A SYSTOLIC ARCHITECTURE FOR ALMOST LINEAR-TIME SOLUTION OF THE SYMMETRIC EIGENVALUE PROBLEM

Richard P. Brent*
Franklin T. Luk**

TR 82-525
August 1982

Department of Computer Science
Upson Hall
Cornell University
Ithaca, New York 14853

*Department of Computer Science, Australian National University, Canberra A.C.T. 2600 Australia

**Supported in part by the U.S. Army Research Office under grant DAAG 29-79-C0124 and in part by the Mathematical Sciences Research Centre and the Centre for Mathematical Analysis, Australian National University

A Systolic Architecture for Almost Linear-Time Solution of the Symmetric Eigenvalue Problem

Richard P. Brent
Department of Computer Science
Australian National University
Canberra A.C.T. 2600 Australia

Franklin T. Luk*
Department of Computer Science
Cornell University
Ithaca N.Y. 14853 U.S.A.

TR 82-525
August, 1982

* Supported in part by U.S. Army Research Office under grant DAAG 29-79-C0124 and in part by the Mathematical Sciences Research Centre and the Centre for Mathematical Analysis, Australian National University

Abstract

An algorithm is presented for computing the eigenvalues and eigenvectors of an $n \times n$ real symmetric matrix. The algorithm is essentially a Jacobi method implemented on a two-dimensional systolic array of $O(n^2)$ processors with nearest-neighbour communication between processors. The speedup over the serial Jacobi method is $\theta(n^2)$, so the algorithm converges to working accuracy in time $O(nS)$, where S is the number of sweeps (typically $S \leq 10$).

Key Words and Phrases

Eigenvalue decomposition, real symmetric matrices, Hermitian matrices, Jacobi method, linear-time computation, systolic arrays, VLSI, real-time computation.

Short title

A systolic architecture for the eigenvalue problem.

1. Introduction

Recently, several researchers have investigated the problem of computing the eigenvalues of an $n \times n$ real symmetric matrix on a systolic array. Schreiber [11] suggests that one first reduce the given matrix to a matrix of bandwidth w and then apply the QR method. If the QR method takes I iterations for convergence, then Schreiber's approach requires $O(nI + n^2/w)$ units of time and $O(wn)$ processors. (I is $\Theta(n)$ if all the eigenvalues are to be computed.) Heller and Ipsen [7] consider only band matrices. They reduce the given matrix to a tridiagonal form and then apply the QR method with the standard shift. For a matrix of bandwidth w, their procedure requires $O(w)$ processors and $O(wn^2)$ time. Bojanczyk et al. [1] outline how one may implement an iteration of the QR method using $O(n^2)$ processors and $O(n)$ units of time, but without discussing the choice of shifts.

In this paper we show how one may implement a Jacobi method on a two-dimensional systolic array to compute all the eigenvalues and the corresponding eigenvectors. Our architecture uses $O(n^2)$ processors and only $O(n \log n)$ units of time. This is within a factor $O(\log n)$ of that required for the solution of n linear equations in n unknowns on a systolic array [1,2], and improves on the earlier results quoted above.

Preliminary results are given in Section 2, and the basic idea of our algorithm is outlined in Section 3. Details are filled in and some variations and extensions of the basic algorithm are given in Sections 4 and 5. The results of some numerical simulations are given in the Appendix.

2. Serial Jacobi method

We may describe the serial Jacobi method as follows. Denote the original symmetric matrix by A_1 and generate a sequence of symmetric matrices $\{A_k\}$ by the relation

$$A_{k+1} = R_k^T A_k R_k , \tag{2.1}$$

where R_k is a plane rotation. Let $R_k \equiv (r_{pq}^{(k)})$ and $A_k \equiv (a_{pq}^{(k)})$, and suppose that R_k represents a rotation in the (i,j) plane, with $i < j$. We have

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

The rotation angle θ is chosen so as to reduce the (i,j) element of A_k to zero. If the element is already zero, no rotation will take place, i.e. $\theta = 0$. If $a_{ij}^{(k)} \neq 0$, then we use the formulas given by Rutishauser [10] to compute $\sin\theta$ and $\cos\theta$:

$$\begin{aligned} \xi &= \frac{a_{jj}^{(k)} - a_{ii}^{(k)}}{2a_{ij}^{(k)}} , \\ t &= \frac{\operatorname{sign}(\xi)}{|\xi| + \sqrt{1 + \xi^2}} = \tan\theta , \\ \cos\theta &= \frac{1}{\sqrt{1 + t^2}} , \text{ and} \\ \sin\theta &= t.\cos\theta . \end{aligned} \tag{2.3}$$

