CAC Document No. 9
DCS Report No. 487

PARALLEL COMPUTATION OF EIGENVALUES OF REAL MATRICES

by

David J. Kuck
Ahmed Sameh

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

November 1, 1971

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the U. S. Army Research Office-Durham under Contract No. DAHC04 72-C-0001.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

PARALLEL COMPUTATION OF EIGENVALUES OF REAL MATRICES

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Research Report

5. AUTHOR(S) (First name, middle initial, last name)

David J. Kuck and Ahmed Sameh

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
November 1, 1971	43	11

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DAHC04 72-C-0001 b. PROJECT NO. ARPA Order 1899 c. d.	CAC Document No. 9 DCS Report No. 487 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Copies may be obtained from the address given in (1) above.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	U.S. Army Research Office-Durham Duke Station Durham, North Carolina

13. ABSTRACT

~~This~~ paper describes the implementation of three standard matrix eigenvalue computation methods on an array machine with high efficiency. A brief description of the ILLIAC ~~IV~~ computer is provided as background material. Three major sections follow--the first two describe Jacobi and Householder algorithms for real symmetric matrices, and the third describes the QR algorithm for real nonsymmetric matrices. Each of these sections is divided into four parts. The theoretical background of the method is presented first. An ILLIAC ~~IV~~ implementation of the algorithm is presented and timing estimates are included. Next, the efficiency (ratio of number of computations actually performed to number computations possible in a given time) of the computation on an array machine is derived for primary memory contained matrices. Finally, eigenvalue computations for matrices too large to be contained in primary memory are discussed in terms of a head-per-track secondary storage device. It is shown that for Jacobi and Householder algorithms, parallel computation efficiencies of 90% are possible, while those of the QR algorithm are rather low.

DD FORM 1 NOV 68 1473

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Mathematics of Computation (General)						

ABSTRACT

This paper describes the implementation of three standard matrix eigenvalue computation methods on an array machine with high efficiency. A brief description of the ILLIAC IV computer is provided as background material. Three major sections follow--the first two describe Jacobi and Householder algorithms for real symmetric matrices, and the third describes the QR algorithm for real nonsymmetric matrices. Each of these sections is divided into four parts. The theoretical background of the method is presented first. An ILLIAC IV implementation of the algorithm is presented and timing estimates are included. Next, the efficiency (ratio of number of computations actually performed to number of computations possible in a given time) of the computation on an array machine is derived for primary memory contained matrices. Finally, eigenvalue computations for matrices too large to be contained in primary memory are discussed in terms of a head-per-track secondary storage device.

It is shown that for Jacobi and Householder algorithms, parallel computation efficiencies of 90% are possible, while those of the QR algorithm are rather low.

TABLE OF CONTENTS

1. INTRODUCTION

The implementations on a parallel computer of three of the most commonly used techniques for finding the eigenvalues and eigenvectors of matrices are discussed. These are Jacobi's, Householder's and the QR method. Jacobi's algorithm for real symmetric matrices was modified in order to take advantage of parallelism in such a way that instead of two, only one orthogonal transformation would eliminate n off-diagonal elements, where n is the size of the matrix. Storage schemes for all three methods on the parallel computer are discussed in detail. It is well known that Jacobi's algorithm proved to be highly efficient for parallel computation; yet, its use is recommended primarily when evaluation of all the eigenvalues is required. Even then, if high accuracy is also desired, it should be used in conjunction with a scheme for establishing machine bounds for the eigensystem.

Householder's algorithm for tridiagonalization of a real symmetric matrix has also proven to be very efficient on a parallel computer; a parallel modification of the method of bisection can be used on the resulting tridiagonal matrix to obtain one or two eigenvalues or to investigate the distribution of all of them. If, however, all the eigenvalues are required, the QR method is usually preferred. The QR algorithm is discussed here only for nonsymmetric matrices. Implementation of this method on the parallel machine proved to be less efficient than the other two, though it is the most reliable for obtaining the eigenvalues of any matrix. For Householder's or the QR algorithm, the eigenvectors of the tridiagonal or the upper Hessenberg matrices may be obtained by the method of inverse iteration. If the transformation matrices used in reducing the matrix to the condensed form are stored, the eigenvectors of the original matrix can be readily computed.

2. ILLIAC IV OVERVIEW

2.1 System Organization

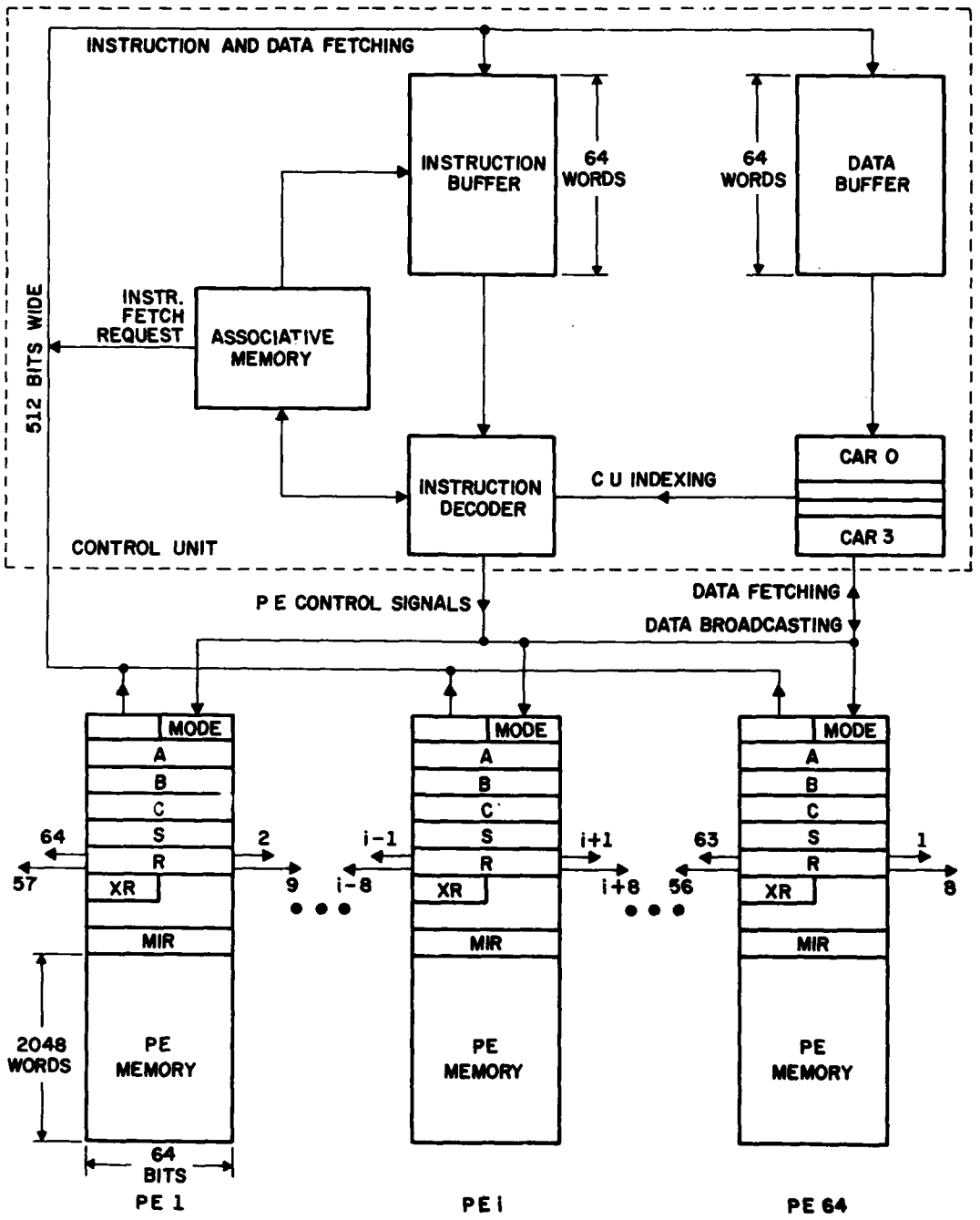
ILLIAC IV is an array of coupled computers driven by instructions from a common control unit.[1] Each of the processing elements (PE's) has 2048 words of 64-bit memory with a cycle time under 350 nanoseconds. Each PE is capable of 64-bit floating point multiplication in about 500 nanoseconds and addition in about 350 nanoseconds. Two 32-bit floating point operations may be performed in each PE in approximately the same time. The PE instruction set is similar to that of conventional machines, with two exceptions. First, the PE's are capable of communicating data to four neighboring PE's by means of routing instructions. Second, the PE's are able to set their own mode registers to effectively disable or enable themselves.

Figure 1 shows 64 PE's, each having three arithmetic registers (A, B, and C) and one protected programmer register (S). The registers, words, and paths in Figure 1 are all 64 bits wide, except the PE index registers (XR), mode registers, and as noted. The mode register may be regarded as one bit which may be used to block the participation of its PE in any action. The routing registers are shown connected to neighbors at distances of ± 1 and ± 8; similar end-around connections are provided at 1, 64, etc. Programs and data are stored in PE memory. Instructions are fetched by the control unit (CU) as required from PE memory.

