A Jacobi-like Algorithm for Computing the QR-Decomposition

Franklin T. Luk

TR 84-612
May 1984

Department of Computer Science
Cornell University
Ithaca, New York 14853

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

A Jacobi-like Algorithm for Computing the QR-Decomposition

Franklin T. Luk

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

ABSTRACT

A parallel Jacobi-like method for computing the QR-decomposition of an $n \times n$ matrix is proposed. It requires $O(n^2)$ processors and $O(n)$ units of time. The method can be extended to handle an $m \times n$ matrix $(m \geq n)$. The requirements become $O(n^2)$ processors and $O(m)$ time.

Keywords and Phrases: Systolic arrays, QR-decomposition, cyclic Jacobi method, real-time computation, VLSI.

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

1. Introduction

Let $A \in R^{n \times n}$ and

$$A = QR \tag{1.1}$$

(Q orthogonal, R upper triangular) be its QR-decomposition (QRD). The sequential computation of this decomposition requires time $O(n^3)$. Often, the QRD of $A \in R^{m \times n}$ $(m \geq n)$ is desired; the required time becomes $O(mn^2)$. For real-time signal processing (cf. Bromley and Speiser[6]) fast parallel algorithms are needed and various methods [1,2,7,8,9,11] (most of which are applicable only to square matrices) have been proposed in the literature. Ahmed, Delosme and Morf[1], Bojanczyk, Brent and Kung[2] and Gentleman and Kung[7] all use Givens rotations and a triangular array of $O(n^2)$ processors. Both [1] and [2] store Q (in product form) in the array and propagate R, whereas [7] stores R and propagates Q. The decomposition is computable in time $O(n)$. The technique in [7] can be applied to an $m \times n$ matrix, and it will use time $O(m)$. Heller-Ipsen[8] and Johnsson[9] both consider a banded matrix A, say with bandwidth w. Based on the Givens rotations, the technique of Heller and Ipsen requires time $O(n)$ and a rectangular array of wq processors, where q equals the number of subdiagonals of A. Johnsson[9] discusses a parallel implementation of the Householder transformations. He uses w processors and $O(nw)$ units of time. Sameh[11] considers an $m \times n$ matrix and a ring of p processors. He describes procedures based on the Givens and the Householder transformations; his algorithms require time $O(mn^2/p)$.

The parallel algorithms of Brent, Luk and Van Loan[4,5] for computing the ordinary and the generalized singular value decomposition (SVD) may require a preliminary QR-decomposition step. However, the mesh-connected multiprocessor arrays in [4,5] are very different from the QR-arrays in [1,2,7,8,9,11] and the interfacing of different arrays can be a serious problem. In this paper we present a Jacobi-like method that computes a QR-decomposition using a mesh-connected processor array. Our idea is to determine a QR-decomposition of an $n \times n$ matrix by iteratively computing in parallel $\lfloor n/2 \rfloor$ two-by-two QR-decompositions. This strategy of decomposing an n-by-n problem into $\lfloor n/2 \rfloor$ two-by-two subproblems has been used successfully by

Brent and Luk[3] for the symmetric eigenvalue decomposition, by Brent, Luk and Van Loan[4] for the SVD, and by Stewart[12] for the Schur decomposition. The algorithm in [3] (or [4]) divides an $n \times n$ matrix into blocks of 2×2 submatrices and assigns one processor to each block, resulting in a mesh-connected grid of $(\lceil n/2 \rceil)^2$ processors. The Schur decomposition array in [12] consists of two computational networks, each one quite similar to the multiprocessor array in [3] (or [4]). A total of approximately $n^2/2$ processors are needed (see Section 2 for a precise count). However, if we assign two nodes (one from each network) to a processor, we can simulate this complex array using a mesh-connected grid of processors (cf. O'Leary-Stewart[10]). Our QRD algorithm requires the Schur decomposition array, hence $O(n^2)$ processors.

In Section 2 we present our new algorithm and prove that it always converges after $2n$ time steps. The algorithm is extended to handle an $m \times n$ $(m \geq n)$ matrix in Section 3. The requirements become $O(n^2)$ processors and $O(m)$ time.