Note that the rotation angle θ may always be chosen to satisfy

$$|\theta| \leq \frac{\pi}{4} . \tag{2.4}$$

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

$$
\begin{aligned}
a_{ii}^{(k+1)} &= a_{ii}^{(k)} - t.a_{ij}^{(k)} , \\
a_{jj}^{(k+1)} &= a_{jj}^{(k)} + t.a_{ij}^{(k)} , \\
a_{ij}^{(k+1)} &= a_{ji}^{(k+1)} = 0 , \\
a_{iq}^{(k+1)} &= a_{qi}^{(k+1)} = \cos\theta.a_{iq}^{(k)} - \sin\theta.a_{jq}^{(k)} \\
a_{jq}^{(k+1)} &= a_{qj}^{(k+1)} = \sin\theta.a_{iq}^{(k)} + \cos\theta.a_{jq}^{(k)}
\end{aligned}
\Bigg\} \ (q \neq i,j) . \tag{2.5}
$$

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

$$(1,2),(1,3),\ldots,(1,n),(2,3),\ldots,(2,n),(3,4),\ldots,(n-1,n) . \tag{2.6}$$

Forsythe and Henrici [5] prove that, subject to (2.4), the cyclic-by-rows Jacobi method always converges.

Unfortunately, the cyclic-by-rows scheme is apparently not amenable to parallel processing. In Section 3 we present an ordering that enables us to do $\lfloor \frac{n}{2} \rfloor$ rotations simultaneously. The (theoretical) price we pay is the loss of guaranteed convergence. Hansen [6] discusses the convergence properties associated with various orderings for the serial Jacobi method. He defines a certain "preference factor" for comparing different ordering schemes. Our new ordering is in fact quite desirable, for it asymptotically optimizes the preference factor as $n \to \infty$. Thus, although the convergence proof of [5] does not apply, we expect convergence in practice to be faster than for the cyclic-by-rows ordering. Simulation results (presented in the Appendix) support this conclusion.

To ensure convergence, we may adopt a threshold approach [13, pp. 277-278]: associate with each sweep a threshold value, and when making the transformations of that sweep, omit any rotation if the doomed off-diagonal element is smaller in magnitude than the threshold. It is well-known that the serial Jacobi method enjoys ultimate quadratic convergence [12]; in practice it requires only six to ten sweeps for convergence to working accuracy (see [10] and the Appendix).

3. An idealized systolic architecture

In this section we describe an idealized systolic architecture for implementing the Jacobi method to compute the eigenvalues of a symmetric n by n real matrix A. The architecture is idealized in that it assumes the ability to broadcast row and column rotation parameters in constant time. In Section 5 we show how to avoid this assumption, after showing in Section 4 how to take advantage of symmetry, compute eigenvectors, etc.

Assume that n is even and that we have a square array of $n/2$ by $n/2$ processors, each processor containing a 2 by 2 submatrix of A. Initially processor P_{ij} contains $\begin{pmatrix} a_{2i-1,2j-1} & a_{2i-1,2j} \\ a_{2i,2j-1} & a_{2i,2j} \end{pmatrix}$ for $i,j=1,\ldots,n/2$, and P_{ij} is connected to its nearest neighbours $P_{i\pm1,j}$ and $P_{i,j\pm1}$ (see Figure 1). In general P_{ij} contains four real numbers $\begin{pmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{pmatrix}$, where $\alpha_{ij} = \alpha_{ji}$, $\delta_{ij} = \delta_{ji}$ and $\beta_{ij} = \gamma_{ji}$ by symmetry.

The diagonal processors P_{ii} $(i=1,\ldots,n/2)$ act differently from the off-diagonal processors P_{ij} $(i \neq j,\ 1 \leq i,j \leq n/2)$. Each time step the diagonal processors P_{ii} compute rotations $\begin{pmatrix} c_i & s_i \\ -s_i & c_i \end{pmatrix}$ to annihilate their off-diagonal elements β_{ii} and γ_{ii} (actually $\beta_{ii} = \gamma_{ii}$), i.e. so that $c_i^2 + s_i^2 = 1$ and