Figure 1 also shows the essential registers and paths in the CU and their relations to the PE's. Instructions are decoded and control signals are sent to the PE array from the control unit. Some instructions are executed directly in the CU, e.g., the loading of CU accumulator registers (CAR's) with program literals. Operand addresses may be indexed once in the CU by a CAR and again separately in each PE by XR. It is possible to load the local data buffer (64 words of 64 bits each) and CAR's from PE memory. Local data buffer registers and CAR's may be loaded from each other. A broadcast instruction allows the contents of a CAR to be transmitted simultaneously to all PE's. It

[1] A 64-PE ILLIAC IV system is expected to be operating soon. For a more complete description of the system see [1].

FIGURE 1


INSTRUCTION AND DATA FETCHING
INSTRUCTION BUFFER
64 WORDS
64 WORDS
DATA BUFFER
512 BITS WIDE
INSTR. FETCH REQUEST
ASSOCIATIVE MEMORY
INSTRUCTION DECODER
C U INDEXING
CAR 0
CAR 3
CONTROL UNIT
P E CONTROL SIGNALS
DATA FETCHING
DATA BROADCASTING
MODE
A
B
C
S
R
XR
MIR
64
57
2
9
i-1
i-8
i+1
i+8
63
56
1
8
2048 WORDS
PE MEMORY
64 BITS
PE 1
PE i
PE 64

is often convenient to manipulate all PE mode bits or a number from one PE in a CAR. For this purpose, the broadcast path is bi-directional.

The complete ILLIAC IV system includes a Burroughs 6500 which performs input/output operations, compilation, and contains an operating system to control ILLIAC IV. There is a 10^9 bit, head-per-track disk with a 40 millisecond rotation speed and an effective transfer rate of 10^9 bits/second. This allows the loading of the 8×10^6 bit ILLIAC IV memory from the disk in 32 milliseconds. The input/output controller (IOC) contains a queuer for disk requests and 2^{18} bit buffer memory to smooth transmissions to and from the slower B6500 memory, or an external input/output device.

2.2 Programming ILLIAC IV

It is apparent that ILLIAC IV may be suited to the execution of algorithms defined on arrays of data (e.g., matrix operations and certain partial differential equations). However, there are two major difficulties in using such a machine. One is that an algoritnm must be devised which allows identical computations to be performed simultaneously on a large number of operands. The other, closely related to the first, is that the data must be so placed in the ILLIAC IV memory that over the course of the calculation few PE's are disabled. In other words, the algorithm and the data storage allocation must be mutually designed for highly parallel computation. For this reason, ILLIAC IV codes are expressed as two parts, a storage allocation block, followed by a computational algorithm block. These codes may be written in a high level language. See, for example, [2].

3. JACOBI'S METHOD

3.1 Theoretical Background

The classical Jacobi method reduces a symmetric matrix to the diagonal form by a series of orthogonal transformations, [3],

$$A_{r+1} = \Phi_r \, A_r \, \Phi_r^t \quad . \tag{0}$$

Each transformation eliminates two off-diagonal elements. It is possible, however, to eliminate n off-diagonal elements by one orthogonal transformation, where n is the order of the matrix A. This can be achieved if the transformation matrices Φ_r are of the form,

$$\Phi_r = \text{diag.} \left(T_0, T_1, \ldots, T_{\frac{n}{2}-1} \right)$$

assuming that n is even, and where,

$$T_k = \begin{bmatrix} \cos \alpha_k & \sin \alpha_k \\ -\sin \alpha_k & \cos \alpha_k \end{bmatrix} . \tag{1}$$

Therefore, the matrix A_{r+1} will consist of 2×2 submatrices of the form

$$\left(A_{pq}\right)_{r+1} = \left(T_p A_{pq} T_q^t\right)_r \quad , \qquad p,q = (0, 1, \ldots, \tfrac{n}{2} - 1) \text{ and } A_{pq}^t = A_{qp}.$$

For a diagonal submatrix,

$$\left(A_{kk}\right)_r = \begin{bmatrix} a_{2k,2k} & a_{2k,2k+1} \\ a_{2k,2k+1} & a_{2k+1,2k+1} \end{bmatrix}_{(r)} ,$$

if the elements of the transformation submatrix T_k are chosen such that,

$$\cos^2\alpha_k = \left[\frac{1}{2} + \left(Z_k\right)_r\right], \quad \sin^2\alpha_k = \left[\frac{1}{2} - \left(Z_k\right)_r\right], \quad \text{and } |\alpha_k| \leq \frac{\pi}{4} \tag{2}$$

where

$$\left(Z_k\right)_r = \frac{1}{2}\sqrt{1 \Big/ \left[\left(4a^2_{2k,2k+1} \Big/ \left(a_{2k+1,2k+1} - a_{2k,2k}\right)^2\right) + 1\right]_{(r)}} \tag{3}$$

then the diagonal submatrix $\left(A_{kk}\right)_{r+1}$ will be of the form:

$$\left(A_{kk}\right)_{r+1} = \begin{bmatrix} a_{2k,2k} & 0 \\ 0 & a_{2k+1,2k+1} \end{bmatrix}_{(r+1)}$$

where,

$$\left(a_{2k,2k}\right)_{r+1} = \left[a_{2k,2k}\cos^2\alpha_k + 2a_{2k,2k+1}\cos\alpha_k \sin\alpha_k + a_{2k+1,2k+1}\sin^2\alpha_k\right]_{(r)}$$

and, $\text{trace}\left(A_{kk}\right)_r = \text{trace}\left(A_{kk}\right)_{r+1}$.

Therefore, after one transformation, the off-diagonal elements

$\left(a_{pq}\right)_{r+1} = \left(a_{qp}\right)_{r+1}$, where $p = (0, 2, 4, \ldots, n-2)$, and $q = p + 1$,

are eliminated. In order to prepare the matrix for another transformation, the zero elements in the diagonal submatrices are replaced by carrying out the following permutation:

$$A'_{r+1} = \Psi A_{r+1} \Psi^t \tag{4}$$

where $\Psi = \begin{bmatrix} e_1^t & 0 \\ 0 & I \\ e_2^t & 0 \end{bmatrix}$, $\Psi\Psi^t = I$, [I] being the identity matrix, and

$e_1^t = (1,0)$, $e_2^t = (0,1)$.

In effect, this permutation shifts the second row to the bottom and the second column to the far right. The matrix A'_{r+1} is then ready for another transformation $A_{r+2} = \Phi_{r+1} A'_{r+1} \Phi^t_{r+1}$. It can easily be shown that all the off-diagonal elements will be exposed to this process of elimination after every (n - 1) orthogonal transformation, if the first row and first column are shifted to the bottom and far right, respectively. This second kind of shifting is equivalent to the permutation

$$A'_n = \Gamma A_n \Gamma^t \qquad (5)$$

where $\Gamma = \begin{bmatrix} e_2^t & 0 \\ 0 & I \\ e_1^t & 0 \end{bmatrix}$, $\Gamma\Gamma^t = I$.

To check for convergence the sum, E, of squares of the off-diagonal elements is compared to the sum, D, of squares of the diagonal elements after every orthogonal transformation. If the ratio $\eta = E/D$ is less than an arbitrarily small number ξ, then the diagonal elements of the matrix A_m are assumed to have converged to the eigenvalues of the original matrix A_1. In general, each diagonal element $(a_{ii})_m$ will lie in an interval of width $2\sqrt{E}$. The practically diagonal matrix A_m will be of the form:

$$A_m = W A_1 W^t$$

where

$$W^t = \left(\Phi_{m-1} \cdots \Phi_{n-1} \cdots \Phi_2 \Phi_1\right)^t$$

is the matrix whose columns are the eigenvectors.

3.2 Implementation

Consider the implementation of Jacobi's method on a 16-PE machine using the storage allocation scheme of Figure 2 for an upper triangular 16 x 16 matrix. The mapping of element a_{ij} into PE memory follows, where P_E (assumed even) is the number of PE's being used ($\lfloor x \rfloor$ is the greatest integer $\leq x$):

$$0 \leq i \leq \frac{P_E}{2} - 1$$

$$a_{ij} \rightarrow \text{loc. } i,\ PE \left[\sqrt{P_E}\left(j(\text{mod } \sqrt{P_E})\right) + \left\lfloor \frac{j}{\sqrt{P_E}} \right\rfloor + i(\sqrt{P_E} + 1)\right] (\text{mod } P_E)$$

$$\frac{P_E}{2} \leq i \leq P_E - 1$$

$$a_{ij} \rightarrow \text{loc. } j - \frac{P_E}{2} + 1,\ PE \left[\sqrt{P_E}\ \left(i(\text{mod } \sqrt{P_E} + 1\right) + \left\lfloor \frac{i - \frac{P_E}{2}}{\sqrt{P_E}} \right\rfloor + \left(j - \frac{P_E}{2}\right)(\sqrt{P_E} + 1) + 1\right] (\text{mod } P_E)$$

This storage allocation scheme is used because of the negligible amount of routing involved in implementing the Jacobi process. A step-by-step discussion of the process follows.