2. The Algorithm

We parallelize the computations by simultaneously triangularizing $\lfloor n/2 \rfloor$ two-by-two submatrices of $A \in \mathrm{R}^{n \times n}$. Consider the basic transformation: a QR-decomposition with column pivoting is computed of the 2×2 matrix

$$B = \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix} .$$

We get

$$\hat{B} \equiv JB\Pi = \begin{pmatrix} \hat{a}_{ii} & \hat{a}_{ij} \\ 0 & \hat{a}_{jj} \end{pmatrix} ,$$

where

$$J = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \text{ and } \Pi = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} .$$

The rotation parameters are calculated using the formulae:

$$h = \sqrt{a_{ij}^2 + a_{jj}^2} \, , \; c = a_{ij}/h \, , \; s = a_{jj}/h \, .$$

The full transformation on A is defined by

$$T_{ij} : A \rightarrow \hat{A} \equiv J_{ij} A \Pi_{ij} \, , \tag{2.1}$$

where J_{ij} denotes a plane rotation and Π_{ij} a permutation, both in the (i,j)-plane. The transformation T_{ij} will annihilate the (j,i)-element.

The pivot block is always taken from two contiguous diagonal elements of A, so as not to destroy any given upper triangular structure (see Lemma 5). A bonus (unimportant here) of this choice is that the sum of squares of the strictly lower triangular elements will decrease. More precisely, define

$$\sigma(A) \equiv \sum_{p>q} |a_{pq}|^2 .$$

The transformation $T_{i,i+1}$ will produce a matrix $\hat{A}$ satisfying

$$\sigma(\hat{A}) = \sigma(A) - |a_{i+1,i}|^2 .$$

Our procedure thus shares with other Jacobi methods (cf. [3,4,12]) the property that it drives the matrix to the desired form. An important difference is that our QRD algorithm converges in

$2n$ time steps (see Theorem 2), where one time step is defined as the time required to do one transformation T_{ij}. The orthogonal matrix Q is readily computable through accumulating the plane rotations. We present our new algorithm:

Algorithm QRD.

$Q \leftarrow I$;
for $t = 1, 2, \ldots, n$ do
begin
for $i = 1, 3, \cdots$ (i odd) do
begin $A \leftarrow J_{i,i+1} A \Pi_{i,i+1}$; $Q \leftarrow Q J_{i,i+1}$ end;
for $i = 2, 4, \cdots$ (i even) do
begin $A \leftarrow J_{i,i+1} A \Pi_{i,i+1}$; $Q \leftarrow Q J_{i,i+1}$ end
end.

The column pivotings are essential. For example, Algorithm QRD without pivoting will stagnate on a matrix with a zero subdiagonal:

$$A = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ \times & 0 & \times & \times \\ \times & \times & 0 & \times \end{pmatrix}.$$

Interestingly, the pivotings will nullify one another after $2n$ steps.

Lemma 1. All columns of A will return to their original positions after exactly $2n$ time steps.

Proof. A column vector moves forwards (backwards) after each time step until it gets to the first (last) position, where it stays for one step. It then reverses direction and moves again. The required number of steps for all columns to return to their initial positions equals the sum of $2n-2$ (for moving) and 2 (for the rest periods at the two ends). □

Figure 1 exhibits these interchanges for the case $n=6$. The numbers to the side are time steps, and the six numbers following them mark the positions of the original columns. A dash between two elements indicates an interchange that will take place at the next time step.

0. 1-2 3-4 5-6
1. 2 1-4 3-6 5
2. 2-4 1-6 3-5
3. 4 2-6 1-5 3
4. 4-6 2-5 1-3
5 6 4-5 2-3 1
6. 6-5 4-3 2-1
7. 5 6-3 4-1 2
8. 5-3 6-1 4-2
9. 3 5-1 6-2 4
10. 3-1 5-2 6-4
11. 1 3-2 5-4 6
12. 1-2 3-4 5-6

Figure 1. *Positions of original columns after each time step.*

We shall prove that the matrix is triangularized after at most $2n-2$ time steps. First, we introduce a notation and define a property indicating that a column vector is in "upper-triangular" form.

Notation. Let $A^{(0)} = A$ and denote by $A^{(t)}$ the matrix A after time step t. Set also

$$A^{(t)} \equiv (a_1^{(t)}, \ldots, a_n^{(t)}) \equiv (a_{ij}^{(t)}) \ . \ \square$$

Definition. We say $a_i^{(t)} \in U$ (the i-th column is in "upper-triangular" form) if elements $a_{i+1,i}^{(t)}, \ldots, a_{n,i}^{(t)}$ all equal 0. $\square$

Let us state and prove three lemmas.