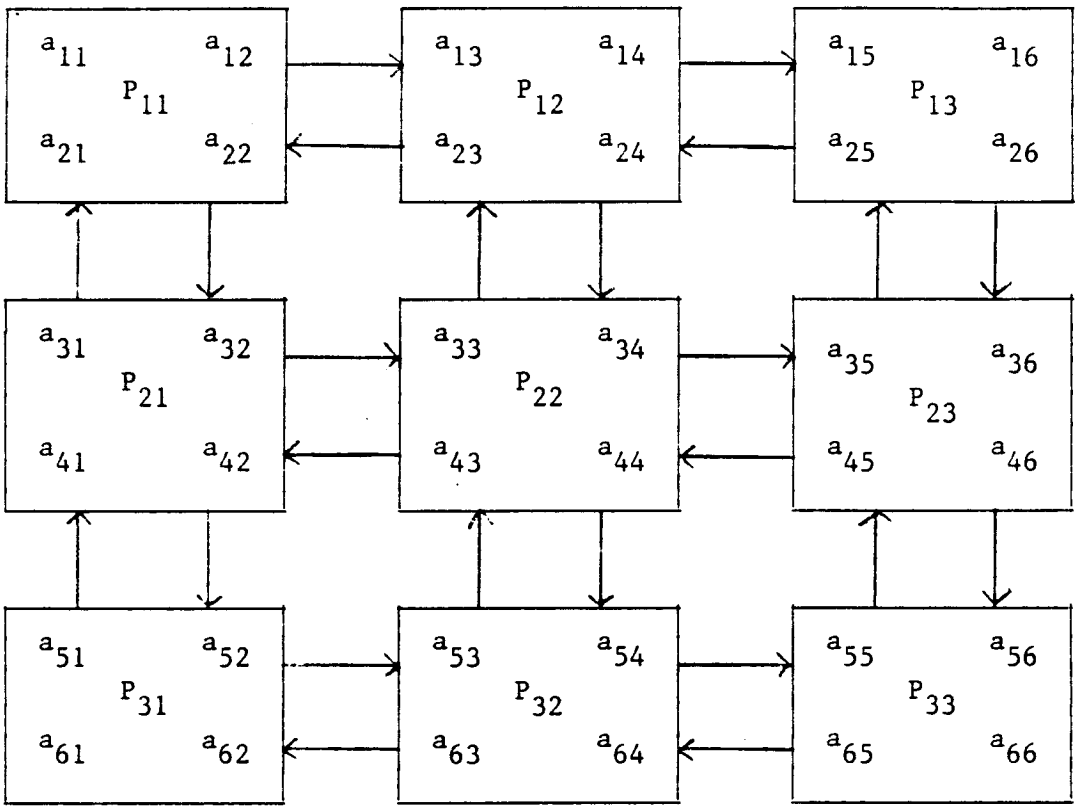


Figure 1: Initial configuration (idealized, n = 6)

$\begin{pmatrix} c_i & -s_i \\ s_i & c_i \end{pmatrix} \begin{pmatrix} \alpha_{ii} & \beta_{ii} \\ \gamma_{ii} & \delta_{ii} \end{pmatrix} \begin{pmatrix} c_i & s_i \\ -s_i & c_i \end{pmatrix} = \begin{pmatrix} \alpha'_{ii} & 0 \\ 0 & \delta'_{ii} \end{pmatrix}$ is diagonal. From (2.3) and (2.5) with a change of notation we find that

$$(3.1) \qquad \begin{pmatrix} c_i \\ s_i \end{pmatrix} = \frac{1}{\sqrt{1 + t_i^2}} \begin{pmatrix} 1 \\ t_i \end{pmatrix}$$

and

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

where

$$(3.2) \qquad t_i = \begin{cases} 0 & \text{if } \beta_{ii} = 0 \\ \dfrac{\operatorname{sign}(\xi_i)}{|\xi_i| + \sqrt{1 + \xi_i^2}} & \text{if } \beta_{ii} \neq 0 , \end{cases}$$

and

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

To complete the rotations which annihilate β_{ii} and γ_{ii}, $i = 1,\ldots,n/2$, the off-diagonal processors P_{ij} $(i \neq j)$ must perform the transformations $\begin{pmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{pmatrix} \leftarrow \begin{pmatrix} \alpha'_{ij} & \beta'_{ij} \\ \gamma'_{ij} & \delta'_{ij} \end{pmatrix}$, where

$\begin{pmatrix} \alpha'_{ij} & \beta'_{ij} \\ \gamma'_{ij} & \delta'_{ij} \end{pmatrix} = \begin{pmatrix} c_i & -s_i \\ s_i & c_i \end{pmatrix} \begin{pmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{pmatrix} \begin{pmatrix} c_j & s_j \\ -s_j & c_j \end{pmatrix}$. We assume that the diagonal processor P_{ii} broadcasts the rotation parameters c_i and s_i to processors P_{ij} and P_{ji} $(j = 1,\ldots,n/2)$ in constant time, so that the off-diagonal processor P_{ij} has access to the parameters c_i, s_i, c_j and s_j when required. (This assumption is removed in Section 5.)

To complete a step, columns (and corresponding rows) are interchanged between adjacent processors so that a new set of n off-diagonal elements is ready to be annihilated by the diagonal processors during the next time step. This is done in two sub-steps. First, adjacent columns are exchanged as in the SVD algorithm described in Sections 3-4 of [3], and illustrated in Figure 2, where each box denotes a column of $n/2$ processors.

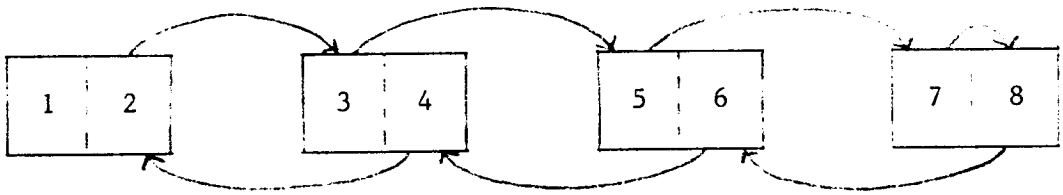


Figure 2: Column interchanges (n = 8)

