition (SVD) problems.

For any complex-valued $m \times n$ matrix A, the classical spectral decomposition [41] of the $n \times n$ Hermetian matrix A^HA, is given by

$$A^HA = \sum_{i=1}^{n} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^H = V\Lambda V^H \tag{1}$$

Manuscript received March 19, 1990; revised October 8, 1990. This work was supported in part by the Minta Martin Award, College of Engineering, University of Maryland, a University of California MICRO grant, and by the NSF under Grant NCR-8814407.

K. J. R. Liu is with the Department of Electrical Engineering, Systems Research Center, University of Maryland, College Park, MD 20742.

K. Yao is with the Department of Electrical Engineering, University of California, Los Angeles, CA 90024-1594.

IEEE Log Number 9104008.

where $V = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n]$ is an $n \times n$ unitary matrix, $\Lambda = \text{diag}\,[\lambda_1, \cdots, \lambda_n]$, and H is the complex conjugate transpose operator. The $\lambda_i's$ are the eigenvalues satisfying $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_n \geq 0$ and the $v_i's$ are the eigenvectors satisfying $A^HA\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$. The decomposition of A^HA follows from the SVD [10] of A given by

$$A = USV^H \tag{2}$$

where $U = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n]$ is an $m \times n$ matrix with orthogonal column vectors, $S = \text{diag}\,[s_1, \cdots, s_n]$, and V is an $n \times n$ unitary matrix. The $s_i's$ are the singular values satisfying $s_1 \geq s_2 \geq \cdots \geq s_n \geq 0$ and are the positive square roots of $\lambda_i's$ such that $\Lambda = S^2$. In this paper, we shall use spectral decomposition in the broad sense of not only including the decompositions of (1) and (2), but also including the eigenvalue decomposition of an arbitrary complex-valued $n \times n$ matrix A given by

$$AX = \Lambda X \tag{3}$$

where X is an $n \times n$ matrix of eigenvectors and $\Lambda = \text{diag}\,[\lambda_1, \cdots, \lambda_n]$ is the matrix of eigenvalues of A.

Luk [24], Brent and Luk [3], and Gao and Thomas [7] have effectively used the Jacobi-like method to solve these problems for either a multiprocessor system or systolic array. The basic problem concerns the diagonalization of a 2×2 matrix by the rotation matrices $J(\theta)$ and $K(\phi)$ in

$$J(\theta)^T \begin{bmatrix} w & x \\ y & z \end{bmatrix} K(\phi) = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix} \tag{4}$$

where w, x, y, and z are elements in two corresponding rows and columns of A. A two-stage procedure is then used to find θ and ϕ [24]. To find the SVD of a square matrix A, an appropriate sequence of 2×2 matrices is computed by using the basic Jacobi transformation in

$$T_{ij}\colon\ A \leftarrow J_{ij}^T A K_{ij} \tag{5}$$

where J_{ij} and K_{ij} are rotations in the (i, j) plane chosen to annihilate the (i, j) and (j, i) elements of A, respectively [24]. While the Jacobi-like method, as considered in [24], is currently known as one of the most effective parallel SVD algorithm for full dense matrices, the computations required to obtain the rotational matrices needed in this approach to obtain the singular vectors are not simple (either through broadcast in the array or by slowing down the operations) [24]. Moreno and Lang [29] also considered some alternatives to the algorithms in [3].

1053-587X/92$03.00 © 1992 IEEE

On the other hand, other researchers have used the *QR* algorithm to solve the eigenvalue problems. Heller and Ipsen [13], [17] performed the *QR* iteration for banded matrix based on an orthogonal systolic network, and Schreiber [35] combined their network with Gentleman and Kung's *QR* array to compute the *QR* algorithm. These methods required the computation of the unitary matrix Q. However, problems exist in the concurrent computation of Q and the pipeline operation of the *QR* iteration [17]. In [28], Moldovan *et al.* studied the mapping of a large *QR* algorithm onto a fixed size array. Torralba and Navarro [42] further purposed a size-independent linear array for *QR* iteration and Hessenberg reduction. While this approach can provide an efficient computation of one iteration of the *QR* iteration, it is not obvious how to pipeline the iteration.

For some system applications, such as matrix rank determination and system identification [20], the efficient computation of singular values is sufficient, while in other applications such as antenna beamformation [27], [38], spectral estimation [19], [34], direct finding [11], [30], etc., the eigenvectors are crucially needed. This makes the practical implementation of the systolic arrays discussed above difficult for many applications since they either cannot compute the eigenvector or cannot obtain the eigenvector without broadcast. For example, for the MUSIC algorithm [12], once we determine the signal subspace and noise subspace from the eigenvectors, the sample spectrum is then determined by

$$S(\omega) = \frac{1}{s^H(\omega) X_N X_N^H s(\omega)}$$

where X_N is the matrix of eigenvectors which generate the noise subspace and

$$s(\omega) = [1, e^{-j\omega}, \cdots, e^{-j\omega(K-1)}]$$

with K being the dimension of the matrix X_N. A system which consists of several systolic modules to compute the MUSIC algorithm has been proposed in [33]. However, communication problems among the modules and the difficulty of matching the pipeline rates and timings among different modules may pose difficulties for practical implementation.

Presently, there is no known simple efficient systolic array approach for the generation of eigenvectors. The main reason is that there is no single architecture that is capable of handling all the steps required in the algorithm such that we can pipeline the successive iteration readily. The communication cost among different architectures is high and the interface problem for an efficient data flow is demanding. In this paper, we propose two multiphase systolic algorithms to solve the spectral decomposition problem based on the *QR* algorithm. By multiphase operations we mean that the processing cells can perform different arithmetic operations in different phase of the computations. Two systolic arrays, one triangular and the other rectangular, are designed based on the multiphase concept. A key feature in our method for the successful application of the *QR* algorithm is that the Q matrix of the *QR* decomposition can be computed explicitly by multiphase operations. With the proper feedback of this Q matrix, the *QR* algorithm can be computed and pipelined effectively in a single systolic array. From the accumulation of those Q matrices in another array, eigenvectors can be computed without needing global communication *inside* the array.

In Section II, some preliminary matrix operations useful for the multiphase operations are discussed. In Section III, we review the *QR* algorithm and show the evaluation of the eigenvector from cumulative multiplications of the Q matrices. Then two multiphase systolic arrays for the *QR* algorithm and the Hessenberg reduction are presented in Section IV. Their performances, numerical stabilities, and convergence rates are studied in Section V. Finally, some efficient fault-tolerant schemes that can be incorporated with the arrays are discussed in Section VI. A brief conclusion is given in Section VII.

II. Systolic Array Matrix Processing

In this section, we consider some preliminary matrix and associated systolic array operations needed in the multiphase systolic algorithms for spectral decompositions.