First, consider the calculation of the transformation matrices of Equation 1 according to Equations 2 and 3. The transformation matrices may be computed in the pairs of PE's shown in Figure 3. Proper a_{ij} elements may be accessed, as shown in Figure 3, in three memory cycles plus fewer than ten unit routes (distance 1 or 4) under mode control. This entire time is negli-

FIGURE 2

		PE 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOC. 0	ROW 0	0,0	0,4	0,8	0,12	0,1	0,5	0,9	0,13	0,2	0,6	0,10	0,14	0,3	0,7	0,11	0,15
1	1	1,14	1,3	1,7	1,11	1,15	8,8	1,4	1,8	1,12	1,1	1,5	1,9	1,13	1,2	1,6	1,10
2	2	2,9	2,13	2,2	2,6	2,10	2,14	2,3	2,7	2,11	2,15	8,9	2,4	2,8	2,12	9,9	2,5
3	3	3,4	3,8	3,12	9,10	3,5	3,9	3,13	10,10	3,6	3,10	3,14	3,3	3,7	3,11	3,15	8,10
4	4	11,11	4,7	4,11	4,15	8,11	4,4	4,8	4,12	9,11	4,5	4,9	4,13	10,11	4,6	4,10	4,14
5	5	5,13	10,12	5,6	5,10	5,14	11,12	5,7	5,11	5,15	8,12	12,12	5,8	5,12	9,12	5,5	5,9
6	6	6,8	6,12	9,13	13,13	6,9	6,13	10,13	6,6	6,10	6,14	11,13	6,7	6,11	6,15	8,13	2,13
7	7	7,7	7,11	7,15	8,14	12,14	7,8	7,12	9,14	13,14	7,9	7,13	10,14	14,14	7,10	7,14	11,14
8	8	10,15	14,15			11,15	15,15			8,15	12,15			9,15	13,15		

FIGURE 3

	PE 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ELEMENT a_{ij}	0,0	0,1	2,2	2,3	4,4	4,5	6,6	6,7	8,8	8,9	10,10	10,11	12,12	12,13	14,14	14,15
	1,1		3,3		5,5		7,7		9,9		11,11		13,13		15,15	
TRANSFORMATION MATRIX	T_0		T_1		T_2		T_3		T_4		T_5		T_6		T_7	

gible with respect to the times for the calculations which follows.

The computations of Equations 2 and 3 are carried out in the pairs of PE's shown in Figure 3. The time required for these is less than 15 microseconds.

The major part of the computation time is consumed in carrying out the transformations indicated by Equation 0. This computation involves a number of 2 x 2 matrix multiplications, and it is important to see that these may be performed efficiently using the storage allocation scheme of Figure 2. Notice that for $0 \leq i \leq 7$, a_{ij} and $a_{i,j+1}$ are, in blocks of four, four PE's apart, and can be brought to the same PE in one unit route. Across the boundaries of the blocks of four, elements are distance five apart. Similarly, for $0 \leq i \leq 6$, a_{ij} and $a_{i+1,j}$ are five PE's apart and can be brought to the same PE in two unit routes. For $8 \leq i \leq 15$, a_{ij} and $a_{i,j+1}$ are five PE's apart while for $8 \leq i \leq 14$, a_{ij} and $a_{i+1,j}$ are four PE's apart in blocks of four with five PE distance at the block boundaries. The distances between a_{7j} and a_{8j} are irregular but these rows do not interact while so labeled.

By moving the computed transformation matrices $T_0, \ldots, T_7$ to the control unit, pairs (T_0, T_1), (T_2, T_3), etc. can be broadcast to the appropriate PE's. Figure 4 shows the first two steps of this process. Line 1 shows the subscripts of the a_{ij} which may be accessed in one memory cycle and line 2 shows the subscripts of the A_{ij} partitions to which the a_{ij} of line 1 belong. By broadcasting T_0 and T_1 from the control unit, $T_0A_{00}T_0^t$, $T_0A_{01}T_1^t$, and $T_1A_{11}T_1^T$ are computed in parallel. Lines 3 and 4 show similar subscripts for the second step of the processs, during which $T_0A_{02}T_2^t$, $T_0A_{03}T_3^t$, $T_1A_{12}T_2^t$, and $T_1A_{13}T_3^t$ are computed. Since T_0 and T_1 were broadcast in step 1, they are still available in step 2. The 2 x 2 matrix multiplications are facilitated by spacing the adjacent row and column elements one or two unit routes apart. On the next step A_{04}, A_{05}, A_{14}, and A_{15} will be transformed. This processing is continued by rows until the entire matrix has been transformed. Notice that at the main diagonal this procedure suffers from an efficiency of approximately 50% if computed in the straightforward way shown above; this point will be discussed later.

The total time to carry out one transformation on a 64 x 64 upper triangular matrix is approximately 6 milliseconds using 64 PE's. The time

FIGURE 4

	PE 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
(1) ELEMENT a_{ij}	0,0	1,3	2,2						0,2	1,1		3,3	0,3	1,2		
(2) MEMBER OF PARTITION A_{ij}	0,0	0,1	1,1						0,1	0,0		1,1	0,1	0,1		
(3) ELEMENT a_{ij}	3,4	0,4	1,7	2,6	3,5	0,5	1,4	2,7	3,6	0,6	1,5	2,4	3,7	0,7	1,6	2,5
(4) MEMBER OF PARTITION A_{ij}	1,2	0,2	0,3	1,3	1,2	0,2	0,2	1,3	1,3	0,3	0,2	1,2	1,3	0,3	0,3	1,2

FIGURE 5

		PE 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOC. 0 ROW 8			1,9	1,13			1,10	1,14			1,11	1,15			1,12	1,1	
1	0	0,14	0,3	0,7	0,11	0,15	0,4	0,8	0,12	0,1	0,0	0,5	0,9	0,13	0,2	0,6	0,10
2	1	2,9	2,13	2,2	2,6	2,10	2,14	2,3	2,7	2,11	2,15	2,4	2,8	2,12	1,2	9,9	2,5
3	2	3,8	3,12	1,3	9,10	3,5	3,9	3,13	10,10	3,6	3,10	3,14	3,3	3,7	3,11	3,15	3,4
4	3	11,11	4,7	4,11	4,15	4,4	4,8	4,12	1,4	9,11	4,5	4,9	4,13	10,11	4,6	4,10	4,14
5	4	5,13	10,12	5,6	5,10	5,14	11,12	5,7	5,11	5,15	12,12	5,8	5,12	1,5	9,12	5,5	5,9
6	5	6,12	1,6	9,13	13,13	6,9	6,13	10,13	6,6	6,10	6,14	11,13	6,7	6,11	6,15	12,13	6,8
7	6	7,7	7,11	7,15	12,14	7,8	7,12	1,7	9,14	13,14	7,9	7,13	10,14	14,14	7,10	7,14	11,14
8	7	10,15	14,15	8,10	8,14	11,15	15,15	8,11	8,15	12,15	8,8	8,12	1,8	9,15	13,15	8,9	8,13

to transform a square 64 x 64 matrix is less than 12 milliseconds.

Next, the rows as shown in Equation 4 are permuted. The storage scheme of Figure 2 allows this to be done by moving relatively few elements of the matrix. Figure 5 shows the resulting matrix in which rows 0, 1, and 8 and columns 4, 8, and 12 have been moved. Each column is moved left one, row 0 is moved right three unit routes, and rows 1 and 8 are moved somewhat irregularly. However, the total amount of routing time involved is negligible with respect to the transformation time. In general, three rows and $\sqrt{P_E} - 1$ columns will be moved in a similar way.

Figure 6 is a relabeled version of Figure 5, showing the new names of the shuffled elements. Note by comparison with Figure 2 that the origin has moved nine PE's to the right and one row down in PE memory. This is a matter which is easy to accommodate in a program for carrying out the Jacobi process. At this point the entire process is repeated, started by computing a new set of transformations. After n - 1 such steps the second shuffle is performed according to Equation 5.

The second shuffle requires less routing than the first. Row 0's routing is similar to the routing of row 1 in the above discussion, with columns 4, 8, and 12 pushed back distance one to accommodate the new column.

3.3 Efficiency

First, an efficiency calculation for the simple process described above shall be performed, operating on a $(k\sqrt{P_E} \times k\sqrt{P_E})$ matrix using P_E PE's. Assuming one time unit to operate on a $(\sqrt{P_E} \times \sqrt{P_E})$ partition, $\sum_{i=1}^{k-1} = k(k-1)/2$ time steps is used for square partitions plus k time steps for triangular partitions on the main diagonal. If the elements which are to be made zero are not computed (just set to zero), only $\frac{k}{2}$ steps should be used to transform the triangular partitions on the main diagonal. Thus, there is an efficiency of

$$\frac{k(k-1)/2 + k/2}{k(k-1)/2 + k} = \frac{k}{k+1}.$$

For a 64 x 64 matrix and 64 PE s, (i.e., k = 8 and $\sqrt{P_E} = 8$), there is an efficiency of $\frac{8}{9} \approx 89\%$.