Next, the same permutation is applied to rows, so as to maintain symmetry. Formally, we can specify the operations performed by a processor P_{ij} with outputs $outh\alpha_{ij},\ldots,outh\delta_{ij},outv\alpha_{ij},\ldots,outv\delta_{ij}$, and inputs $inh\alpha_{ij},\ldots,inv\delta_{ij}$ by Program 1. Note that outputs of one processor are connected to inputs of adjacent processors in the obvious way, e.g. $outh\beta_{ij}$ is connected to $inh\alpha_{i,j+1}$

```
{subscripts (i,j) omitted if no ambiguity results}
{column interchanges}
if i = 1 then [outhβ ← β; outhδ ← δ]
      else if i < n/2 then [outhβ ← α; outhδ ← γ];
if i > 1 then [outhα ← β; outhγ ← δ];
{wait for outputs to propagate to inputs of adjacent processors}
if i < n/2 then [β ← inhβ; δ ← inhδ]
           else [β ← α; δ ← γ];
if i > 1 then [α ← inhα; γ ← inhγ];
{row interchanges}
if j = 1 then [outvγ ← γ; outvδ ← δ]
         else if j < n/2 then [outvγ ← α; outvδ ← β];
if j > 1 then [outvα ← γ; outvβ ← δ];
{wait for outputs to propagate to inputs of adjacent processors}
if j < n/2 then [γ ← invγ; δ ← invδ]
           else [γ ← α; δ ← β];
if j > 1 then [α ← invα; β ← invβ];
```

Program 1: Column and row interchanges for idealized processor P_{ij}

($1 \leq i \leq n/2$, $1 \leq j < n/2$): see Figure 3. Note that, in Figure 3 and elsewhere, we have omitted subscripts (i,j) if no ambiguity arises, e.g. invα is used instead of $\text{inv}\alpha_{ij}$.

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

required for convergence (see Section 2), the total time required to diagonalize A to working accuracy is $O(n)$.

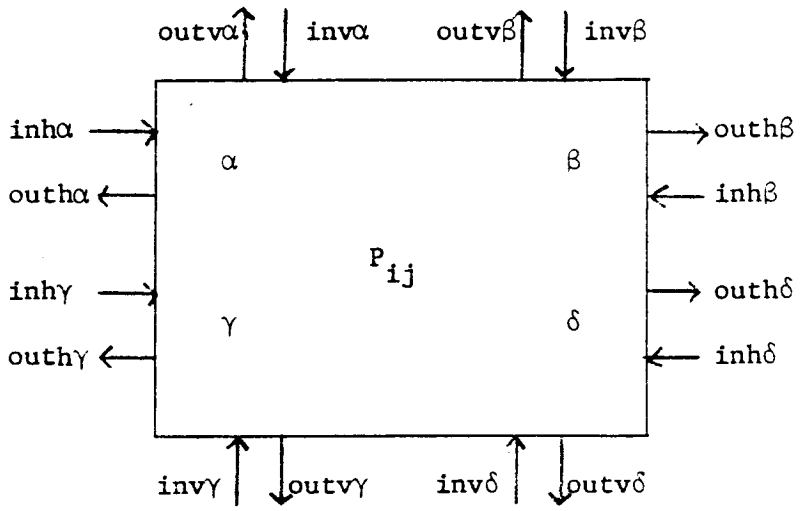


Figure 3: Input and output lines for idealized processor P_{ij} with nearest-neighbour connections

4. Further details

Several assumptions were made in Section 3 to simplify the exposition. In this section we show how to remove these assumptions (except for the broadcast of rotation parameters, discussed in Section 5) and we also suggest some practical optimizations.

4.1 Threshold strategy

It is clear that a diagonal processor P_{ii} might omit rotations if its off-diagonal elements $\beta_{ii} = \gamma_{ii}$ were sufficiently small. All that is required is to broadcast $\begin{pmatrix} c_i \\ s_i \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ along processor row and column i .

As discussed in Section 2, a suitable threshold strategy guarantees convergence, although we do not know any example for which our ordering fails to give convergence even without a threshold strategy.

4.2 Computation of eigenvectors

If eigenvectors are required, the matrix Q of eigenvectors can be accumulated at the same time as A is being diagonalized. Each systolic processor P_{ij} $(1 \leq i,j \leq n/2)$ needs four additional memory cells $\begin{pmatrix} \mu_{ij} & \nu_{ij} \\ \sigma_{ij} & \tau_{ij} \end{pmatrix}$, and during each step sets

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

Each processor transmits its $\begin{pmatrix} \mu & \nu \\ \sigma & \tau \end{pmatrix}$ values to adjacent processors in the same way as its $\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$ values (see Program 1). Initially $\mu_{ij} = \nu_{ij} = \sigma_{ij} = \tau_{ij} = 0$ if $i \neq j$, and $\mu_{ii} = \tau_{ii} = 1$, $\sigma_{ii} = \nu_{ii} = 0$. After a sufficiently large (integral) number of sweeps, we have Q defined to working accuracy by

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

4.3 Diagonal connections

In Program 1 we assumed that only horizontal and vertical nearest-neighbour connections were available. Except at the boundaries, diagonal connections are more convenient. This is illustrated in Figures 4 and 5 (with subscripts (i,j) omitted).