A. QR Decomposition

A nondegenerate $m \times n$ rectangular matrix A can be factored into two matrices Q and R such that $A = QR$, where Q is an $m \times m$ unitary matrix and R is an $m \times m$ upper triangular matrix. The matrix Q can be computed using sequences of Givens rotations. An elementary Givens transformation has the form of

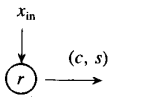
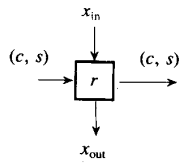
$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} 0 & \cdots & 0 & r_i & r_{i+1} & \cdots & r_k \\ 0 & \cdots & 0 & x_i & x_{i+1} & \cdots & x_k \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & r_i' & r_{i+1}' & \cdots & r_k' \\ 0 & \cdots & 0 & 0 & x_{i+1}' & \cdots & x_k' \end{bmatrix} \tag{6}$$

where

$$c = \frac{r_i}{\sqrt{r_i^2 + x_i^2}}, \qquad s = \frac{x_i}{\sqrt{r_i^2 + x_i^2}}.$$

Several different *QR* systolic arrays have been considered by Gentleman and Kung [9], Heller and Ipsen [14], and Luk [25]. In particular, the computation of the Q matrix without broadcast is difficult for the array considered in [24, p. 266]. On the other hand, [9] has shown that a triangular systolic array can be used to obtain the upper triangular matrix R based on sequences of Givens rotations. This approach also leads to an efficient method for performing recursive least squares computation [26], and is also useful for finding the singular value of a matrix [8]. This systolic array is shown in Fig. 1 and the operations of the cells are described in the first column (i.e.,

TABLE I
OPERATIONS OF THE PROCESSING CELLS FOR DIFFERENT PHASES

Cell \ Phase	Phase 1	Phase 2	Phase 3
	If $x_{in} = 0$ then $c \leftarrow 1$; $s \leftarrow 0$; otherwise $r' = \sqrt{r^2 + x_{in}^2}$; $c \leftarrow r/r'$; $s \leftarrow x_{in}/r'$ $r \leftarrow r'$; end	$s \leftarrow x_{in}/r$	$s \leftarrow x_{in} r$
	$x_{out} \leftarrow cx_{in} - sr$ $r \leftarrow sx_{in} + cr$	$x_{out} \leftarrow x_{in} - sr$	$s_{out} \leftarrow s_{in} + x_{in} r$

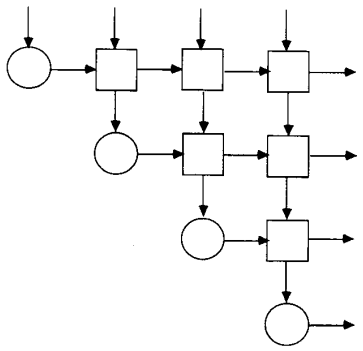


Fig. 1. Triangular systolic array for QR decomposition.

phase 1) of Table I. While the rotation parameters are propagated to the right, the Q matrix will not appear directly at the right as originally suggested by [36]. In order to demonstrate this point, denote G_{ij} as the Givens rotation matrix of the (i, j) plane. Then matrix Q can be obtained as

$$Q^T = \prod_{i=m-1}^{1} \prod_{j=i+1}^{m} G_{ij} \tag{7}$$

where Π is an ordered matrix produced defined by $\Pi_{i=m-1}^{1} C_i = C_{m-1} C_{m-2} \cdots C_1$, while Π denotes a conventional product, where the ordering of the terms are not relevant. From Table II, we can see, for a $n \times n$ QR triarray, the first rotation parameter coming out at the right edge occurs at a time $n + 1$. After that, rotation parameters for different plane rotation come out successively. If assuming that the operation of Π discussed above can be obtained immediately, then there are $m - 1$ operations of Π to be processed when all of Π are available and need to be multiplied. This observation leads to the conclusion that we cannot obtain the Q matrix by cumulatively multiplying the rotation parameters propagated to the right edge unless an additional rectangular array is used to accumulate the rotation matrices. Thus, this is not an efficient way to obtain the Q matrix.

B. Computation of $R^{-T}x$

In [6], Comon and Robert presented a rectangular systolic array for the computation of $B^{-1}A$, where B and A are square and rectangular matrices, respectively. The computation takes two phases. First, the matrix B is fed into the array and B^{-1} is computed. In the second phase, the matrix A is input to produce $B^{-1}A$. For the special case where B is an upper triangular matrix denoted by R, instead of a full dense matrix, McWhirter and Shepherd [27] used the property that a triangular array can compute $R^{-T}x$ in one phase with the matrix R prestored in the triarray. Since this property is needed in phase 2 of our work, and no full derivation was given in [27], we present a brief derivation of this result. Define $r_{ij} = (R)_{ij}$ and $r'_{ij} = (R^{-1})_{ij}$, where $r_{ij} = 0$ and $r'_{ij} = 0$ for $i > j$, then it can be shown that

$$r'_{ij} = \begin{cases} 1/r_{ii}, & i = j \\ -\sum_{k=i}^{j-1} r'_{ik} r_{kj} / r_{jj}, & i < j \le n. \end{cases} \tag{8}$$

Let $[y_1, \cdots, y_n]^T = R^{-T}x$, then

$$y_j = \sum_{i=1}^{j} x_i r'_{ij}, \qquad j = 1, \cdots, n. \tag{9}$$

In particular, y_i can be expressed in terms of r_{ij} as

$$\begin{aligned} y_j &= \frac{1}{r_{jj}} \cdot \left(x_j - \sum_{i=1}^{j-1} x_i \sum_{k=i}^{j-1} r'_{ik} r_{kj} \right) \\ &= \frac{1}{r_{jj}} \cdot \left(x_j - \sum_{k=1}^{j-1} \sum_{i=1}^{k} x_i r'_{ik} r_{kj} \right). \end{aligned} \tag{10}$$

From (9), y_j is given by

$$y_j = \underbrace{\frac{1}{r_{jj}}}_{(1)} \underbrace{\left(x_j - \sum_{k=1}^{j-1} y_k r_{kj} \right)}_{(2)}. \tag{11}$$

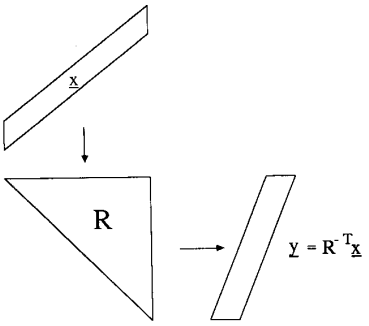


Fig. 2. Computation of $R^{-T}x$ using a triarray.

Thus, y_i can be computed recursively according to the above equation in the following algorithm:

Recursive algorithm for computing $y = R^{-T}x$
$y_1 = 1/r_{11} \cdot x_1$
for $j = 2$ to n
  begin
  $z_j = x_j$
  for $k = 1$ to $j - 1$
    $z_j = z_j - y_k r_{kj}$
  $y_j = z_j / r_{jj}$
end.

The corresponding systolic array to implement the above algorithm is the same as the one shown in Fig. 1. The operations of the cells are shown in the second column of Table I. The first part of (11), i.e., the division, is performed by the boundary cell while the second part of (11) is accumulated by the internal cells. With R prestored in the triarray, Fig. 2 shows the data flow of the input x and the output y.

C. Triangular-Matrix Multiplication

The multiplication of a triangular matrix R and an rectangular full dense matrix B is given by

$$(C)_{ij} = (RB)_{ij} = \sum_{k=i}^{n} r_{ik} b_{kj} \tag{12}$$

where r_{ik} and b_{kj} are elements of matrices R and B. Using the same array as in Fig. 1, with R prestored in the triarray and the operations shown in the third column of Table I, this multiplication can be easily obtained if B is input row by row as in Fig. 3.

D. Matrix Multiplication

There are many ways to implement a full matrix–matrix multiplication in a systolic array [21]. In Fig. 4, we show a typical architecture that can be incorporated with the multiphase operations to obtain eigenvectors. With input matrices Q and A arranged as in Fig. 4, the matrix B^T, where $B = AQ$, will reside in the rectangular array when the computation is completed. Details on this issue will be discussed in later sections.

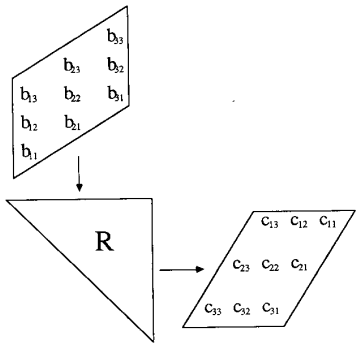


Fig. 3. Multiplication of a triangular matrix R and a rectangular full dense matrix B with $C = RB$ using a triarray.