FIGURE 6

	PE 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOC.0 ROW 8		8,15	12,15			9,15	13,15			10,15	14,15			11,15	15,15	
1 0	0,13	0,2	0,6	0,10	0,14	0,3	0,7	0,11	0,15	0,0	0,4	0,8	0,12	0,1	0,5	0,9
2 1	1,8	1,12	1,1	1,5	1,9	1,13	1,2	1,6	1,10	1,14	1,3	1,7	1,11	1,15	8,8	1,4
3 2	2,7	2,11	2,15	8,9	2,4	2,8	2,12	9,9	2,5	2,9	2,13	2,2	2,6	2,10	2,14	2,3
4 3	10,10	3,6	3,10	3,14	3,3	3,7	3,11	3,15	8,10	3,4	3,8	3,12	9,10	3,5	3,9	3,13
5 4	4,12	9,11	4,5	4,9	4,13	10,11	4,6	4,10	4,14	11,11	4,7	4,11	4,15	8,11	4,4	4,8
6 5	5,11	5,15	8,12	12,12	5,8	5,12	9,12	5,5	5,9	5,13	10,12	5,6	5,10	5,14	11,12	5,7
7 6	6,6	6,10	6,14	11,13	6,7	6,11	6,15	8,13	12,13	6,8	6,12	9,13	13,13	6,9	6,13	10,13
8 7	9,14	13,14	7,9	7,13	10,14	14,14	7,10	7,14	11,14	7,7	7,11	7,15	8,14	12,14	7,8	7,12

13

FIGURE 7

	PE 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOC. 0	0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,10	0,11	0,12	0,13	0,14	0,15
1	1,12	1,13	1,14	1,15	15,15	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10	1,11
2	2,8	2,9	2,10	2,11	2,12	2,13	2,14	2,15	14,15	14,14	2,2	2,3	2,4	2,5	2,6	2,7
3	3,4	3,5	3,6	3,7	3,8	3,9	3,10	3,11	3,12	3,13	3,14	3,15	13,15	13,14	13,13	3,3
4	12,15	12,14	12,13	12,12	4,4	4,5	4,6	4,7	4,8	4,9	4,10	4,11	4,12	4,13	4,14	4,15
5	5,12	5,13	5,14	5,15	11,15	11,14	11,13	11,12	11,11	5,5	5,6	5,7	5,8	5,9	5,10	5,11
6	6,8	6,9	6,10	6,11	6,12	6,13	6,14	6,15	10,15	10,14	10,13	10,12	10,11	10,10	6,6	6,7
7	9,11	9,10	9,9	7,7	7,8	7,9	7,10	7,11	7,12	7,13	7,14	7,15	9,15	9,14	9,13	9,12
8	8,15	8,14	8,13	8,12	8,11	8,10	8,9	8,8								

This same technique and storage allocation scheme may be used on matrices of dimensions less than the number of PE's. The discussion holds of the transformation of A_{02}, A_{03}, A_{12}, and A_{13} operated on a $\sqrt{P_E} \times \sqrt{P_E}$ partition of a larger matrix. The process may be used repeatedly on a large matrix or just once on a $\sqrt{P_E} \times \sqrt{P_E}$ matrix. Full efficiency can be achieved on square matrices of dimension $k\sqrt{P_E} \times k\sqrt{P_E}$.

The maximum efficiency which can be achieved for the Jacobi process on a parallel machine depends on how the diagonal partitions and any columns in addition to $k\sqrt{P_E}$ are treated. Suppose an attempt is made to transform simultaneously A_{00}, A_{01}, A_{11}, A_{44}, A_{45}, and A_{55} of the example, and to subsequently repeat this pattern on the main diagonal. This procedure will achieve 100% efficiency during the transformation of the triangular matrices, but an irregular route of two elements will be incurred since a_{00} and $a_{11,11}$ are in the same PE as are a_{02} and $a_{9,11}$.

Matrices of dimension larger than 64 x 64 may be partitioned into a number of blocks of dimension 64 x 64 or smaller. For example, a 100 x 100 matrix becomes a 64 x 64 upper triangular partition, a 36 x 36 upper triangular partition and a 64 x 36 rectangular partition. All rows of rectangular partitions are stored according to the mapping given earlier for rows between 0 and $P_E/2 - 1$. In other words, the lower part of the matrix is not packed into holes in the upper part, but is stored in rows just like the upper part. In applying the Jacobi process to such a rectangular matrix, proceed as above transforming blocks of dimension $\sqrt{P_E} \times \sqrt{P_E}$ and achieving full efficiency on partitions of dimension $p\sqrt{P_E} \times q\sqrt{P_E}$. If the dimension of the matrix is not a multiple of $\sqrt{P_E}$, the leftover columns may be handled with 100% efficiency if the transformations are treated as column operations. Entire columns are available in one memory cycle using the storage allocation scheme described above.

Thus it seems reasonable to conclude that efficiencies of 90% or greater may be attained for any size matrix without inordinate programming difficulties.

3.4 Disk Considerations

Large matrices that do not fit into primary memory[2] are partitioned into blocks of 64 x 64. The number of rows of blocks is $NR = \left\lfloor \frac{N-1}{64} \right\rfloor + 1$, where $N > 64$ is the size of the matrix. Let the number of any block b be given by

$$b = R + \sum_{k=0}^{C-1} k \qquad (C,R = 1, 2, \ldots, NR)$$

where R is the row number of the block and C is the column number of the block. The blocks

$$b = R + \sum_{k=0}^{R-1} k, \; R = (1, 2, \ldots, NR),$$

are brought in from the disk sequentially to be subjected to an orthogonal transformation, Equation 0. Thus all the rotation angles are determined and the rest of the blocks,

$$b = R + \sum_{k=0}^{C-1} k, \; R = (0, 1, \ldots, NR) \; C \neq R$$

are fetched from the disk to complete the given transformation of the whole matrix. Some of the rows and columns of each submatrix are stored in arrays in primary memory, for purposes of the shifting process that takes place after the whole matrix is subjected to a transformation. Therefore, the following elements will be stored in primary memory for shifting the second row and second column:

Submatrix	Elements to be stored
1. $b = R + \sum_{k=0}^{R-1} k$ $(R > 1)$	first row of the submatrix, i.e., the elements a_{ij}. $i = 64(R - 1)$ $j = 64(R - 1), \ldots, 64R - 1$

[2] The results of this section are valid for various secondary storage devices, e.g., drum, delay line or the head per track disk which we discuss.

2. $b = R + \sum_{k=0}^{C-1} k$

$(R > 1,\ C > R)$

first row and first column of the submatrix, i.e., the elements $\left(a_{k\ell}\right)$ and $\left(a_{rs}\right)$.

$k = 64(R - 1)$

$\ell = 64(C - 1), \ldots, 64\ C - 1$

$s = 64(C - 1)$

$r = 64(R - 1), \ldots, 64\ R - 1$

3. $b = 1 + \sum_{k=0}^{C-1} k$

$C > 1$

second row and first column of each submatrix, i.e., the element $\left(a_{pq}\right)$ and $\left(a_{tu}\right)$.

$p = 1$

$q = 64(C - 1), \ldots, 64\ C - 1$

$u = 64(C - 1)$

$t = 0, \ldots, 63$

4. $b = 1$

second row and second column of the submatrix, i.e., the elements $\left(a_{vw}\right)$ and $\left(a_{xy}\right)$.

$v = 1$

$w = 1, \ldots, 63$

$x = 0$

$y = 1$

For the first row and first column shifting, the elements to be stored in the core are the same as above except those of items (3) and (4):

Submatrix	Elements to be stored
3'. $b = 1 + \sum_{k=0}^{C-1} k$	first row and column of each submatrix, i.e., the element (a_{pq}), (a_{tu}).
$C > 1$	$p = 0$ $q = 64(C - 1), \ldots, 64\,C - 1$ $u = 64(C - 1)$ $t = 0, \ldots, 63$
4'. $b = 1$	first row only, i.e., the elements (a_{vw}), $v = 0$ $w = 0, \ldots, 63$

The total number of elements to be stored in primary memory in each kind of shifting is $N(NR + 1)$.

The primary memory of 64 PE's is capable of holding 32 partitions 64 x 64; the transformation time for a square block is about 12 ms, and the rotational delay of the head-per-track disk is 40 ms. If the partitions on the disk are laid out in such a way that four of them are accessed at once, then 48 ms later the four 64 x 64 partitions will have been transformed and the disk will have rotated more than one complete revolution. Thus, if the requests for four partitions in the disk queuer are kept one rotation ahead, the rotational latency of the disk can be masked in computing on large matrices.

4. HOUSEHOLDER'S METHOD

4.1 Theoretical Background

This method [3,4] reduces a symmetric matrix $A = [a_{ij}]$ $(i,j = 0,1,2,\ldots,n-1)$ to the tridiagonal form T using elementary Hermitian orthogonal matrices. There are (n-2) steps in this reduction. We define the matrices A_r by the relations,

$$A_r = P_r A_{r-1} P_r \qquad (r = 1,2,\ldots,n-2) \tag{6}$$

where $A_o = A$ is the original matrix. The matrices P_r are defined by

$$P_r = I - \frac{u_r u_r^t}{2K_r^2} \tag{7}$$

in which u_r^t is a row vector (whose first r elements are zero) given by,

$$u_r^t = (0,\ldots,0,\ a_{r-1,r} \pm S_r,\ a_{r-1,r+1},\ldots,\ a_{r-1,n-1}) \tag{8}$$

$$S_r = \left(\sum_{i=r}^{n-1} a_{r-1,i}^2\right)^{\frac{1}{2}} \tag{9}$$

and,
$$2K_r^2 = S_r\,(S_r \pm a_{r-1,r}) \tag{10}$$

where the elements a_{ij} are those of A_{r-1}. The signs in both expressions (8) and (10) are chosen such that,

$$|a_{r-1,r} \pm S_r| = |a_{r-1,r}| + S_r\ . \tag{11}$$

Therefore, in the r-th step $(A_r = P_r A_{r-1} P_r)$ [n - (r + 1)] zeros are introduced in the r-th column and row of A_{r-1} without destroying the zeros introduced in the previous step.

In order to take advantage of symmetry, the matrices A_r may be defined by the relations [3],

$$A_r = A_{r-1} - q_r u_r^t - u_r q_r^t \tag{12}$$

where u_r is as given above, and q_r is given by

$$q_r = p_r - \frac{1}{2} \frac{u_r u_r^t}{2K_r^2} p_r \tag{13}$$

in which,

$$p_r = A_{r-1} \frac{u_r}{2K_r^2} . \tag{14}$$

The total number of multiplications required to reduce the matrix to the tridiagonal form is essentially $\frac{2}{3} n^3$.