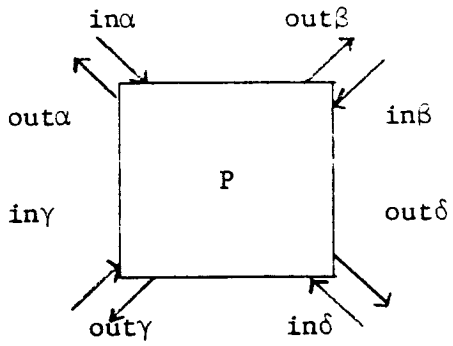


Figure 4: Diagonal input and output lines for processor P_{ij}

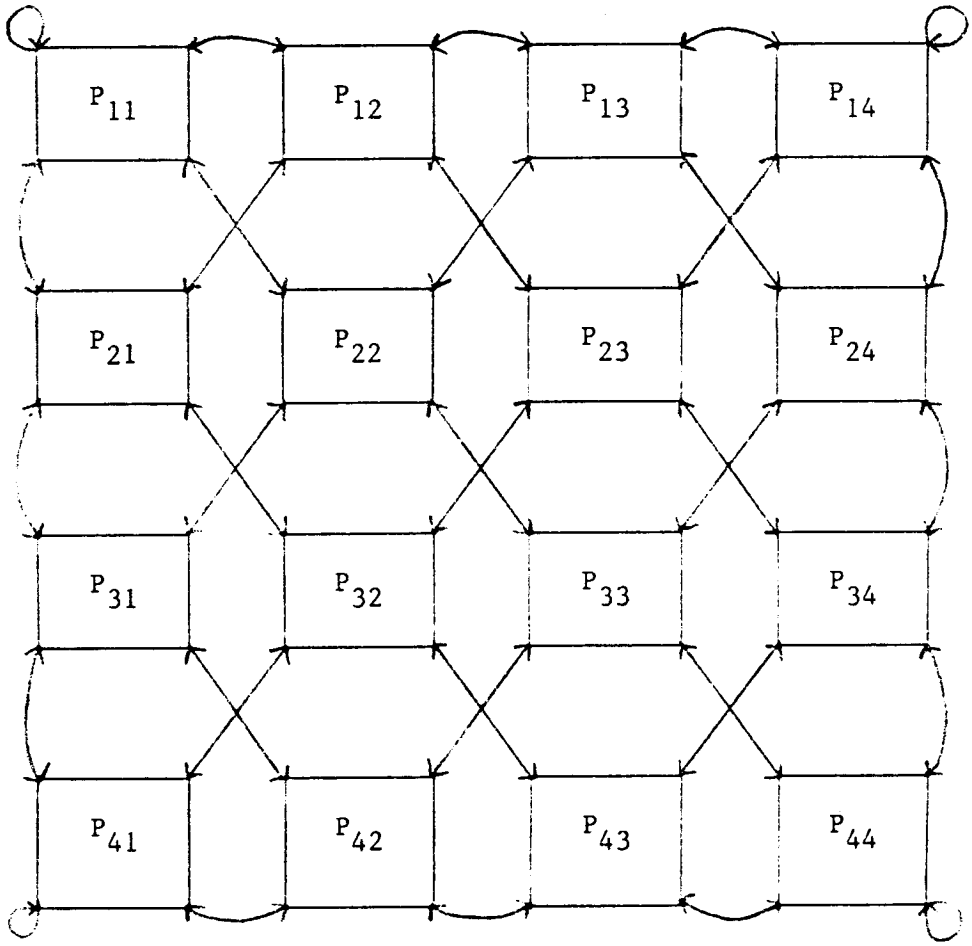


Figure 5: "Diagonal" connections, n = 8

(here and below ⟷ stands for ⇄)

Diagonal outputs and inputs are connected in the obvious way, as shown in Figure 5,

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

Program 2 is equivalent to Program 1 but assumes a diagonal connection pattern as illustrated in Figures 4 and 5. Subsequently we assume the diagonal connection pattern for convenience, although it can easily be simulated if only horizontal and vertical connections are available.

{subscripts (i,j) omitted for clarity}

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

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

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

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

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

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

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

Program 2: Diagonal interchanges for processor P_{ij}

4.4 Taking full advantage of symmetry

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

4.5 Odd n

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

4.6 Rotation parameters

In Section 3 we assumed that the diagonal processor P_{ii} would compute c_i and s_i according to (3.1), and then broadcast both c_i and s_i along processor row and column i. It may be preferable to broadcast only t_i (given by (3.2)) and let each off-diagonal processor P_{ij} compute c_i, s_i, c_j and s_j from t_i and t_j. Thus communication costs are reduced at the expense of requiring off-diagonal processors to compute two square roots per time step (but this may not be significant since the diagonal processors must compute one or two square roots per step in any case). In what follows a "rotation parameter" may mean either t_i or the pair (c_i, s_i).

5. Avoiding broadcast of rotation parameters

The most serious assumption of Section 3 is that rotation parameters computed by diagonal processors can be broadcast along rows and columns in constant time. We now show how to avoid this assumption, and merely transmit rotation parameters at constant speed between adjacent processors, while retaining total time $O(n)$ for the algorithm.