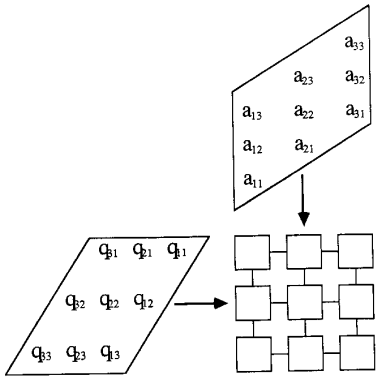


Fig. 4. Matrix–matrix multiplication in a rectangular array.

III. QR Algorithm

In this section we review briefly the basic operation of the QR algorithm and show the evaluation of the eigenvectors from the cumulative multiplication of successive Q matrices. For a complex-valued $n \times n$ matrix A, it states that there is a unitary transform U such that $R = UAU^H$ is a upper triangular matrix with diagonal eigenvalues of descending order. This follows from the QR algorithm [10], [40], [43] where by setting $A_1 = A$, we have $A_k = Q_k R_k$ and $A_{k+1} = R_k Q_k = Q_k^H A_k Q_k$, $k = 1, \cdots$, with unitary Q_k and upper triangular R_k. Furthermore, A_k converges to the upper triangular matrix with diagonal eigenvalue elements. However, it is not obvious how to compute the eigenvectors from those Q_k and R_k we have calculated. With a derivation similar to that used in [43], here we shows how to obtain the eigenvector associated with the largest eigenvalue from cumulative multiplications of Q_k. From the above discussions, we have

$$A_{k+1} = Q_k^H Q_{k-1}^H \cdots Q_1^H A_1 Q_1 Q_2 \cdots Q_k. \tag{13}$$

Define

$$\tilde{Q}_k = \coprod_{i=1}^{k} Q_i = Q_1 Q_2 \cdots Q_k$$

$$\tilde{R}_k = \coprod_{i=k}^{1} R_i = R_k R_{k-1} \cdots R_1. \tag{14}$$

Then we have

$$\tilde{Q}_k A_{k+1} = A_1 \tilde{Q}_k. \tag{15}$$

Thus the multiplications of $\tilde{Q}_k \tilde{R}_k$ can be expressed as

$$\begin{aligned}\tilde{Q}_k \tilde{R}_k &= Q_1 Q_2 \cdots Q_k R_k R_{k-1} \cdots R_1 \\ &= \tilde{Q}_{k-1} A_k \tilde{R}_{k-1} = A_1 \tilde{Q}_{k-1} \tilde{R}_{k-1} = A_1^k = A^k. \end{aligned} \tag{16}$$

Let the eigenvalues of A satisfy, $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$. Denote the matrix eigenvectors and eigenvalues of A by X and Λ, respectively. Then A^k is given by

$$A^k = X \Lambda^k X^{-1}. \tag{17}$$

Let the QR decomposition of X be $X = QR$ and the LU decomposition of X^{-1} be $X^{-1} = LU$, where L is a unit-lower triangular matrix. Then

$$A^k = QR\Lambda^k LU = QR(\Lambda^k L \Lambda^{-k}) \Lambda^k U \tag{18}$$

where

$$\Lambda^k L \Lambda^{-k} = I + E_k \tag{19}$$

and

$$(E_k)_{ij} = \begin{cases} l_{ij}(\lambda_i/\lambda_j)^k, & i > j \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

Since we have $\lim_{k \to \infty} E_k = 0$ thus $\Lambda^k L \Lambda^{-k}$ approaches the identity matrix. Then (18) can be rewritten as

$$A^k \to QR\Lambda^k U. \tag{21}$$

Since the term $R\Lambda^k U$ is an upper triangular matrix, comparing to (16) we can see that $\tilde{Q}_k \to Q$ when k is large. That is, the Q matrix of the QR decomposition of A^k approaches that of the Q matrix of the QR decomposition of the matrix of eigenvector X. Define

$$\begin{aligned}\tilde{Q}_k &= [\tilde{q}_1, \tilde{q}_2, \cdots, \tilde{q}_n] \\ X &= [x_1, x_2, \cdots, x_n] \end{aligned} \tag{22}$$

and r_{ij} as the (i, j) element of R. From $\tilde{Q}_k \to Q$, we find $r_{11}\tilde{q}_1 \to x_1$ when k is large. Since x_1 is the eigenvector associated with the largest eigenvalue, we conclude that the first column of the matrix $\tilde{Q}_k$ approaches the eigenvector with the largest eigenvalue of matrix A when k is large. If the matrix A is symmetric, which is often the case for signal processing applications, the similar transformation $A_{k+1} = \tilde{Q}_k^H A \tilde{Q}_k$ is also symmetric. Since A_{k+1} approaches the upper triangular matrix by the QR algorithm, A_{k+1} approaches a diagonal matrix. That is

$$A_k \to \Lambda \tag{23}$$

and

$$\tilde{Q}_k \Lambda \to X. \tag{24}$$

In this case, for large k, the columns of $\tilde{Q}_k$ become proportional to the columns of eigenvector in X.

If A is real, then A_k will converge to a real block upper triangular matrix with 1×1 and 2×2 main diagonal blocks. The complex conjugate pairs of eigenvectors of the 2×2 blocks can be solved easily using the quadratic formula. When A is not a square matrix, the singular values and vectors are of interest. For a $m \times n$ matrix B, where $m > n$, the SVD of B shows $B = U\Sigma V^T$, where U is a $m \times n$ matrix of orthogonal columns, V is a $n \times n$ unitary matrix, and Σ is a $n \times n$ diagonal matrix with diagonal singular values given in descending order. For many situations where high condition numbers are not encountered, a simple symmetric $n \times n$ matrix $C = B^T B$ can be formed and the matrix V can be found by direct use of the QR algorithm. Similarly, U can be found by using $D = BB^T$.

IV. Multiphase Systolic Algorithms

In this section, we introduce the multiphase systolic algorithms to compute the QR algorithm. Two arrays, triangular and rectangular, can be used to compute the QR algorithm with some advantages and disadvantages for each. We shall show that our methods compute the Q matrix explicitly without requiring any global communication within the array. Before we consider the multiphase algorithms, two communication switches are first discussed. A circular multiplexer is a device which takes its inputs and distributes them in different output positions as shown in Fig. 5. We use a skewed row to represent the circular multiplexor. A first in/first out (FIFO) buffer is a buffer which takes its input to output in a first in first out manner as shown in Fig. 6. Both devices are commonly used in computer and microprocessor systems for data arrangement [16]. The computation of a QR algorithm consists of two basic steps. Initially, set $A_1 = A$, then

1) for $k = 1, 2, \cdots$, compute $A_k = Q_k R_k$;
2) compute $A_{k+1} = R_k Q_k$, stop if converge, otherwise go back to step 1.

A. Multiphase Triangular Systolic Array

The QR decomposition triarray proposed by Gentleman and Kung [9] is used in our approach. The R matrix is stored in the triarray after the computation. To compute the matrix A_{k+1} in step 2, the Q_k matrix has to be computed first. Let us call the computations in step 1 and 2 an iteration. Several iterations are required for A_k to converge. For each iteration, we propose a three phase operation on a triarray as follows:

- *Phase 1: QR Decomposition for A_k:* Compute the QR decomposition of the matrix $A_k = Q_k R_k$, with the upper triangular matrix R_k being stored in the triarray [9]. The data in A_k is input row by row and skewed in time as shown in Fig. 7.
- *Phase 2: Computing the Q_k Matrix:* From the QR decomposition, we have $R_k^{-T} A_k^T = Q_k^T$. Let the ith column of matrices A_k^T and Q_k^T be denoted by a_i and q_i, respectively. Then

$$R_k^{-T}[a_1, a_2, \cdots, a_n] = [q_1, q_2, \cdots, q_n]. \tag{25}$$

Section II showed that $R_k^{-T} x$ can be computed in the same triarray that was used in phase 1. Since the ith column of

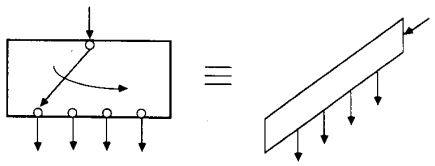


Fig. 5. A circular multiplexor.