Lemma 2. If $T_{n-1,n}$ is made at time step t, then $a_{n-1-i}^{(t+i)} \in U$, for $i = 0, 1, \ldots, n-2$.

Proof. The transformation $T_{n-1,n}$ annihilates the $(n,n-1)$-element. So $a_{n-1}^{(t)} \in U$. Now use induction. At time step $t+j$ $(j \geq 0)$ we perform $T_{n-1-j,n-j}$ so that column $a_{n-1-j}^{(t+j)} \in U$. At the next time step, the rotation in $T_{n-2-j,n-1-j}$ will create a new zero in the $(n-1-j,n-2-j)$-position, and the pivoting will bring zeros to the other subdiagonal positions of column $n-2-j$ from column $n-1-j$. Hence $a_{n-2-j}^{(t+j+1)} \in U$. $\square$

Lemma 3. If $T_{n-1,n}$ is made at time step t, then $a_{2i-1}^{(t+n-2)} \in U$ for $i = 1, 2, \ldots, \lfloor n/2 \rfloor$.

Proof. We perform the transformation $T_{n-1,n}$ at time steps t, $t+2$, $t+4$, $\cdots$. Hence $a_{n-1}^{(t+2j)} \in U$, for $j = 0, 1, \cdots$. Apply Lemma 2 to each of these vectors. □

Lemma 4. If $a_1^{(t)}, \ldots, a_i^{(t)}, a_{i+2}^{(t)}$ are all in U and $T_{i+1,i+2}$ is made at time step $t+1$, then $a_1^{(t+1)}, \ldots, a_{i+1}^{(t+1)}$ all belong to U.

Proof. We can check that our basic transformation (2.1) applied to any two consecutive columns in $\{a_1^{(t)}, \ldots, a_i^{(t)}\}$ will not move the resulting pair out of U. The transformation $T_{i+1,i+2}$ will put column $a_{i+1}^{(t+1)}$ in U (see the proof of Lemma 2). □

In words, after transformation $T_{n-1,n}$ the $(n-1)$-st column will satisfy the "upper-triangular" property. The column will then move left and pick up appropriate zero elements along the way (Lemma 2). The same event recurs every two time steps. Eventually all odd-numbered columns will be in "upper-triangular" form (Lemma 3). After that, the first two columns will be in U, then the first three columns, and so on (Lemma 4). We thus need only to determine when the transformation $T_{n-1,n}$ first occurs to compute the time at which the matrix becomes triangularized.

Theorem 1. The matrix $A^{(2n-3)}$ ($A^{(2n-2)}$) is upper triangular for n even (n odd).

Proof. For n even (n odd), we do the transformation $T_{n-1,n}$ at time step 1 (step 2). After $n-2$ additional time steps, all odd-numbered columns will be in U (Lemma 3). At the next time step, T_{23} is made. We need $n-2$ time steps for columns $2, 3, \ldots, n-1$ to get in U (Lemma 4). □

It is now obvious why we have restricted the pivot block to only contiguous elements.

Lemma 5. Let $A^{(t)}$ be upper triangular. Then $A^{(t+1)}$ stays upper triangular.

Proof. Apply $T_{i,i+1}$ $(1 \leq i < n)$ to $A^{(t)}$. Columns $a_i^{(t+1)}$, $a_{i+1}^{(t+1)}$ still belong to U. □

Since each column of A will return to its original position after $2n$ steps, we have proved our principal result:

Theorem 2. Algorithm QRD computes a QR-factorization of A after exactly $2n$ steps.