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

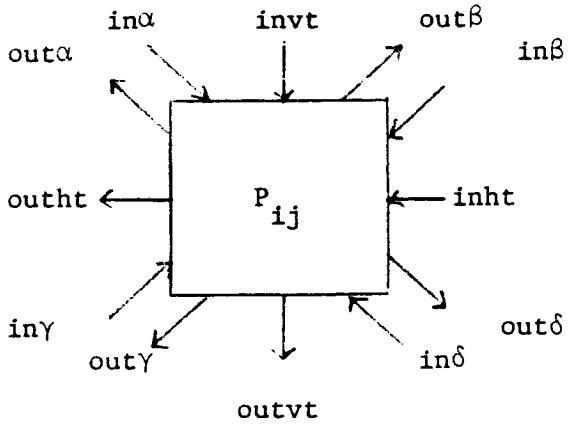
A processor cannot commence a rotation until data from earlier rotations is available on all its diagonal input lines. Thus, processor P_{ij} needs data from processors $P_{i-1,j-1}$, $P_{i-1,j+1}$, $P_{i+1,j-1}$ and $P_{i+1,j+1}$ if $1 < i < n/2$, $1 < j < n/2$ (for the other cases see Section 4.3). Since

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

it is sufficient for processor P_{ij} to be idle for two time steps while waiting for the processors $P_{i\pm1,j\pm1}$ to complete their (possibly delayed) steps. Thus, the price paid to avoid broadcasting rotation parameters is that each processor is active for only one third of the total computation time. A similar inefficiency occurs with many other systolic algorithms, see e.g. [1,2,8,9]. (The fraction one-third can be increased almost to unity if rotation parameters are propagated at greater than unit speed.)

A typical processor P_{ij} $(1 < j \leq i < n/2)$ has input and output lines as shown in Figure 6 (with subscripts (i,j) or (i,i) omitted). Figure 6 differs from Figure 4 in that it shows the horizontal and vertical lines inht, outht, invt, outvt for transmission of rotation parameters. Processors interconnect as shown in Figure 7.

Subdiagonal ($1 < j < i < n/2$)



Diagonal ($1 < i < n/2$)