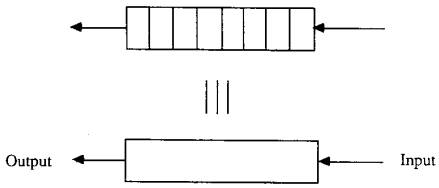


Fig. 6. A first in/first out buffer.

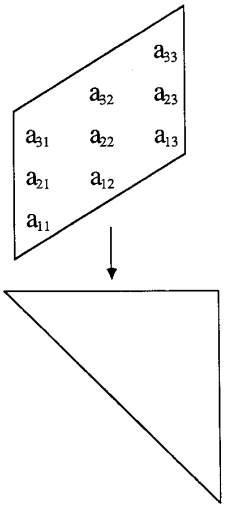


Fig. 7. Phase 1: The QR decomposition.

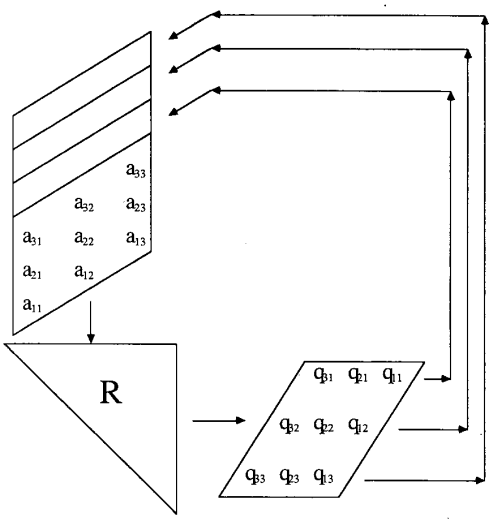


Fig. 8. Phase 2: Computing the Q matrix.

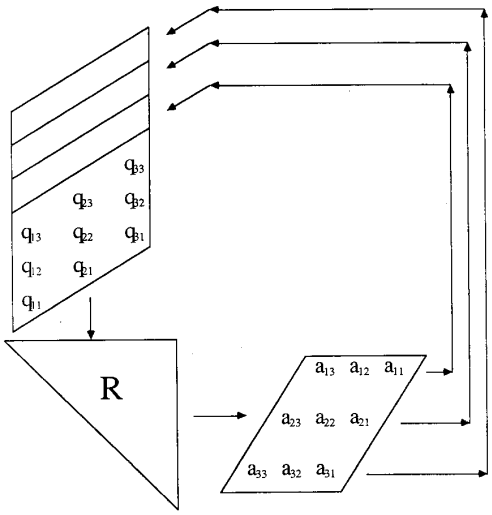


Fig. 9. Phase 3: Computing the matrix product RQ.

A_k^T is the ith row of A_k, then with A_k inputted row by row and skewed in time as shown in Fig. 8, the operations of the processing cells are given in the second column of Table I. The triarray computes the Q_k matrix of A_k. The matrix Q_k is then output row by row as shown in Fig. 8. In order to start phase 3, the matrix Q_k has to be in the form of Fig. 9. Observe that the output Q_k of phase 2 shares the same snapshot order as the desired arrangement of Q_k in phase 3 after a transpose operation. A circular multiplexer is used to distribute each column output of Q_k into row input as indicated in Fig. 8.

- *Phase 3: Computing R_kQ_k:* With the operations of the processing cell as shown in the third column of Table I and the Q_k obtained in phase 2, Fig. 9 shows the computation of $A_{k+1} = R_kQ_k$ in the triarray. Then the matrix A_{k+1} comes out column by column from the right side of the triarray. Again, we observe that A_{k+1} shares the same snapshot order as the desired arrangement of A_k in phase 1 after a transpose operation. If not convergent, a new iteration is repeated by feeding back A_{k+1} into the triarray after using a circular multiplexer as shown in Fig. 9. Then phase 1 operation begins as in Fig. 7.

An attractive property of this multiphase operation is that the feedback requirements of the matrices in different phases are identical. Thus, only a circular multiplexer is needed for each row outside the array. Observe that each column of the matrices input in all of the phases needs n time steps to process and the next phase can be started at time $n + 1$. We find once the result is output at the right-hand side of the triarray, after passing through the circular multiplexer, it can be piped into the array for the next phase computation without suffering any delay. If we assume the multiplexer is ideal such that the delay through it can be ignored, it takes $3n + (2n - 1) = 5n - 1$ system clocks for one iteration. The $(2n - 1)$ term represents the

initial time to feed the data into the array. If the number of iterations required for convergence is S, then the total number of system clocks needed is $3Sn + (2n - 1)$. Thus, the converge rate of this algorithm is of the order of $O((3S + 2)n)$. After the convergence of the A_k matrix, those values on the boundary cell are the eigenvalues of the A matrix.

B. Multiphase Rectangular Systolic Array

The above method requires the use of the R^{-T} operation in the computations. From a numerical stability point of view, we may want to consider an alternative that uses a square matrix for cumulative multiplication of the rotation parameters. Fig. 10 shows a square matrix which is an extended version of the Gentleman–Kung's triangular array with two delay elements (represented by black dots in Fig. 10) in the vertical communication links of the lower triangular part of the array. The processors in the lower triangular part are identical to the internal cells in the upper triangular part. In [32], Reilly *et al.* used the same array for a *QRD* array processing application. Denote

$$[A//I] = [\mathbf{a}_1; \mathbf{e}_1, \mathbf{a}_2; \mathbf{e}_2, \cdots, \mathbf{a}_n; \mathbf{e}_n] \tag{26}$$

as the parallel combination of matrix A and I, where $A = [\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n]$ and I is a $n \times n$ identity matrix with $\mathbf{e}_i$ as its ith column. The square array takes the input $[A//I]$. While rotating A into an upper triangular matrix, it uses I to accumulate the rotation parameters by

$$Q[A//I] = [R//Q]. \tag{27}$$

We note that processors in the upper triangular part not only rotate the matrix A but also cumulatively multiply the rotation parameters with I. Thus, its work load is, in general, twice that of Gentleman–Kung's internal cell. Processors in the lower part, on the other hand, only accumulate Q from the propagated rotation parameters. A two phase operation for *QR* iteration is proposed as follows:

- *Phase 1: QR Decomposition:* Compute the *QR* decomposition of matrix $A_k = Q_k R_k$; both Q_k and R_k are obtained and stored. Then each row of Q_k is piped out and fed back to the array through a FIFO buffer as shown in Fig. 10.
- *Phase 2: Computing $R_k Q_k$*: In this phase, the operation is identical to that of the phase 3 in the triangular array. A circular multiplexer is used to transform A_{k+1} from row output into column input. Continue this iteration until converged.