The eigenvalues of the tridiagonal matrix may be obtained by several methods. For this parallel computer it is proposed that if all the eigenvalues are required, then the QR algorithm with origin shift [3,5,6] is used, (this algorithm is discussed for real nonsymmetric matrices in the following section). However, if one is interested in the general distribution of the eigenvalues and not in their accurate values, or the eigenvalues in a given interval, the bisection method [3,7] may be used. In order to show how the method of bisection would be used for a parallel computer, assume that it is necessary to obtain all the eigenvalues of the resulting tridiagonal matrix T using such a method. For the sake of illustration, assume also that all the subdiagonal elements are non-zero and hence the matrix has simple eigenvalues. The eigenvalues lie in the interval [-b,b], where $b = || T ||_\infty = \max_i \sum_j | t_{ij} |$. This interval is then divided into 63 subintervals and for all μ_k ($k = 1,a,\ldots,64$), where $\mu_1 = -b$, and $\mu_{64} = +b$, the quantities,

$$f_o(\mu_k) = 1$$

$$f_1(\mu_k) = t_{oo} - \mu_k \tag{15}$$

$$f_i(\mu_k) = (t_{i-1,i-1} - \mu_k)\, f_{i-1}(\mu_k) - t_{i-2,i-1}\, f_{i-2}(\mu_k) (i = 2,3,\ldots,n)$$

are evaluated in parallel, one PE to each μ_k. For each μ_k the number of agreements in sign $s(\mu_k)$ of consecutive members of the sequence $f_o(\mu_k)$, $f_1(\mu_k)$, ..., $f_n(\mu_k)$ is the number of eigenvalues larger than μ_k. The process is repeated for those intervals $[\mu_k, \mu_{k+1}]$ that contain more than one eigenvalue until all the eigenvalues are separated. Assuming that the matrix is of order 64, 64 intervals, each containing an eigenvalue are established. By assigning each interval to a PE and by successive bisection of each interval, using the Sturm sequence property, all the eigenvalues are obtained.

Once all the eigenvalues are available, the eigenvectors of the tridiagonal matrix are obtained by the method of inverse iteration [3,8]. Again this can be done in parallel, each PE evaluating an eigenvector. Finally if Z_i (i = 1,2,...,n) is an eigenvector of the tridiagonal matrix T, then the corresponding eigenvector Y_i of the original matrix A is computed by,

$$Y_i = P_1\, P_2 \cdots P_{n-2}\, Z_i. \tag{16}$$

4.2 Implementation of Householder Tridiagonalization

We shall consider the implementation of Householder's method on a 16-PE machine using the storage allocation scheme of Figure 7 for an upper triangular 16 × 16 matrix. This scheme is somewhat less complicated than Figure 2 since it is not necessary to provide for shuffling. In particular, adjacent row and column elements are always a unit route from each other, except between the seventh and eighth rows. The lower half of the matrix is stored in reverse order to allow efficient computation on this part of the matrix. This scheme also allows access to

any row with one memory fetch. Operations may be performed on $P_E \times P_E$ blocks at any step, where the size of the matrix n is equal to the number of PE's P_E.

The memory map for this scheme consists of two parts:
if $0 \leq i \leq \frac{P_E}{2} - 1$,

$$a_{ij} \rightarrow \text{loc } i, \text{ PE } \left[((i(\text{mod } \sqrt{P_E}))\sqrt{P_E} + j) \ (\text{mod } P_E) \right]; \qquad (17)$$

if $\frac{P_E}{2} \leq i \leq P_E - 1$

$$a_{ij} \rightarrow \text{loc } P_E - i, \text{ PE } \left[(P_E - 1) - ((i(\text{mod } \sqrt{P_E}))\sqrt{P_E} + j) \ (\text{mod } P_E) \right].$$

A step-by-step discussion of the process follows.

As the first step (r = 1), consider row 0 of the matrix, (a_{00}, a_{01}, . . ., $a_{0,15}$). This row is in location 0, Figure 7. The first diagonal element of the tridiagonal matrix T is $t_{00} = a_{00}$. The element t_{01} takes the value of $\pm S_1$ determined by Equation 9 (with r = 1) and sign opposite to that determined by Equation 11. The rest of the elements of the first row of T are, of course, zero. The vector u_1 is then readily constructed by Equations 8, 9, and 10 in a time which is negligible compared to the following calculations. The vectors p_1 and q_1 are then computed with almost full efficiency using Equations 13 and 14.

Once u_1, q_1, and the first row of the tri-diagonal matrix T are constructed, matrix A_2 is obtained using Equation 12. However, this transformation can be applied on the individual $\sqrt{P_E} \times \sqrt{P_E}$ submatrices A_{RC}, where R, C = (0, 1, . . . , $\sqrt{P_E} - 1$), thus dividing the vectors u_1 and q_1 into $\sqrt{P_E}$ subvectors each of $\sqrt{P_E}$ components. The submatrices A_{RC} at any step r are given by,

$$(A_{RC})_r = (A_{RC})_{r-1} - q_r^{(R)} u_r^{(C)^t} - u_r^{(R)} q_r^{(C)^t} \qquad (18)$$

for all R and C such that $R \leq C$.

It may be noted from Figure 7 that for the steps r = (1, 2, . . . , $\sqrt{P_E}$), the efficiency of parallel computation on the submatrices A_{00}, A_{01}, A_{02}, and A_{03}, Figure 8, descreases as r increases, while that of the remaining off-diagonal submatrices A_{12}, A_{13}, and A_{23} is 100%. Moreover, efficiencies of higher than 50% can be obtained for the diagonal submatrices A_{11}, A_{22}, and A_{33}. The entire time required for reducing a 64 x 64 symmetric matrix to the tri-diagonal form on a 64-PE machine is less than 4 ms.

4.3 Efficiency of the Tridiagonalization

The efficiency of the Householder processs on a symmetric P_E x P_E matrix is computed using the storage allocation scheme of Figure 7. Assume that computations are performed on $\sqrt{P_E}$ x $\sqrt{P_E}$ blocks and that no attempt is made to mask inefficiencies incurred due to the annihilation of rows and columns within particular $\sqrt{P_E}$ x $\sqrt{P_E}$ blocks. It is assumed that diagonal blocks are transformed in pairs and that no inefficiency is incurred when there is an even number of such blocks. In Figure 7 it may be noted that for example, the upper left and lower right triangular blocks may be accessed and transformed simultaneously, i.e., the elements with subscripts (0,0; 0,1; 0,2; 0,3; 15,15; 1,1; 1,2; 1,3; 14,15; 14,14; 2,2; 2,3; 13,15; 13,14; 13,13; 3,3). This pairing can be observed along the entire diagonal. It is also possible to overlap parts of square blocks which have been annihilated. For example, when $\sqrt{P_E}/2$ rows in a strip of block are set to zero, adjacent blocks could be paired and 100% efficiency regained. Similarly after $\sqrt{P_E}/4$, $\sqrt{P_E}/8$, . . . , etc. rows are annihilated, full efficiency could be regained. This latter matter requires some delicate discussion as well as programming which will not be included here.

The number of operations required to carry out the equations of Householder's process on a triangular matrix of decreasing dimension is

$$\sum_{j=1}^{P_E} \sum_{i=1}^{j} i = \frac{1}{2} \sum_{j=1}^{P_E} (j^2 + j) = \frac{P_E^3 + 3P_E^2 + 2P_E}{6} .$$

FIGURE 8

		BLOCK 0				1				2				3			
		PE 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	LOC 0	0,0	0,1	0,2	0,3	1,0	1,1	1,2	1,3	2,0	2,1	2,2	2,3	3,0	3,1	3,2	3,3
STRIP 0	1	3,4	3,5	3,6	3,7	0,4	0,5	0,6	0,7	1,4	1,5	1,6	1,7	2,4	2,5	2,6	2,7
	2	2,8	2,9	2,10	2,11	3,8	3,9	3,10	3,11	0,8	0,9	0,10	0,11	1,8	1,9	1,10	1,11
	3	1,12	1,13	1,14	1,15	2,12	2,13	2,14	2,15	3,12	3,13	3,14	3,15	0,12	0,13	0,14	0,15
	4	7,3	4,0	4,1	4,2	4,3	5,0	5,1	5,2	5,3	6,0	6,1	6,2	6,3	7,0	7,1	7,2
STRIP 1	5	6,7	7,4	7,5	7,6	7,7	4,4	4,5	4,6	4,7	5,4	5,5	5,6	5,7	6,4	6,5	6,6
	6	5,11	6,8	6,9	6,10	6,11	7,8	7,9	7,10	7,11	4,8	4,9	4,10	4,11	5,8	5,9	5,10
	7	4,15	5,12	5,13	5,14	5,15	6,12	6,13	6,14	6,15	7,12	7,13	7,14	7,15	4,12	4,13	4,14
	8	11,2	11,3	8,0	8,1	8,2	8,3	9,0	9,1	9,2	9,3	10,0	10,1	10,2	10,3	11,0	11,1
STRIP 2	9	10,6	10,7	11,4	11,5	11,6	11,7	8,4	8,5	8,6	8,7	9,4	9,5	9,6	9,7	10,4	10,5
	10	9,10	9,11	10,8	10,9	10,10	10,11	11,8	11,9	11,10	11,11	8,8	8,9	8,10	8,11	9,8	9,9
	11	8,14	8,15	9,12	9,13	9,14	9,15	10,12	10,13	10,14	10,15	11,12	11,13	11,14	11,15	8,12	8,13
	12	15,1	15,2	15,3	12,0	12,1	12,2	12,3	13,0	13,1	13,2	13,3	14,0	14,1	14,2	14,3	15,0
STRIP 3	13	14,5	14,6	14,7	15,4	15,5	15,6	15,7	12,4	12,5	12,6	12,7	13,4	13,5	13,6	13,7	14,4
	14	13,9	13,10	13,11	14,8	14,9	14,10	14,11	15,8	15,9	15,10	15,11	12,8	12,9	12,10	12,11	13,8
	15	12,13	12,14	12,15	13,12	13,13	13,14	13,15	14,12	14,13	14,14	14,15	15,12	15,13	15,14	15,15	12,12