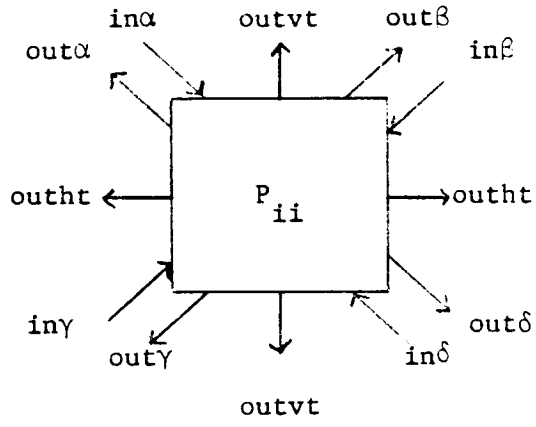


Figure 6: Input and output lines for typical subdiagonal and diagonal processors

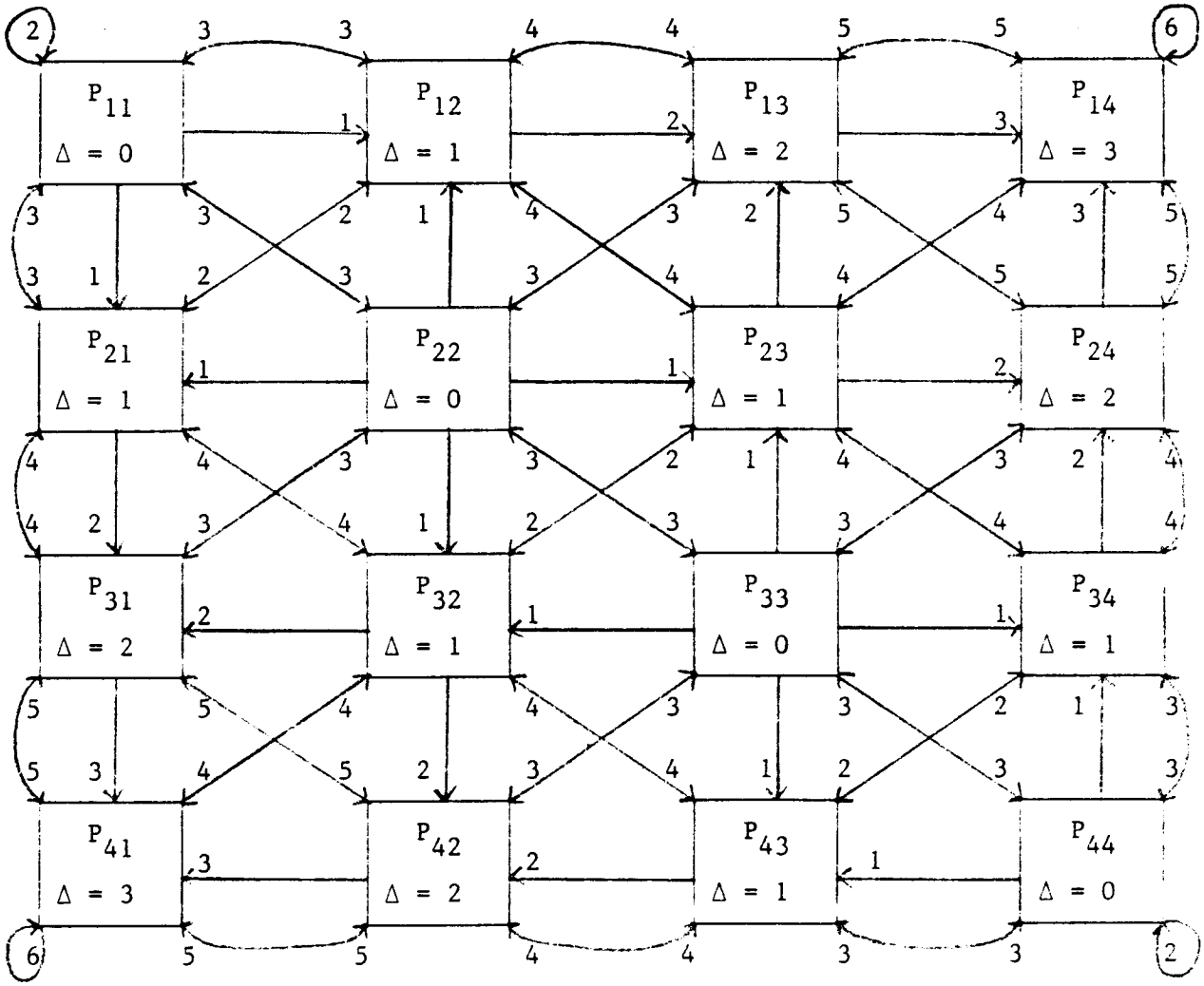


Figure 7: Interprocessor connections (n = 8)

(The first times at which inputs are available are indicated.)

Assuming that the array $(a_{ij})_{1 \le i,j \le n}$ is available in the systolic array at time $T = 0$, the operation of processor P_{ij} proceeds as described by Program 3. We assume that each time step has nonoverlapping read and write phases; the result of a write at step T should be available at the read phase of steps $T + 1$, $T + 2$ and $T + 3$ in a neighbouring processor, but should not interfere with a read at step T in a neighbouring processor. The first time steps at which data are available on various processors' input lines are indicated in Figure 7.

Program 3 does not compute eigenvectors, but may easily be modified to do so (as outlined in Section 4). We have also omitted a termination criterion. The simplest is to perform a fixed number S (say conservatively 10) sweeps; then processor P_{ij} halts when $T = 3S(n - 1) + \Delta_{ij} + 3$, since a sweep takes $3(n - 1)$ time steps. A more sophisticated criterion is to stop if no nontrivial rotations were performed during the previous sweep. This requires communication along the diagonal, which can be done in $n/2$ time steps.

The simulation results given in the Appendix indicate that $S = O(\log n)$ is sufficient to ensure convergence to within a fixed absolute error, so our algorithm takes time $O(n \log n)$.

```
if (T ≥ Δ) and (T - Δ ≡ 0 (mod 3)) then
      begin
      if T ≠ Δ then [α β; γ δ] ← [inα inβ; inγ inδ];
      if Δ = 0 then {diagonal processor}
            begin
            if β = 0 then ξ ← 0 else ξ ← (δ - α)/(2*β);
            if ξ = 0 then t ← 0 else t ← sign(ξ)/(|ξ| + √(1 + ξ²));
            t' ← t;
            α ← α - t*β; δ ← δ + t*β;
            β ← 0;        γ ← 0
            end
      else {off-diagonal processor}
            begin
            t ← inht; t' ← invt;
            c ← 1/√(1 + t²); c' ← 1/√(1 + t'²);
            s ← t*c;        s' ← t'*c';
            [α β; γ δ] ← [c -s; s c][α β; γ δ][c' s'; -s' c']
            end;
      outht ← t; outvt ← t';
      if i > j then set outβ as in Program 2;
      if i < j then set outγ as in Program 2
      end
else if (T ≥ Δ) and (T - Δ ≡ 1 (mod 3)) then
      begin
      if (i = 1) or (j = 1) then set outα as in Program 2;
      if (i = n/2) or (j = n/2) then set outδ as in Program 2
      end
else if (T ≥ Δ) and (T - Δ ≡ 2 (mod 3)) then
      begin
      if (i > 1) and (j > 1) then set outα as in Program 2;
      if i ≤ j then set outβ as in Program 2;
      if i ≥ j then set outγ as in Program 2;
      if (i < n/2) and (j < n/2) then set outδ as in Program 2
      end
else {do nothing this time step}.
```

Program 3: Program for one time step of processor P_{ij}

6. Conclusion

We have described how a square systolic array of $\lceil \frac{n}{2} \rceil$ by $\lceil \frac{n}{2} \rceil$ processors, each capable of performing floating-point operations (including square roots) and with a small amount of local storage, and having connections to nearest horizontal and vertical (and preferably also diagonal) neighbours, can compute the eigenvalues and eigenvectors of a real symmetric matrix in time $O(n \log n)$. The constant is sufficiently small that the method is competitive with the usual $O(n^3)$ serial algorithms, e.g. tridiagonalization followed by the QR iteration, for quite small n. The speedup should be significant for real-time computations with moderate or large n.