Figure 2 shows how a 6×6 matrix is triangularized after 9 steps. Steps 10 to 12 are necessary to return all columns to their original positions.

```
1. xxxxxx    2. xxxxxx    3. xxxxxx
   0xxxxx       xxxxxx       0xxxxx
   xxxxxx       x0xxxx       xxxxxx
   xx0xxx       xxxxxx       xx0xxx
   xxxxxx       xxx0xx       xx0xxx
   xxxx0x       xxx0xx       xx0x0x

4. xxxxxx    5. xxxxxx    6. xxxxxx
   xxxxxx       0xxxxx       0xxxxx
   x0xxxx       0xxxxx       00xxxx
   x0xxxx       0x0xxx       00xxxx
   x0x0xx       0x0xxx       00x0xx
   x0x0xx       0x0x0x       00x0xx

7. xxxxxx    8. xxxxxx    9. xxxxxx
   0xxxxx       0xxxxx       0xxxxx
   00xxxx       00xxxx       00xxxx
   000xxx       000xxx       000xxx
   000xxx       0000xx       0000xx
   000x0x       0000xx       00000x

10.xxxxxx   11.xxxxxx   12.xxxxxx
   0xxxxx       0xxxxx       0xxxxx
   00xxxx       00xxxx       00xxxx
   000xxx       000xxx       000xxx
   0000xx       0000xx       0000xx
   00000x       00000x       00000x
```

Figure 2. *The zero–nonzero structure after each time step.*

To implement the algorithm we associate a processor with each 2×2 block of four contiguous elements. The architecture is same as the Schur decomposition array introduced in Stewart[12] and detailed in O'Leary-Stewart[10]. There are $(n^2 + 2n - 6)/2$ processors for n even and $(n^2 + 2n - 3)/2$ processors for n odd. Only nearest neighbor connections are required of the processors, since each needs only to receive rotations from some of its neighbors, apply them and pass them on to other neighbors. We do not assume broadcasting of the rotation parameters and so each cluster of rotations requires $(\lceil n/2 \rceil - 1)$ time steps to pass completely out of the matrix. Since the clusters follow each other at intervals of two time steps and the last ($2n$-th) cluster begins at step $4n-1$ our algorithm will require $(\lceil 9n/2 \rceil - 2)$ time steps. The rotations propagate through the matrix as shown in Figure 3 [12].

1. 11xxxxxx	2. xx11xxxx
11xxxxxx	xx11xxxx
xx11xxxx	11xx11xx
xx11xxxx	11xx11xx
xxxx11xx	xx11xx11
xxxx11xx	xx11xx11
xxxxxx11	xxxx11xx
xxxxxx11	xxxx11xx
3. xxxx11xx	4. x22xxx11
x22x11xx	2xx22x11
x22xxx11	2xx22xxx
xxx22x11	x22xx22x
11x22xxx	x22xx22x
11xxx22x	xxx22xx2
xx11x22x	11x22xx2
xx11xxxx	11xxx22x
5. 33x22xxx	6. xx33x22x
33xxx22x	xx33xxx2
xx33x22x	33xx33x2
2x33xxx2	33xx33xx
2xxx33x2	xx33xx33
x22x33xx	2x33xx33
x22xxx33	2xxx33xx
xxx22x33	x22x33xx

Figure 3. *Propagations of the rotations.*

As mentioned in the Introduction, we are motivated to find an algorithm for a square processor array so that we need not interface different arrays in an SVD computation[4,5]. All other quadratic QRD arrays[1,2,7] are triangular structures composed of $n(n+1)/2$ processors (including the n delay registers in [1,2]) and they require $3n-2$ units of time for the computations. Admittedly, Algorithm QRD needs a little more time and the array structure is slightly more complex. But the possibility of computing an SVD using just one programmable array of processors justifies the additional efforts.

It is of independent interest to compare the methods in [1,2,7] with Algorithm QRD. The methods of Ahmed et al.[1] and of Gentleman-Kung[7] annihilate elements of A from top down by chess knight moves, while the method of Bojanczyk et al.[2] performs the Givens rotations from bottom up by "long" chess knight moves. These methods all require $3n-5$ "sweeps", whereas only $2n$ "sweeps" are required by Algorithm QRD (but each "sweep" must be separated by one unit of time). Our new algorithm creates zeros in a rather unusual manner. The precise orderings (they differ depending on whether n is even or odd) are unimportant and will be omitted. Figure 4 illustrates the three different orderings for $n=8$.

×	×	×	×	×	×	×	×
1	×	×	×	×	×	×	×
2	4	×	×	×	×	×	×
3	5	7	×	×	×	×	×
4	6	8	10	×	×	×	×
5	7	9	11	13	×	×	×
6	8	10	12	14	16	×	×
7	9	11	13	15	17	19	×

(a) *Ahmed et al.* [1] and *Gentleman–Kung* [7].