The number of operations performed by a parallel maching using the storage allocation scheme of Figure 7, with the above assumptions, follows. There are P_E elements per $\sqrt{P_E} \times \sqrt{P_E}$ block and each block is swept $\sqrt{P_E}$ times per pass. This product times the total number of $\sqrt{P_E} \times \sqrt{P_E}$ blocks to be transformed in all passes is

$$\sqrt{P_E}P_E\left[\sum_{j=1}^{\sqrt{P_E}-1}\sum_{i=1}^{j} + 2\sum_{i=1}^{\frac{1}{2}\sqrt{P_E}} i\right]$$

where the first term in the brackets counts the number of rectangular blocks, and the second term counts the number of paired triangular blocks on the diagonal. This is equal to

$$\frac{P_E^3 + 2P_E^2 + 1.5\, P_E^2\sqrt{P_E}}{6} .$$

Thus for the Householder process on symmetric matrices we have:

$$\text{processing efficiency} = \frac{\text{ops. required}}{\text{ops. used}} = \frac{P_E^2 + 3P_E + 2}{P_E^2 + 1.5\sqrt{P_E} \cdot P_E + 2P_E} .$$

If $n = P_E = 64$ this yields an efficiency of approximately 86%. With the introduction of the pairing of partially annihilated rectangular blocks mentioned above, efficiencies of well over 90% can be achieved.

As in our discussion of the Jacobi process, matrices of larger and smaller dimensions may be handled using the same storage scheme. Matrices of dimension smaller than the number of PE's are handled using

$\sqrt{P_E} \times \sqrt{P_E}$ blocks. Those of dimension larger than the machine size are partitioned into triangular and rectangular blocks. The rectangular blocks are stored using the memory map specified for rows $0 \leq i \leq \frac{P_E}{2} - 1$ for all rows, since there are no holes. In the case of either large or small matrices of dimension other than a multiple of $\sqrt{P_E}$ the above efficiency formula is a close approximation of the correct value.

4.4 Disk Considerations

Large matrices that do not fit the primary memory are partitioned into blocks of 64 x 64. The number of rows of blocks is,

$$NR = \left\lfloor \frac{N - 1}{64} \right\rfloor + 1$$

where $N > 64$ is the size of the matrix. The procedure is the same as above. At the r-th step, the first row of the remaining matrix is fetched from the disk and the vectors u_r, p_r, and q_r are constructed. The 64 x 64 blocks are then fetched from the disk one at a time and transformed as discussed above. The efficiency of the computation increases as the size of the matrix gets larger.

The elements of the resulting tridiagonal matrix are stored in the primary storage. Because of symmetry, the number of these elements is only (2N - 1). The time taken for the transformation of an off-diagonal 64 x 64 block, Equation 18, on a 64-PE machine is approximately 6 ms., assuming that the vectors u, p, and q are already available. Thus if requests for eight 64 x 64 partitions in the disk queuer are kept one rotation ahead, the rotational latency of the disk would be masked.

5. THE QR ALGORITHM

5.1 Theoretical Background

Throughout this discussion it is assumed that the matrix $A = [a_{ij}]$ $(i,j = 0,1,2,\ldots,n-1)$ under consideration is both real and non-singular. The QR transformation consists of forming a sequence of matrices A_k $(k = 1,2,\ldots)$ by the relation [5],

$$A_{k+1} = Q_k^t A_k Q_k \tag{17}$$

where Q_k is an orthogonal matrix such that

$$Q_k^t A_k = R_k \tag{18}$$

is an upper triangular matrix. In general, for large k the matrix A_k tends to a form in which all the eigenvalues are either isolated on the diagonal or are the eigenvalues of 2×2 diagonal submatrices.

The QR algorithm is always used after the original matrix is reduced to the upper Hessenberg form, because:

(1) The upper Hessenberg form is invariant under the QR transformation [5].

(2) The number of multiplications involved in a QR transformation is greatly reduced, n^2 after reduction vs. n^3 for the unreduced matrix, where n is the order of the matrix.

(3) The QR algorithm has strong convergence properties when applied to upper Hessenberg matrices [9].

There are several stable methods for reducing any square matrix to upper Hessenberg form [3,4]. The method of Householder, which is described in the previous section, may be used for that purpose. The reduction to upper Hessenberg form is therefore completed in (n - 2) steps, the r-th of which is given by,

$$A_r = A_{r-1} - v_r p_r^t - (q_r - \alpha_r u_r) v_r^t \qquad (r = 1,2,\ldots, n-2) \tag{19}$$

where A_o is the original matrix A and,

$$u_r^t = (0, \ldots, 0, a_{r,r-1} \pm S_r, a_{r+1,r-1}, \ldots, a_{n-1,r-1}) \tag{20}$$

$$S_r = \left(\sum_{i=r}^{n-1} a_{i,r-1}^2 \right)^{\frac{1}{2}} \tag{21}$$

$$p_r^t = u_r^t A_{r-1} \tag{22}$$

$$q_r = A_{r-1} u_r \tag{23}$$

$$v_r = \frac{u_r}{S_r(S_r \pm a_{r,r-1})} \tag{24}$$

$$\alpha_r = p_r^t v_r \ . \tag{25}$$

The sign of $(a_{r,r-1} \pm S_r)$ is chosen as indicated in Equation 11. The total number of multiplications involved in this reduction is essentially $\frac{5}{3} n^3$.

The QR algorithm is invariably used with origin shifts [3,5,6,10] described by,

$$Q_k^t (A_k - \zeta_k I) = R_k, \ A_{k+1} = Q_k^t A_k Q_k \qquad (k = 1,2,\ldots) \tag{26}$$

where Q_k is orthogonal, R_k is upper triangular, ζ_k is the origin shift, and A_1 is the upper Hessenberg form of the original matrix A. In order to speed up convergence, the origin shift ζ_k is chosen such that it ultimately approaches an eigenvalue of A. It often happens that the matrix A has complex conjugate eigenvalues which lead to a complex value of ζ_k. Francis [5] proposed the method of double origin shift which combines two QR iterations in one, avoiding complex arithmetic. This method may be described by the relations,

$$W_k^t (A_k - \zeta_k I)(A_k - \zeta_{k+1} I) = U_k \tag{27}$$

$$A_{k+2} = W_k^t A_k W_k \tag{28}$$

where $W_k = Q_k Q_{k+1}$ is orthogonal, ζ_k and ζ_{k+1} are the origin shifts, and $U_k = R_{k+1} R_k$ is upper triangular. The origin shifts ζ_k and ζ_{k+1} are usually taken as the eigenvalues of the last 2×2 diagonal submatrix, and since A is real they are either both real or complex conjugate. Hence the matrix $B_k = (A_k - \zeta_k I)(A_k - \zeta_{k+1} I)$ is real and no complex arithmetic is involved. However, since A_k is nonsingular, A_{k+2} and W_k are determined by the first column of W_k only, and the determination is unique if the subdiagonal elements of A_{k+2} are positive. The first column of W_k is the same as the first column of the elementary Hermitian orthogonal matrix N_o that eliminates the elements of the first column of the matrix B_k below the diagonal. The first column of B_k has only three non-zero elements. They are given by,

$$\begin{aligned} b_{00}^{(k)} &= a_{00}^{(k)}(a_{00}^{(k)} - \epsilon_k) + a_{01}^{(k)} a_{10}^{(k)} + \eta_k \\ b_{10}^{(k)} &= a_{10}^{(k)}(a_{00}^{(k)} + a_{11}^{(k)} - \epsilon_k) \\ b_{20}^{(k)} &= a_{10}^{(k)} a_{21}^{(k)} \end{aligned} \tag{29}$$

where,

$$\begin{aligned} \epsilon_k &= \zeta_k + \zeta_{k+1} = a_{n-2,n-2}^{(k)} + a_{n-1,n-1}^{(k)} \\ \eta_k &= \zeta_k \zeta_{k+1} = a_{n-2,n-2}^{(k)} a_{n-1,n-1}^{(k)} - a_{n-2,n-1}^{(k)} a_{n-1,n-2}^{(k)} \end{aligned} \tag{30}$$

Thus the matrix N_o is given by,

$$N_o = I - \frac{2 y_o y_o^t}{\|y_o\|_2^2} \tag{31}$$

where,

$$y_o^t = (1,\ s_o,\ t_o,\ 0,\ldots,\ 0)$$

$$s_o = \frac{b_{10}^{(k)}}{b_{00}^{(k)} \pm S_o} \qquad (32)$$

$$t_o = \frac{b_{20}^{(k)}}{b_{00}^{(k)} \pm S_o}$$

$$S_o = \left[(b_{00}^{(k)})^2 + (b_{10}^{(k)})^2 + (b_{20}^{(k)})^2\right]^{\frac{1}{2}} .$$