Due to the delay elements at the lower triangular part of the array and the work load of each processor (except those in the lower triangular part) being twice that of the triangular array, the time to obtain the ith row of the Q_k matrix, t_i, is

$$t_i = \max\,(2(n + 3i - 3), 2(2n + i - 2))$$

where $2(n + 3i - 3)$ is the time for the leftmost cell of the ith row to obtain its Q element and $2(2n + i - 2)$ is the time for the rightmost cell to finish. Obviously, when $i \geq \lceil n + 1/2 \rceil$, $t_i = 2(n + 3i - 3)$. Thus, the time required to obtain the whole Q matrix is $t_n = 8n - 6$. By assuming that it takes time n to sequentially pipe out the Q_k matrix, this algorithm takes $(9n - 6) + n + (2n - 1)$ to complete an iteration in the worst case. Again, denoting the number of iteration as S, this algorithm converge in the order of $O(S(10n - 6) + 2n - 1) = O((10S + 2)n)$. Of course, the performance can be improved by piping out each row of Q matrix when it is available instead of waiting for the whole Q matrix to be available. With this, the performance can reach to the order of $O((9S + 2)n)$.

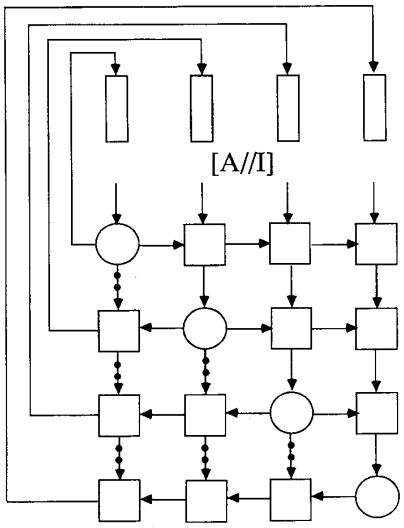


Fig. 10. Multiphase rectangular array for the *QR* iteration.

C. The Hessenberg Reduction

In order to perform the *QR* algorithm efficiently in conventional Von Neumann type serial computers, we usually transform the data matrix A into an upper Hessenberg matrix before applying the *QR* iteration. With this transformation, the amount of work per iteration is reduced from $O(n^3)$ to $O(n^2)$ [10]. However, this motivation may not be relevant for parallel processing architectures. The reasons for twofold:

1) Due to the hardware resources in a parallel processing architecture, the computations can be performed concurrently without hindering the processing time. For example, the computation time of the two multiphase arrays discussed above are of the order $O(n)$.

2) The data matrix is usually not in the Hessenberg form. The preprocessing of the data matrix to Hessenberg form may possibly not be incorporated with the following computations. That is, the preprocessing must be done separately.

Both reasons may lead to the conclusion that the Hessenberg form is not of practical interest in parallel processing of the *QR* algorithm unless the Hessenberg form

can be obtained easily by using the same parallel processing architecture. Many prior works avoid this issue by assuming that the Hessenberg form (or sometimes the tridiagonal form) is already available from the beginning. Fortunately, the Hessenberg form can be obtained easily in conjunction with the above proposed multiphase algorithms and architectures.

To obtain the Hessenberg form, we can choose a unitary similarity transformation U such that $A_1 = U^H A U$ is an upper Hessenberg matrix [10]. The transformation U can be obtained from sequences of Givens rotations. Denote G_i as the product of the Givens rotation matrices which zero out the proper positions of the ith column. Since the first i rows will not be affected by G_i, the matrix G_i is of the form $G_i = \text{diag}\,(I_i, \bar{G}_i)$, where I_i is an identity matrix of dimension i. Suppose the Hessenberg form through its first $k-1$ columns has been obtained

$$(G_1 \cdots G_{k-1})^H A (G_1 \cdots G_{k-1}) = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ 0 & B_{32} & B_{33} \end{bmatrix} \tag{28}$$

where B_{11} and B_{33} are $(k-1) \times (k-1)$ and $(n-k) \times (n-k)$ matrices, respectively. Then

$$(G_1 \cdots G_k)^H A (G_1 \cdots G_k) = \begin{bmatrix} B_{11} & B_{12} & B_{13}\bar{G}_k \\ B_{21} & B_{22} & B_{23}\bar{G}_k \\ 0 & \bar{G}_k^H B_{32} & \bar{G}_k^H B_{33} \bar{G}_k \end{bmatrix} \tag{29}$$

is a Hessenberg form through its first k columns. Thus

$$A_1 = (G_1 \cdots G_{n-1})^H A (G_1 \cdots G_{n-1}) \tag{30}$$

is an upper Hessenberg matrix and

$$U = G_1 \cdots G_{n-1} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & G \end{bmatrix}. \tag{31}$$

Denote

$$A = \begin{bmatrix} a^T \\ \bar{A} \end{bmatrix} = U\tilde{A} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & G \end{bmatrix} \cdot \begin{bmatrix} a^T \\ \tilde{R} \end{bmatrix} \tag{32}$$

where $\bar{A} = G\tilde{R}$ and $\tilde{R}$ is of the form of an upper triangular matrix without the lowest right element. Then $A_1 = \tilde{A}U = U^H A U$. Obviously, this is similar to the computations in the QR iteration. To obtain G and $\tilde{R}$, let

$$\hat{A} = \left[\begin{array}{c} \bar{A} \\ \hline 0, 0, \cdots, 1 \end{array}\right]. \tag{33}$$

The QRD of $\hat{A}$ is

$$\hat{A} = \hat{Q}\hat{R} = \begin{bmatrix} G & \mathbf{0}^T \\ \mathbf{0} & 1 \end{bmatrix} \cdot \begin{bmatrix} \tilde{R} \\ 0, 0, \cdots, 1 \end{bmatrix}. \tag{34}$$

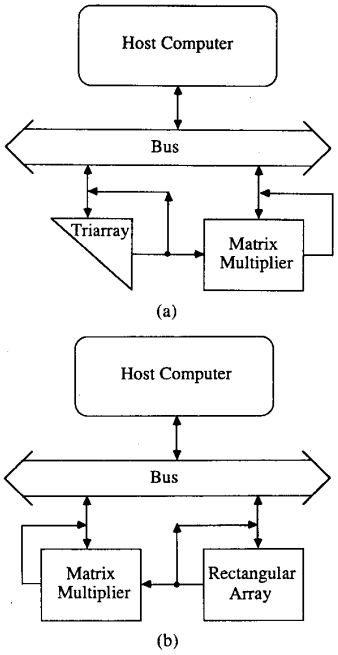


Fig. 11. (a) System configuration of the multiphase triarray. (b) System configuration of the multiphase rectangular array.

Now we can use the multiphase operations to obtain the upper Hessenberg form. We call this phase 0, with three internal operations.

- *Phase 0:* The Hessenberg reduction.

1) Use phase 1 operation for QRD of $\hat{A}$. From $\hat{A} = \hat{Q}\hat{R}$, we obtain $\tilde{R}$ in the triarray.
2) Use phase 2 operation for computing the G matrix. From $\hat{Q} = \hat{R}^{-T}\hat{A}^T$, we obtain matrix G.
3) Use phase 3 operation for computing the Hessenberg matrix A_1. By forming

$$\tilde{A} = \begin{bmatrix} a^T \\ \tilde{R} \end{bmatrix}$$

and

$$U = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & G \end{bmatrix}$$

we obtain the Hessenberg matrix $A_1 = \tilde{A}U$.

D. Computing the Eigenvectors

To compute an eigenvector, a matrix multiplication systolic array can be incorporated with the multiphase array such that those matrices $Q_1, \cdots, Q_k$ are accumulated to form the $\tilde{Q}_k$ matrix. Noted that $\tilde{Q}_k = \tilde{Q}_{k-1}Q_k$ and the matrix $\tilde{Q}_{k-1}$ is available at the start of the kth iteration, while the matrix Q_k is coming out at phase 2 operation of the kth iteration. Then $\tilde{Q}_k$ is obtained by multiplying $\tilde{Q}_{k-1}$ and Q_k as shown in Fig. 4. A system configuration for triangular array is shown in Fig. 11(a) and that for rectangular is shown in Fig. 11(b). As discussed in Sec-

tion III, for a symmetric A matrix, when A_k converged, $\tilde{Q}_k$ yields the matrix of eigenvectors. For a nonsymmetric A matrix, the first column of $\tilde{Q}_k$ yields the eigenvector associated with the largest eigenvalue.