In [3] we present a related algorithm for computing the singular value decomposition on a systolic array. The problem of computing eigenvalues and eigenvectors of an unsymmetric real matrix on a systolic array is currently being investigated; unfortunately, the ideas used for symmetric matrices do not appear to carry over to Eberlein's methods [4] in an obvious way. However, everything that we have said concerning real symmetric matrices goes over with the obvious changes to complex Hermitian matrices.

References

[1] 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.*, to appear. Also available as Tech. Report TR-CS-81-01, Dept. of Computer Science, Aust. Nat. Univ., 1981.

[2] R.P. Brent and F.T. Luk, *Computing the Cholesky factorization using a systolic architecture*, Tech. Report TR-CS-82-08, Dept. of Computer Science, Aust. Nat. Univ., August 1982.

[3] R.P. Brent and F.T. Luk, *A systolic architecture for the singular value decomposition*, Tech. Report TR-CS-82-09, Dept. of Computer Science, Aust. Nat. Univ., August 1982.

[4] P.J. Eberlein and J. Boothroyd, "Solution to the eigenproblem by a norm reducing Jacobi type method", in [14], 327-338.

[5] G.E. Forsythe and P. Henrici, "The cyclic Jacobi method for computing the principal values of a complex matrix", *Trans Amer. Math. Soc.* 94 (1960), 1-23.

[6] E.R. Hansen, "On cyclic Jacobi methods", *J. Soc. Indust. Appl. Math.* 11 (1963), 448-459.

[7] D.E. Heller and I.C.F. Ipsen, "Systolic networks for orthogonal equivalence transformations and their applications", *Proc. 1982 Conf. on Advanced Research in VSLI*, MIT (1982), 113-122.

[8] H.T. Kung, "Why systolic architectures", *IEEE Computer* 15, 1 (1982), 37-46.

[9] H.T. Kung and C.E. Leiserson, "Algorithms for VLSI processor arrays", in *Introduction to VLSI Systems* (by C. Mead and L. Conway), Addison-Wesley, Reading, Massachusetts, 1980, 271-292.

[10] H. Rutishauser, "The Jacobi method for real symmetric matrices", in [14], 202-211.

[11] R. Schreiber, "Systolic arrays for eigenvalue computation", *Proc. SPIE Symp. East 1982, Vol. 341 (Real-Time Signal Processing V)*, to appear.

[12] J.H. Wilkinson, "Note on the quadratic convergence of the cyclic Jacobi process", *Numer. Math.* 4 (1962), 296-300.

[13] J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

[14] J.H. Wilkinson and C. Reinsch (editors), *Handbook for Automatic Computation, Vol. 2 (Linear Algebra)*, Springer-Verlag, Berlin, 1971.

Appendix: Simulation results

We have compared the ordering described in Section 3 with the cyclic-by-rows ordering (2.6) by applying the Jacobi method with each ordering to random n by n symmetric matrices (a_{ij}), where the elements a_{ij} for $1 \leq i \leq j \leq n$ were uniformly and independently distributed in $[-1,1]$. (Other distributions were also tried, and gave similar results.) The stopping criterion was that the sum $\sum_{i \neq j} a_{ij}^2$ of off-diagonal elements was reduced to 10^{-12} times its initial value. Table 1 gives the mean number of sweeps S_{row} or S_{new} for the cyclic-by-rows ordering and the ordering of Section 3, respectively, where a "sweep" is $n(n-1)/2$ rotations. The maximum number of sweeps required for each ordering is given in parentheses in the Table.

n	trials	S_{row}	S_{new}
4	5000	2.96 (4.17)	2.64 (4.00)
6	5000	3.63 (4.87)	3.37 (4.40)
8	2000	4.07 (5.04)	3.79 (4.75)
10	2000	4.39 (5.56)	4.09 (5.47)
20	1000	5.23 (5.93)	4.94 (5.81)
30	1000	5.67 (6.62)	5.41 (6.49)
40	1000	5.92 (6.76)	5.74 (6.54)
50	1000	6.17 (7.13)	5.99 (6.78)
100	175	6.81 (7.43)	6.76 (7.21)

Table 1: Simulation results for row and new orderings

From Table 1 we see that our new ordering is better than the cyclic-by-rows ordering, perhaps for the reason suggested in Section 2, although the difference between the two orderings becomes less marked as n increases. For both orderings, the number of sweeps S grows slowly with n. Empirically we find that $S = O(\log n)$, and there are theoretical reasons for believing this, although it has not been proved rigorously. In practice S can be regarded as

a constant (say 10) for all realistic values of n (say $n \leq 1000$): see [10]. More extensive simulation results for six different classes of orderings will be reported elsewhere.