Therefore, the matrix $C_1 = N_o^t A_k N_o$ is of the form,

$$\begin{bmatrix} \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} \\ \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} \\ \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} \\ \underline{X} & \underline{X} & \underline{X} & X & X & X & X \\ & & & X & X & X & X \\ & & & & X & X & X \\ & & & & & X & X \end{bmatrix}$$

where the underlined elements are those elements of A_k affected by the transformation. Now the matrix C_1 can be reduced to the upper Hessenberg from A_{k+2} by the orthogonal transformations,

$$A_{k+2} = N_{n-2}^t N_{n-3}^t \cdots N_1^t N_o^t A_k N_o N_1 \cdots N_{n-3} N_{n-2} \qquad (33)$$

where the elementary Hermitian matrix N_1 is chosen to eliminate the elements of the first column of C_1 below the subdiagonal. Therefore, $C_2 = N_1^t C_1 N_1$ will be of the form

$$\begin{bmatrix} X & \underline{X} & \underline{X} & \underline{X} & X & X & X \\ \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} \\ 0 & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} \\ 0 & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} & \underline{X} \\ & \underline{X} & \underline{X} & \underline{X} & X & X & X \\ & & & & X & X & X \\ & & & & & X & X \end{bmatrix} .$$

Consequently, N_2 is chosen to eliminate the elements of the second column of C_2 below the subdiagonal, and so forth. In general, the orthogonal matrices $N_r (r = 1,2,\ldots,n-2)$ are given by,

$$N_r = I - \frac{2y_r y_r^t}{||y_r||_2^2} \tag{34}$$

where,

$$y_r^t = (0,\ldots,0,1,s_r,t_r,0,\ldots,0) \tag{35}$$

in which the first r elements are zero,

$$s_r = \frac{c_{r+1,r-1}^{(r)}}{c_{r,r-1}^{(r)} \pm S_r} \tag{36}$$

$$t_r = \frac{c_{r+2,r-1}^{(r)}}{c_{r,r-1}^{(r)} \pm S_r} \tag{37}$$

$$S_r = \left[(c_{r,r-1}^{(r)})^2 + (c_{r+1,r-1}^{(r)})^2 + (c_{r+2,r-1}^{(r)})^2 \right]^{\frac{1}{2}} . \tag{38}$$

After each iteration the subdiagonal elements are examined for possible partitioning into two or more smaller upper Hessenberg matrices and the iterations continued with the bottom submatrix. Wilkinson [3] and Martin [6] mention that, using the double origin shift method, the average

number of iterations per eigenvalue for most matrices that are encountered in practice is less than two.

5.2 Implementation

Figure 8 shows the storage allocation scheme for a 16 x 16 non-symmetric matrix on a 16 PE machine. This storage scheme maps an element a_{ij} into the memory as follows:

$$a_{ij} \rightarrow \text{loc}\sqrt{P_E}\left\lfloor \frac{i}{\sqrt{P_E}} \right\rfloor + \left\lfloor \frac{j}{\sqrt{P_E}} \right\rfloor, \quad PE((i(\text{mod}\sqrt{P_E})) \times \sqrt{P_E} + \left\lfloor \frac{i}{\sqrt{P_E}} \right\rfloor + j) \quad (\text{mod } P_E) \qquad (39)$$

where P_E is the number of PE's in the machine and $\lfloor x \rfloor$ is the greatest integer less than x. The implementation of the reduction of the original matrix to the upper-Hessenberg form and the QR-iterations is similar to that of the Householder tridiagonalization explained in Section 4.2, and hence will not be discussed here in detail.

5.3 Efficiency

As a direct extension of the discussions in Sections 4.2 and 4.3 it can be shown that using Equations 19 - 25 the reduction to the upper-Hessenberg of a 64 x 64 matrix would take less than 10 ms on a 64-PE machine with an efficiency higher than 86%. It will be shown that the QR transformations dominate the overall efficiency calculations. Detailed examination of the efficiency of one step in double QR-iteration $C_{r+1} = N_r C_r N_r$ follows, where N_r is defined by Equations 34 - 38. By virtue of Equations 19 - 25, C_{r+1} may be written as,

$$C_{r+1} = C_r - v_r p_r^t - (q_r - \alpha_r y_r) v_r^t \qquad (40)$$

where $p_r^t = y_r^t C_r$, $q_r = C_r y_r$, $v_r = \frac{2}{||y_r||^2} y_r$, $\alpha_r = p_r^t v_r$, and y_r is defined by Equations 35 - 38. On a parallel computer the efficiency of the process used for finding the square roots affects the overall efficiency of the QR-algorithm. Therefore, in order to minimize the inefficiencies, we describe the following square root routine [11]. Any positive floating number Z may be expressed as $Z = 2^{2m}(x)$ where either $\frac{1}{2} \leq x < 1$ or $\frac{1}{4} \leq x < \frac{1}{2}$. Thus $\sqrt{Z} = 2^m(\sqrt{x})$ and the problem is reduced to finding the square root of x. We use the recursive relation,

$$\omega_{\ell+1} = \tfrac{1}{2}\omega_\ell \, (3 - x\, \omega_\ell^2) \qquad \ell = 0, 1, 2, \ldots \tag{41}$$

where,

$$\omega_0 = a + bx + cx^2 \tag{42}$$

in which,

$$\begin{array}{l} a = 3.16214 \\ b = -5.86118 \\ c = 4.75000 \end{array} \qquad \text{for } \tfrac{1}{4} \leq x < \tfrac{1}{2} \tag{43}$$

and,

$$\begin{array}{l} a = 2.22152 \\ b = -2.03146 \\ c = 0.81250 \end{array} \qquad \text{for } \tfrac{1}{2} \leq x < 1 \tag{44}$$

ω_ℓ converges quadratically to $\frac{1}{\sqrt{x}}$ and $\sqrt{x} = (x) \lim_{\ell \to \infty} \omega_\ell$.

Three iterations prove sufficient for evaluating the square root of x to 48 bits of accuracy. Assuming this accuracy satisfactory, $\sqrt{x}$ is given by,

$$\sqrt{x} = \frac{9}{4}\beta x - \frac{3}{4}\beta^3 x^2 - \frac{1}{2}x^2\gamma^3 \tag{45}$$

where,

$$\beta = \frac{3}{2}\omega_0 - \frac{1}{2}x\omega_0^3$$

$$\gamma = \frac{3}{2}\beta - \frac{1}{2}x\beta^3 \tag{46}$$

and ω_0 is given by Equations 42 - 44.

Figure 9 shows how the arithemtic operations would be performed on a parallel computer for finding the square root,

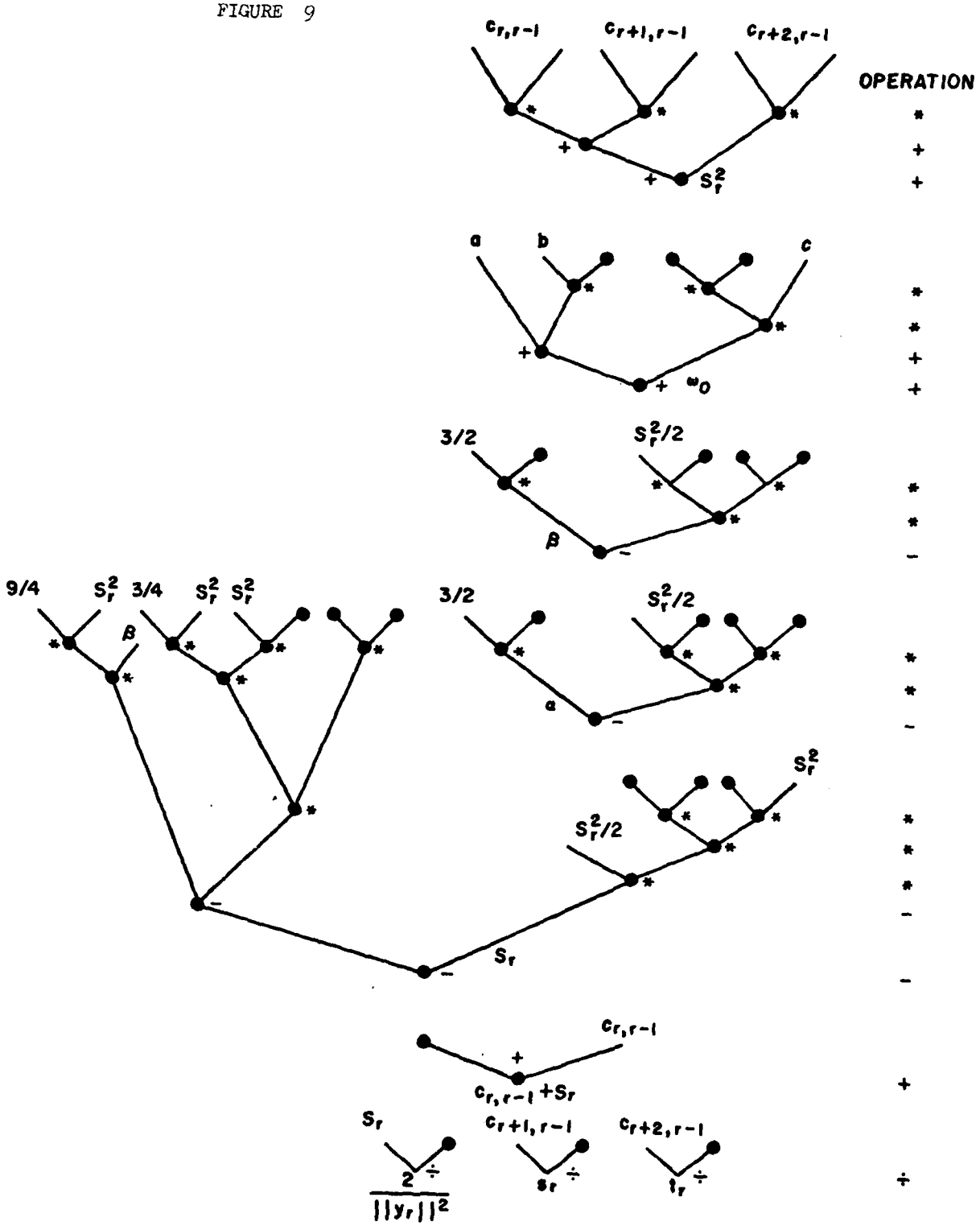
$$S_r = \left[(c^{(r)}_{r,r-1})^2 + (c^{(r)}_{r+1,\ r-1})^2 + (c^{(r)}_{r+2,\ r-1})^2\right]^{\frac{1}{2}},$$