V. Performance Comparisons

A. Comparisons of the Arrays

Although there are three phases of operations, the arithmetic operations in phases 2 and 3 form a subset of the operations executed in phase 1. Therefore we do not increase the cell complexity in the multiphase arrays. The performance and characteristics of both triangular and rectangular arrays considered above are summarized in Table III. The triangular array has several advantages: it requires less computational time and fewer cells, I/O ports, and communication devices. Furthermore, all of the processing cells are fully utilized. However, due to the computation of R^{-T} in phase 2 of the operation, it may be numerically less stable for certain highly ill-conditioned data. For example, consider the matrix given by

$$\begin{bmatrix} 0.7601 & -0.3967 & 0.6060 \\ -0.3967 & 1.7475 & -0.1962 \\ 0.6060 & -0.1962 & 0.4924 \end{bmatrix}$$

with eigenvalues $\{2.0, 1.0, 10^{-12}\}$. If the triarray algorithm (which uses R^{-T} to obtain Q^T) is used, the eigenvalues are obtained as $\{2.0, 1.0, 3.6818 \cdot 10^{-6}\}$. On the other hand, based on the rectangular array (where the R^{-T} is not explicitly computed), the eigenvalues are obtained as $\{2.0, 1.0, 9.9999 \cdot 10^{-13}\}$. All these results are obtained using MATLAB with double precision computations. As a result, we have a complexity versus numerical stability tradeoff for the two multiphase arrays.

B. Rate of Convergence

As in Luk [24], by convergence of the upper triangularity of A, we mean the parameter off (A_k) defined as

$$\text{off}\,(A_k) = \frac{\sum_{i<j} a_{ij(k)}^2}{N} \tag{35}$$

where N is a number of off-diagonal elements, has fallen below some prechosen tolerance value. As indicated in [24], it is difficult to monitor off (A_k) in the parallel computation. Luk then proposed that the iteration be stopped after a sufficiently large number S of iterations. In the studies of Brent and Luk [3], [24], they found that $S \leq 9$ for random symmetric matrices of order $n \leq 230$ and $S \leq 6$ for $n \leq 24$. Therefore, they chose $S = 10$ for $n \leq 100$ for the Jacobi-like method. Similar to their approach, we apply the QR algorithm to random $n \times n$ symmetric matrices (a_{ij}), where the elements a_{ij} for $1 \leq i \leq j \leq n$ were uniformly and independently distributed in $[-1, 1]$. The tolerance to meet the stopping condition is off $(A_k) \leq 10^{-10}$. We can see from Fig. 12 that the number of iterations for a QR algorithm to converge is in the order of 10 for matrix size smaller than 20×20. Even though we can reduce the matrix to Hessenberg form for full dense matrix or tridiagonal form for symmetric matrix, and the QR iteration with origin shift can accelerate the convergence rate [15], [39], [44], the number of iterations is still on the order of 10. As an example, the 4×4 tridiagonal matrix

$$\begin{bmatrix} 1 & 2 & 0 & 0 \\ 2 & 3 & 4 & 0 \\ 0 & 4 & 5 & 6 \\ 0 & 0 & 6 & 7 \end{bmatrix}$$

still requires eight iterations to converge when the symmetric QR algorithm is used, [10, p. 424]. This kind of property is not desirable for parallel processing implementation. It is known that the Jacobi-like method may require more flops than the symmetric QR algorithm. However, due to parallel implementation, many rotations may take place at the same time. The computations involved in QR algorithm and Jacobi-like method are generally of the same complexity. From these discussions, the one which requires fewer iterations is more attractive

TABLE II
The Timing Table for the Rotation Parameters to Reach the Right Edge of the QR Triarray

Time	$n+1$	$n+2$	$n+3$	$n+4$	$n+5$	$n+6$
First row	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)	(1, 7)
Second row			(2, 3)	(2, 4)	(2, 5)	(2, 6)
Third row					(3, 4)	(3, 5)

TABLE III
Comparisons of the Multiphase Triarray and Rectangular Array

	Triangular Array	Rectangular Array
Computation time	$O((3S+2)n)$	$O((10S+2)n)$ worst case
Numerical stability	fair	stable
Number of cells	$n(n+1)/2$	n^2 plus (n^2-n) d-elements
I/O ports	$2n$	$3n$
Utilization	1	<1
Communication devices	1	2

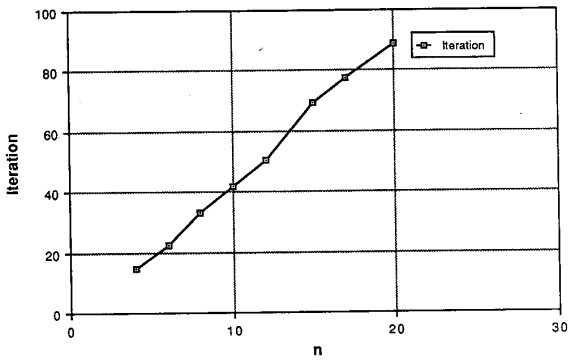


Fig. 12. The number of iterations for a QR algorithm to converge versus the matrix size.

from the parallel implementational point of view. Furthermore, the convergence rate of a QR iteration depends on the ratio of the eigenvalues. In our simulations, in more than 10% of the cases, the randomly generated symmetric matrices required significantly more iterations to converge. As pointed out before, it is difficult to monitor the quantity off (A_k) to decide when the algorithm converges in the parallel computations. Since the convergence rate is highly dependent on the ratio of eigenvalues, there is no general rule for choosing a sufficient number of iterations S to insure convergence. This is an undesirable intrinsic property of the QR algorithm for parallel implementation as compared to the Jacobi-like method.

VI. Efficient Fault-Tolerance Schemes

Reliable implementation is quite essential in parallel processing architectures. For a complex parallel processing system, a single fault from any part of the system can make the whole system useless. For various critical applications using spectral decomposition, highly reliable computations are demanded. Fault-tolerance is therefore needed in many of these problems. A simple and cost effective fault-tolerant scheme is the checksum and weighted checksum proposed by Abraham *et al.* [1]. This scheme is one of the typical examples of the algorithm-based fault-tolerance which has been applied to various signal processing and linear algebra operations [22]. Defining the checksum vector $e^T = [1, 1, \cdots, 1]$, the column, row and full checksum matrices A_c, A_r, and A_f of a square n-by-n matrix A are defined as

$$A_c = \begin{bmatrix} A \\ e^T A \end{bmatrix}$$

$$A_r = [A \quad Ae]$$

$$A_f = \begin{bmatrix} A & Ae \\ e^T A & e^T Ae \end{bmatrix}.$$

If any fault occurs during the computation, the checksum criterion is not met and thus the fault is detected. The weighted checksum scheme can be further used to correct errors [18]. It has been suggested in [5] that a (weighted) checksum scheme can be incorporated into the QR iterations for error detection. These properties as well as others are considered here for the multiphase arrays.

Since there are different operations in different phases, the inherent natures of the operations of each phase are thus different and should be examined for possible fault-tolerant implementation individually. The fault-tolerant schemes for each phase of the multiphase triangular array are given as follows:

- *Phase 1:* As pointed out in [5], [22], row checksum is invariant for the QR decomposition. It can be seen

$$A_r = [A \quad Ae] = Q[R \quad Re] = QR_r. \tag{36}$$

This means that the QR decomposition of a row checksum matrix results in a row checksum upper triangular matrix. Fig. 13 shows the implementation of this scheme.