×	×	×	×	×	×	×	×
7	×	×	×	×	×	×	×
6	9	×	×	×	×	×	×
5	8	11	×	×	×	×	×
4	7	10	13	×	×	×	×
3	6	9	12	15	×	×	×
2	5	8	11	14	17	×	×
1	4	7	10	13	16	19	×

(b) *Bojanczyk et al.* [2].

×	×	×	×	×	×	×	×
15	×	×	×	×	×	×	×
14	16	×	×	×	×	×	×
13	15	11	×	×	×	×	×
12	14	10	16	×	×	×	×
11	13	9	15	7	×	×	×
10	12	8	14	6	16	×	×
9	11	7	13	5	15	3	×

(c) *Algorithm QRD.*

Figure 4. *Three different orders of annihilations.*

3. Rectangular Matrices

In Section 2 we consider only square matrices. If the matrix A has more rows than columns, i.e., $A \in R^{m \times n}$ with $m \geq n$, then we may adopt one of two strategies. The first approach is to apply Algorithm QRD to the square matrix

$$\tilde{A} = (A \,|\, 0) \in R^{m \times m}.$$

We get

$$\tilde{A} = Q \begin{bmatrix} R & 0 \\ 0 & 0 \end{bmatrix},$$

where $R \in R^{n \times n}$ and so

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix}.$$

The procedure requires $O(m)$ time and $O(m^2)$ processors. Its advantage is that no additional hardware is needed, assuming that our processor array can handle $m \times m$ matrices. The problem is of course that this assumption may be wrong.

The second approach is appropriate for a very large m, particularly for $m >> n$. For simplicity, assume that our array can handle $2n \times 2n$ matrices and that

$$k = m/n$$

is an integer. Partition the matrix in the form:

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \cdot \\ \cdot \\ A_k \end{bmatrix},$$

where each block A_i is $n \times n$. We propose a procedure that eliminates the elements of A one block at a time:

Algorithm Block QRD.

Set $R_1 := A_k$;

For $i = 1, 2, \ldots, k-1$ do

Use Algorithm QRD to compute a $2n \times 2n$ QR-decomposition:

$$\begin{pmatrix} A_{k-i} & 0 \\ R_i & 0 \end{pmatrix} = Q_{i+1} \begin{pmatrix} R_{i+1} & 0 \\ 0 & 0 \end{pmatrix},$$

where $Q_{i+1} \in \mathrm{R}^{2n \times 2n}$ and $R_{i+1} \in \mathrm{R}^{n \times n}$. □

We get the QR-decomposition of A from

$$Q = \begin{pmatrix} I_{(k-2)n} & 0 \\ 0 & Q_2 \end{pmatrix} \begin{pmatrix} I_{(k-3)n} & 0 & 0 \\ 0 & Q_3 & 0 \\ 0 & 0 & I_n \end{pmatrix} \begin{pmatrix} I_{(k-4)n} & 0 & 0 \\ 0 & Q_4 & 0 \\ 0 & 0 & I_{2n} \end{pmatrix} \cdots \begin{pmatrix} I_n & 0 & 0 \\ 0 & Q_{k-1} & 0 \\ 0 & 0 & I_{(k-3)n} \end{pmatrix} \begin{pmatrix} Q_k & 0 \\ 0 & I_{(k-2)n} \end{pmatrix},$$

where I_{jn} denotes the identity matrix of order jn, and

$$R = \begin{pmatrix} R_k \\ 0 \\ \cdot \\ \cdot \\ 0 \end{pmatrix}.$$

The requirements are $O(n^2)$ processors and $O(m)$ time. Note that one may speed up the algorithm by using two or more processor arrays. With $\lfloor k/2 \rfloor$ arrays we need but $\lceil \log_2 k \rceil$ steps. Thus, excluding the costs of data input and output, the required time can be cut to $O(n \log k)$.

REFERENCES

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

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

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

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

[5] 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 (1983), pp. 66-71.

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

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

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

[9] L. Johnsson, *A computational array for the QR-method,* Proc. Conf. Adv. Research in VLSI, P. Penfield, ed., Artech House, Inc., Dedham, Mass., 1982, pp. 123-129.

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

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

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