the elements of the vector y_r,

$$s_r = \frac{c^{(r)}_{r+1,\ r-1}}{c^{(r)}_{r,\ r-1} \pm S_r}, \quad t_r = \frac{c^{(r)}_{r+2,\ r-1}}{c^{(r)}_{r,\ r-1} \pm S_r}, \quad \text{and the ratio}$$

$$\frac{2}{||y_r||^2} = \frac{c^{(r)}_{r,\ r-1} + S_r}{S_r}.$$

FIGURE 9


$c_{r,r-1}$
$c_{r+1,r-1}$
$c_{r+2,r-1}$
S_r^2
a
b
c
ω_0
3/2
$S_r^2/2$
β
9/4
S_r^2
3/4
S_r^2
S_r^2
β
3/2
$S_r^2/2$
a
S_r^2
$S_r^2/2$
S_r
$c_{r,r-1}$
$c_{r,r-1}+S_r$
S_r
2
$||y_r||^2$
$c_{r+1,r-1}$
s_r
$c_{r+2,r-1}$
t_r
OPERATION
10 *
9 +
1 ÷

In Figure 9 the operations at a given level in the tree are identical and shown to the right, and the totals are shown below. Unlabeled nodes are assumed to use the result computed immediately above. In this and subsequent discussions, the insignificant routing and broadcasting times are neglected. Assuming that one division is equivalent to five multiplications, and two additions are equivalent to one multiplication, the evaluation of the above quantities on a serial machine would effectively require 35 multiplications. On a $P_E \geq 8$ machine the same process requires only the equivalent of 20 multiplications or 10 μ.s. This is clear from the fact that the tree height is equivalent to 20 multiplications and its width does not exceed 8. The efficiency of the parallel computation of this part is $\frac{35/P_E}{20}$. This is a rather conservative definition of efficiency. It assumes that the total time "necessary" for a parallel machine to evaluate a function is just the equivalent number of multiplications divided by the number of PE's available.

The row vector $p_r^t = y_r^t C_r$ has its first (r - 1) elements equal to zero. Its formation requires the equivalent of 3(n - r+1) multiplications. The column vector $q_r = C_r y_r$ has (n - r - 4) zero last elements for r = 1, 2, . . . , n - 5. Its construction requires the equivalent of (3r + 10) multiplications. Assuming that the size of the matrix is $n = (m)P_E$ where the integer $m \geq 1$, the average number of equivalent multiplications required for computing either p_r or q_r on a parallel computer is given by $\frac{1}{m} \sum_{j=1}^{m} 3j = \frac{3}{2}(m + 1)$. In order to calculate the overall efficiency, observe that for the matrix C_r there are m diagonal $P_E \times P_E$ submatrices; for each, the corresponding portion of either p_r or q_r requires three equivalent multiplications with an average efficiency of 50%, and for each of the $\frac{1}{2} m(m - 1)$ $P_E \times P_E$ submatrices above the diagonal, the corresponding portion of either p_r or q_r also requires three equivalent multiplications, but with an efficiency of 100%. Weighing the individual efficiencies with the corresponding number of equivalent multiplications, the overall average efficiency for computing either p_r or q_r is calculated as follows:

$$\text{Eff}_{p_r,\ q_r} = \frac{(3m)\ 0.5 + \frac{3}{2} m(m - 1)\ (1.0)}{3m + \frac{3}{2} m(m - 1)} = \frac{m}{m + 1}.$$

Since p_r and q_r contain some zero elements and since the computation of $v_r = \frac{2}{||y_r||^2}\ y_r$ requires only two multiplications and that of $\alpha_r = p_r^t\ v_r$ requires four multiplications, both v_r and α_r can be computed simultaneously with p_r and q_r respectively. The computation of the column vector $\alpha_r y_r$ which requires two multiplications can also be performed simultaneously with the computation of the matrix $v_r\ p_r^t$. The average number of multiplications required for computing $v_r\ p_r^t$ on a parallel machine is the same as that of p_r and q_r and with the same efficiency. The construction of the vector $(q_r - \alpha_r\ y_r)$ requires the equivalent of $\frac{3}{2}$ multiplications with an efficiency of $\left(\frac{3}{P_E}\right)$. The average number of multiplications required for forming the matrix $(q_r - \alpha_r\ y_r)\ v_r^t$ on a parallel computer is also identical to that of p_r or q_r and has the same efficiency. The same holds for constructing the sum $C_r - v_r\ p_r^t - (q_r - \alpha_r\ y_r)\ v_r^t$. Neglecting the time taken to find the origin shifts for every iteration, it can be concluded that the total time required for one double QR-iteration on a parallel computer of P_E PE's and for a matrix of size $n = (m)P_E$ is essentially

$$((m)P_E - 2) \quad [10.75 + 3.75\ (m + 1)]\ \mu.s.$$

with an efficiency of

$$\frac{0.5\ (35/P_E) + 3.75\ m + (2.25/P_E)}{10.75 + 3.75\ (m + 1)}\ .$$

Thus, the time required for one QR-iteration for a 256 x 256 matrix on a 16-PE machine (assuming that the matrix can be contained in the primary memory) is 19 ms and the efficiency of parallel computation is 82%. If $P_E = 64$, then the time taken is 7.5 ms with an efficiency of 52%. Clearly, the minimum efficiency of the QR-iteration occurs when $m = 1$. For example, for $n = P_E = 64$, the efficiency is 22%. In order to establish a rough lower bound on the overall efficiency of the QR algorithm, consider the case for which $m = 1$ and when the matrix gets deflated by two rows and two columns every two iterations. This efficiency is given by,

$$\text{Eff.} = \frac{1}{N} \sum_{j} f_j \qquad (48)$$

where,

$$j = 4, 6, 8, \ldots, n-2 \quad , \quad N = \left\lceil \frac{n-5}{2} \right\rceil \qquad \text{for n - even}$$

$$j = 3, 5, 7, \ldots, n-2 \quad , \quad N = \left\lceil \frac{n-4}{2} \right\rceil + 1 \qquad \text{for n - odd}$$

and,

$$f_j = \frac{1}{18.25n} \; (38.5 + 3.75\, j)$$

Thus, for $n = P_E = 64$, the QR algorithm efficiency is approximately 11.50%.

REFERENCES

1. Barnes, G. H., Brown, et. al. "The ILLIAC IV Computer," IEEE Transactions on Computers. C-17:8 (August, 1968), 746-757.

2. Kuck, D. "ILLIAC IV Software and Application Programming," IEEE Transactions on Computers. C-17:8 (August, 1968), 757-770.

3. Wilkinson, J. V. The Algebraic Eigenvalue Problem. Clarendon Press, Oxford, 1965.

4. Martin, R. S., et al. "Householder's Tridiagonalization of a Symmetric Matrix," Numerische Mathematik. Vol. 11 (1968), 181-195.

5. Francis, J. G. F. "The QR Transformation, Parts I and II," Computer Journal. Vol. 4 (1961-1962), 265-271 and 332-345.

6. Martin, R. S., et al. "The QR Algorithm for Real Hessenberg Matrices," Numerische Mathematik. Vol. 14 (1970), 219-231.

7. Wilkinson, J. H. "Calculation of the Eigenvalues of a Symmetric Tridiagonal Matrix by the Method of Bisection," Numerische Mathematik. Vol. 4 (1962), 362-367.

8. Wilkinson, J. H. "Calculation of the Eigenvectors of a Symmetric Tridiagonal Matrix by Inverse Iteration," Numerische Mathematik. Vol. 4 (1962), 368-376.

9. Parlett, B. "Global Convergence of the Basic QR Algorithm on Hessenberg Matrices," Mathematics of Computation. Vol. 22 (1968), 803-817.

10. Wilkinson, J. H. "Global Convergence of Tridiagonal QR Algorithm with Origin Shifts," Linear Algebra and its Applications. Vol. 1 (1968), 409-420.

11. Katz, M. L. New and Revised Mathematical Subroutines for ILLIAC IV. ILLIAC IV Document No. 165, ILLIAC IV Project, University of Illinois, Urbana, Illinois. January 27, 1969.

ACKNOWLEDGEMENT

We are indebted to Professor David Young for helpful discussions and to Mr. James Madden for his comments and computational experiments. Sincere thanks go to Mrs. K. Flessner for her patience and accuracy in typing this report.