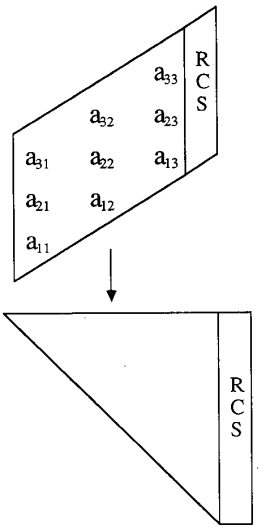


Fig. 13. Row checksum for $A_r = QR_r$.

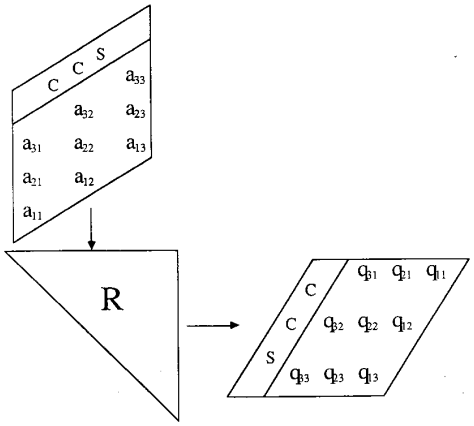


Fig. 14. Column checksum for $R^{-T}A_c^T = Q_c^T$.

- *Phase 2:* Due to the nature of computations in this phase, row checksum is no longer valid. Fortunately, column checksum is possible as given by

$$R^{-T}A_c^T = R^{-T}[A^T \quad A^T e] = [Q^T \quad Q^T e] = Q_c^T. \tag{37}$$

An implementation of this scheme is shown in Fig. 14.

- *Phase 3:* Although a row checksum upper triangular matrix R_r and a column checksum unitary matrix Q_c are obtained in the above phases, unfortunately, $R_r Q_c$ does not yield any relevant use. By defining the trace operation as the sum of the diagonal elements in a square matrix, we obtain tr $[AB]$ = tr $[BA]$, where A and B are square matrices. Therefore,

$$\text{tr}\,[A_{k+1}] = \text{tr}\,[R_k Q_k] = \text{tr}\,[Q_k R_k] = \text{tr}\,[A_k] = \sum_{i=1}^{n} \lambda_i \tag{38}$$

where λ_i is the eigenvalue of matrix A_k. This invariant property can be used to check the result of the phase 3 operation. If the trace of A_k is different from the trace obtained before, a fault is then detected during the phase 3 computation.

For the rectangular array, the phase 2 operation is the same as the phase 3 operation of the triangular array. For its phase 1 operation, an interesting feature of this computation is given by

$$Q[A_r//I_r] = [R_r//Q_r]. \tag{39}$$

That is, a row checksum of the parallel combination of matrices A and I gives a row checksum of the upper triangular matrix R_r and a row checksum of the unitary matrix Q_r.

VII. Conclusions

The multiphase systolic algorithms proposed in this paper can be used efficiently to solve the eigenvalue and SVD problems based on the QR algorithm. In particular, the eigenvectors can be obtained without global communication within the arrays using the multiphase operations. We showed that the QR algorithm can achieve a parallel implementation on a single architecture. Two systolic arrays, a triangular and a rectangular, are proposed for multiphase implementation. Efficient algorithm-based fault-tolerance schemes can be incorporated with both arrays easily. Since the operations in each phase belong to the same types of computation, the cell complexity is thus not increased by multiphase operations. There is a tradeoff between numerical stability and complexity for both arrays. Each iteration takes $O(n)$ time units while the time required for convergence is $O(Sn)$, where S is the number of iterations. Unlike the Jacobi-like method, the convergence rate of the QR algorithm depends on the ratio of the eigenvalues. As a consequence, S may vary for matrices of the same size, with or without origin shift to accelerate the convergence. Generally, S is in the order of 10 for the QR algorithm. From the parallel processing point of view, we have demonstrated the advantage of the QR algorithm that can yield two multiphase systolic algorithms implementable on single architecture without requiring global connections, while from the intrinsic convergence rate point of view, the QR algorithm is somewhat less attractive as compared to the Jacobi-like method. Depending on specific system and hardware requirements, one approach may be more desirable than the other. Of course, it is most meaningful to have two basic approaches to choose from for real-time VLSI signal processing based on spectral decomposition.

References

[1] J. A. Abraham *et al.* "Fault tolerance techniques for systolic array," *IEEE Comput. Mag.*, vol. 20, p. 65, July 1987.

[2] G. Bienvenue and H. F. Mermoz, "New principle of array processing in underwater passive listening," in *VLSI and Modern Signal Processing*, S. Y. Kung *et al.*, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[3] R. P. Brent and F. T. Luk, "The solution of singular-value and symmetric eigenvalue problems on multiprocessor array," *SIAM J. Sci. Stat. Comput.*, vol. 6, pp. 69–84, Jan. 1985.

[4] K. Bromley and J. M. Speiser, "Signal processing algorithm, architectures, and applications," *Proc. SPIE Int. Soc. Opt. Eng.*, vol. 431, pp. 2–6, 1983.

[5] C.-Y. Chen and J. A. Abraham, "Fault-tolerant systems for the computation of eigenvalues and singular values," *Proc. SPIE Int. Soc. Opt. Eng.*, vol. 696, 1986.

[6] P. Comon and Y. Robert, "A systolic array for computing BA^{-1}," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, pp. 717–723, June 1987.

[7] G. R. Gao and S. J. Thomas, "An optimal parallel Jacobi-like solution method for singular value decomposition," in *Proc. Int. Conf. Parallel Processing*, 1988, pp. 47–53.

[8] G. D. de Villiers, "A Gentleman–Kung architecture for finding the singular value of a matrix," in *Proc. Int. Conf. Systolic Array* (Ireland), 1989, pp. 545–554.

[9] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic array," *Proc. SPIE Int. Soc. Opt. Eng.*, vol. 298, p. 298, 1981.

[10] G. H. Golub and C. F. Van Loan, *Matrix Computation*, 2nd ed. Baltimore, MD: Johns Hopkins Press, 1989.

[11] S. Haykin, "Radar array processing for angle-of-arrival estimation," in *Array Signal Processing*, S. Haykin, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1985, pp. 194–292.

[12] S. Haykin, *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[13] D. E. Heller and I. C. F. Ipsen, "Systolic networks for orthogonal equivalence transformations and their application," presented at the 1982 Conf. Advanced Res. VLSI, M.I.T., Cambridge, MA, 1982.

[14] D. E. Heller and I. C. F. Ipsen, "Systolic networks for orthogonal decomposition," *SIAM J. Sci. Stat. Comput.*, vol. 4, pp. 261–269, June 1983.

[15] W. Hoffmann and B. N. Parlett, "A new proof of global convergence for the tridiagonal QL algorithm," *SIAM J. Numer. Anal.*, vol. 15, Oct. 1978.

[16] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*. New York: McGraw-Hill, 1984.

[17] I. Ipsen, "Singular value decomposition with systolic array," *Proc. SPIE Int. Soc. Opt. Eng.*, vol. 495, pp. 13–21, 1984.

[18] J.-Y. Jou and J. A. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures," *Proc. IEEE*, vol. 74, p. 732, May 1986.

[19] S. M. Kay, *Modern Spectral Estimation*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[20] K. Konstantinides and K. Yao, "Statistical analysis of effective singular values in matrix rank determination," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 757–736, May 1988.

[21] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[22] K. J. R. Liu and K. Yao, "Gracefully degradable real-time algorithm-based fault-tolerant method for QR recursive least squares systolic array," in *Proc. Int. Conf. Systolic Array* (Killarney, Ireland), May 1989, pp. 401–410.

[23] F. T. Luk, "A parallel method for computing the generalized singular value decomposition," *J. Parallel Distributed Computing 2*, pp. 250–260, 1985.

[24] F. T. Luk, "A triangular processor array for computing singular value," *Linear Alg. Its Appl.*, vol. 77, pp. 259–273, 1986.

[25] F. T. Luk, "A rotation method for computing the QR decomposition," *SIAM J. Sci. Stat. Comput.*, vol. 7, pp. 452–459, Apr. 1986.

[26] J. G. McWhirter, "Recursive least squares minimization using a systolic array," *Proc. Int. Soc. Opt. Eng.*, vol. 431, 1983.

[27] J. G. McWhirter and T. J. Shepherd, "An efficient systolic array for MVDR beamforming," in *Proc. Int. Conf. Systolic Array*, 1988, pp. 11–20.

[28] D. I. Moldovan, C. I. Wu, and J. A. B. Fortes, "Mapping arbitrary large QR algorithm into a fixed size VLSI array," in *Proc. Int. Conf. Parallel Processing*, 1984, pp. 365–373.

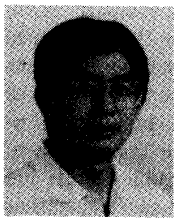
[29] J. H. Moreno and T. Lang, "A multilevel pipelined processor for the singular value decomposition," *Proc. SPIE*, 1986.

[30] N. L. Owsley, "Sonar array processing," in *Array Signal Processing*, Haykin, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1985, pp. 115–193.

[31] C. C. Paige, "Computing the generalized singular value decomposition," *SIAM J. Sci. Stat. Comput.*, vol. 7, Oct. 1986.

[32] J. P. Reilly, W. G. Chen, and K. M. Wong, "A fast QR-based array

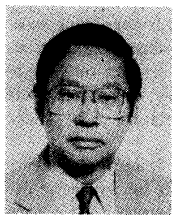
processing algorithm," *Proc. SPIE Int. Soc. Opt. Eng.*, vol. 975, pp. 36–47, 1988.
[33] W. Robertson and W. Phillips, "A systolic MUSIC system for VLSI implementation," in *Proc. IEEE ICASSP*, 1989, pp. 2577–2580.
[34] R. O. Schmidt, "A signal subspace approach to multiple emitter location and spectral estimation," Ph.D. dissertation, Stanford Univ., 1981.
[35] R. Schreiber, "Systolic array for eigenvalue computation," *Proc. SPIE Int. Soc. Opt. Eng.*, vol. 341, 1982.
[36] R. Schreiber, "Systolic linear algebra machines in digital signal processing," in *VLSI and Modern Signal Processing*, S. Y. Kung *et al.*, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1985, pp. 389–405.
[37] R. Schreiber, "Solving eigenvalue and singular value problems on an undersized systolic systolic array," *SIAM J. Sci. Stat. Comput.*, vol. 7, p. 441, Apr. 1986.
[38] R. Schreiber, "Implementation of adaptive array algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 1038–1045, Oct. 1986.
[39] G. W. Stewart, "Incorporating origin shifts into the *QR* algorithm for symmetric tridiagonal matrices," *Commun. Ass. Comput. Mach.*, vol. 13, June 1970.
[40] G. W. Stewart, *Introduction to Matrix Computations*. New York: Academic, 1973.
[41] R. A. Thisted, *Elements of Statistical Computing*. London: Chapman and Hall, 1988.
[42] N. Torralba and J. J. Navarro, "Size-independent systolic algorithms for *QR* iteration and Hessenberg reduction," in *Proc. Int. Conf. Systolic Array*, 1989, pp. 166–175.
[43] J. H. Wilkinson, *Algebraic Eigenvalue Problem*. New York: Oxford, 1965.
[44] J. H. Wilkinson, "Global convergence of tridiagonal *QR* algorithm with origin shift," *Linear Alg. Its Appl.*, vol. 1, pp. 409–420, 1968.
[45] K. J. R. Liu and K. Yao, "Spectral decomposition via systolic triarray based on *QR* iteration," in *Proc. IEEE ICASSP*, Apr. 1990, pp. 1017–1020.



KuoJuey R. Liu (S'86–M'90) received the B.S. degree in electrical engineering from the National Taiwan University, the M.S.E. degree in electrical engineering and computer science from the University of Michigan, Ann Arbor, in 1983 and 1987, respectively, and the Ph.D. degree in electrical engineering from the University of California, Los Angeles, in June 1990.

During 1983–1985, he served in the Signal Corps, Taiwan, as a Communications Officer. He then became a Teaching/Research Assistant at the University of Michigan and the University of California, Los Angeles. He is currently an Assistant Professor of electrical engineering in the Department of Electrical Engineering and Systems Research Center of the University of Maryland, College Park. His research interests include parallel processing algorithms and architectures for signal/image processing and communications, adaptive signal processing, video signal processing, fault-tolerant computing in VLSI systems, design automation for DSP VLSI systems, and fast algorithms.

Dr. Liu was awarded the President Research Partnership from the University of Michigan in 1987, and the University Fellowship and the Hortense Fishbaugh Memorial Scholarship from UCLA in 1987–1988 and 1989, respectively. He was also awarded the Outstanding Graduate Student Award in Science and Engineering from the Taiwanese-American Foundation.



Kung Yao (S'59–M'65–SM'91) was born in Hong Kong on November 24, 1938. He received the B.S.E. (Highest Honors), M.A., and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, in 1961, 1963, and 1965, respectively.

During the summers of his college years, he worked at the Princeton-Penn Accelerator in Penns Neck, NJ, the Brookhaven National Laboratory in Upton, NY, and the Bell Telephone Laboratories in Murray Hill, NJ. While attending Princeton University, he held a Princeton Engineering Predoctoral Fellowship and an IBM Fellowship. In 1965–1966, he was an NAS-NRC Postdoctoral Research Fellow at the University of California, Berkeley. Since September of 1966, he has been with the University of California, Los Angeles. Presently, he is a Professor in the Electrical Engineering Department. In the fall of 1969, he was a Visiting Assistant Professor at the Massachusetts Institute of Technology and a Visiting Senior Research Associate at the NASA Electronics Research Center, Cambridge, MA. In 1973–1974, he was a Visiting Associate Professor at Eindhoven Technical University in Eindhoven, the Netherlands. In 1985–1988, he served as an Assistant Dean of the School of Engineering and Applied Science at UCLA. His research interests include stochastic processes, digital communication theory, satellite communication systems, simulation, radar systems, systolic and VLSI algorithms and systems, and digital signal processing. He is the coauthor of a book, *Processing and Algorithms in Communication and Radar Systems*, now under preparation.

Dr. Yao is a member of Phi Beta Kappa, Sigma Xi, and the American Association for the Advancement of Science. He has served as Program Chairman, Secretary, and Chairman of the IEEE Information Theory Group in Los Angeles and served two terms as a member of the Board of Governors of the IEEE Information Theory Group. He was the Cochairman of the 1981 International Symposium on Information Theory held at Santa Monica, CA, and the representative of the IT-BOG in the organization of the 1987 IEEE Information Theory Workshop held at Bellagio, Italy. He was also the Chair of the Technical Program of the 1990 IEEE Workshop on VLSI Signal Processing. He has served as an Associate Editor for Book Reviews of the IEEE TRANSACTIONS ON INFORMATION THEORY and was a member of the Editorial Board of the journal, *Probability in the Engineering and Informational Sciences*, published by the Cambridge Press. He is an Associate Editor on VLSI Signal Processing for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS.