JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 2, 261–276 (1985)

Tridiagonalization of a Symmetric Matrix on a Square Array of Mesh-Connected Processors

A. BOJAŃCZYK AND R. P. BRENT

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

Received October 22, 1984

A parallel algorithm for transforming an $n \times n$ symmetric matrix to tridiagonal form is described. The algorithm implements Givens rotations on a square array of $n \times n$ processors in such a way that the transformation can be performed in time $O(n \log n)$. The processors require only nearest-neighbor communication. The reduction to tridiagonal form could be the first step in the parallel solution of the symmetric eigenvalue problem in time $O(n \log n)$. © 1985 Academic Press, Inc.

1. INTRODUCTION

Reduction to tridiagonal form is a crucial step in the sequential QR algorithm [8] as it reduces the number of arithmetic operations required to perform one QR iteration from $O(n^3)$ for a full matrix to only $O(n)$ for a tridiagonal matrix. Also, the choice of shifts is critical for fast convergence of the QR algorithm. The unshifted QR algorithm may converge very slowly, and it is not clear how to choose shifts cheaply and effectively unless the matrix is tridiagonal.

For parallel computation the tridiagonal form is not so crucial in reducing the cost of one QR iteration, for it is known how to perform a QR iteration on a full matrix in time $O(n)$ on a two-dimensional array of n^2 processors [1]. However, this result is useless because of the lack of a good shift strategy. Thus, it is natural to ask if a square (or linear) array of processors can be utilized with reasonable efficiency in a parallel implementation of the QR algorithm. Several authors have investigated this problem. Schreiber [6] suggests that one first reduce the given matrix to a matrix of bandwidth w and then apply the QR algorithm. Schreiber shows that the reduction can be done in time $O(n^2/w)$ using $O(wn)$ processors, but he does not discuss the shift strategy. S. Y. Kung and Gal-Ezer [5] propose a linear array of $O(n)$ pro-

cessors to reduce a symmetric matrix to tridiagonal form in time $O(n^2)$ and then apply the QR algorithm with the standard shift. Heller and Ipsen [4] consider only band matrices. For a matrix of bandwidth w their procedure takes time $O(wn^2)$ using $O(w)$ processors.

In this paper we show how a symmetric matrix A can be reduced to tridiagonal form in time $O(n \log n)$, using a two-dimensional array of n^2 processors with nearest-neighbor communication. This is an improvement of order $n/\log n$ in speed over the results mentioned above, on the assumption that n^2 processors are available. The eigenvalues of the tridiagonal matrix can be found either by the QR algorithm or by the method of Sturm sequences and bisection [8]. The latter method is attractive for parallel computation because different processors can compute different eigenvalues independently.

The aim of this paper is to show that symmetric tridiagonalization in time $O(n \log n)$ on a systolic array is *possible*. We do not claim that the algorithm presented here is practical; rather we hope that our existence proof will encourage the development of simpler systolic algorithms which also achieve the time bound $O(n \log n)$. Consequently, we do not attempt to specify our algorithm more precisely than is necessary to establish the $O(n \log n)$ time bound. For example, we do not specify whether the operation of each systolic processor is programmed as a function of n and the time step t (which is certainly possible) or whether the processors are driven by "data ready" lines and other control lines from their neighbors (which might provide a simpler implementation).

Although our algorithm seems to be too complicated for direct hardware implementation at the present time, it is encouraging from the viewpoint of computational complexity. Its main competitor is the parallel Jacobi method of Brent and Luk [2, 3], which finds the eigenvalues of A directly, without first reducing A to tridiagonal form, and also takes time $O(n \log n)$ using $O(n^2)$ processors. The parallel Jacobi method is simpler, and faster by a small constant factor. On the other hand, the new algorithm can easily be extended to reduce an unsymmetric matrix to upper Hessenberg form.

The paper is organized as follows. Preliminary results are given in Section 2, and the systolic model of computation is described in Section 3. The basic idea of the algorithm is outlined in Sections 4 and 5. Section 6 is devoted to the systolic organization. Details and proofs are given in Section 7. Extensions to matrix bidiagonalization and the reduction of a nonsymmetric matrix to Hessenberg form are mentioned in Section 8.

2. Givens Rotations

The reduction of an $n \times n$ symmetric matrix $A = (a_{ij})$ to tridiagonal form can be obtained by means of plane rotations applied to A on the left and on

the right in some prescribed order. A plane rotation is defined by a matrix

$$P_{i+1,j} = \begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \overset{i \text{ column}}{\overset{\downarrow}{\begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & c_i & s_i & \cdots & \\ & & -s_i & c_i & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix}}} \end{array} \; i \text{ row}$$

The matrix $P_{i+1,j}$, $i > j$, applied on the left rotates rows i and $(i + 1)$ of A so as to annihilate the off-diagonal element $a_{i+1,j}$. The parameters of $P_{i+1,j}$ are defined by

$$\begin{aligned} \bar{a}_{ij} &= (a_{ij}^2 + a_{i+1,j}^2)^{1/2} \\ c_i &= a_{ij}/\bar{a}_{ij} \\ s_i &= a_{i+1,j}/\bar{a}_{ij}. \end{aligned}$$

Rows i and $(i + 1)$ and columns i and $(i + 1)$ of the symmetric matrix $\bar{A} = P_{i+1,j} A P_{i+1,j}^{\mathrm{T}}$ are given by

$$\left.\begin{aligned} \bar{a}_{ip} = \bar{a}_{pi} &= a_{ip}c_i + a_{i+1,p}s_i \\ \bar{a}_{i+1,p} = \bar{a}_{p,i+1} &= -a_{ip}s_i + a_{i+1,p}c_i \end{aligned}\right\} \; p \neq i, i+1, j$$

$$\bar{a}_{i+1,j} = \bar{a}_{j,i+1} = 0$$

$$\begin{bmatrix} \bar{a}_{ii} & \bar{a}_{i,i+1} \\ \bar{a}_{i+1,i} & \bar{a}_{i+1,i+1} \end{bmatrix} = \begin{bmatrix} c_i & s_i \\ -s_i & c_i \end{bmatrix} \begin{bmatrix} a_{ii} & a_{i,i+1} \\ a_{i+1,i} & a_{i+1,i+1} \end{bmatrix} \begin{bmatrix} c_i & -s_i \\ s_i & c_i \end{bmatrix}.$$

The orthogonal matrix Q is formed as the product of plane rotations $P_{i+1,j}$ such that the transformed matrix $\bar{A} = QAQ^{\mathrm{T}}$ is tridiagonal; see [8] for details.

3. The Systolic Model of Computation

We want to realize reduction to tridiagonal form by means of a systolic array [2, 7]. Roughly speaking a systolic array is an array of simple processors which operate synchronously. Every processor is able to perform certain floating point operations and has several words of local memory. The processor layout is assumed to be at most two dimensional and processors can

communicate only with their nearest neighbors. The cost of data transmission between neighboring processors is comparable to the cost of the most expensive single operation performed by any processor in the array. Knowing what operations the processors must perform in order to solve a problem, we define a *time unit* to be the maximal time that is necessary for a processor to perform the most time consuming operation together with loading and unloading its registers (see Section 6). A synchronization mechanism allows processors to exchange data at time instants separated by integer multiples of a time unit. We assume that the number of processors available is a function of the problem size, although in practice this number will of course be bounded.

4. The Basic Idea

Direct mapping of the sequential tridiagonalization algorithm to a parallel algorithm for a square array of systolic processors yields processing time of order n^2. It may be shown that this is a consequence of the fact that in the sequential algorithm the order in which elements are annihilated ensures that no new nonzeros are reintroduced during the process of reduction. By relaxing this condition, i.e., by allowing the introduction of new nonzeros, we are able to achieve $O(n \log n)$ time on n^2 processors.

The idea is as follows. In every major step we intend to halve the bandwidth of the matrix. If this could be achieved in time independent of bandwidth and linear in problem size then we would have an $O(n \log n)$ algorithm. Assume that at the mth major step our matrix has a band B_w with w subdiagonals (see Fig. 1).

We annihilate elements within the lower half of the band in some prescribed order (see Section 5). Because of postmultiplications new nonzeros are created forming a bulge beneath the lowest subdiagonal. This bulge is

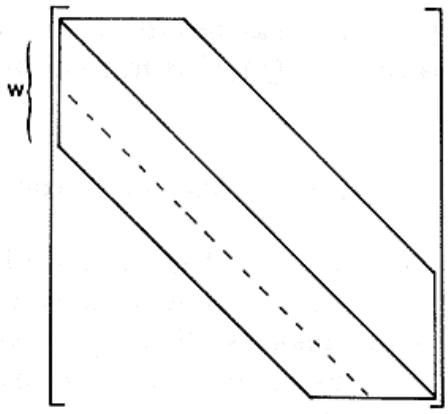

FIG. 1. Regions for bandwidth reduction.

annihilated next, resulting in a second bulge further down the lowest subdiagonal. We end when the bulge is chased out of the matrix. As will be shown in Section 5 the time spent in processing all elements in the lower half of the band is $O(n)$. Thus, by multiprocessing and pipelining we are able to halve the bandwidth in time independent of the bandwidth. This in turn yields $O(n \log n)$ total time for reducing the matrix to tridiagonal form.

5. Elimination Ordering

Let the current band have $w \geq 3$ subdiagonals; the case $w = 2$ is treated in Section 7. We want to annihilate the $k/2$ lowest subdiagonals of the band, where $k = w - 1$ for w odd or $k = w - 2$ for w even. First we annihilate the element in the lower left corner. If the element in position (i, j) is eliminated at time t then the element above, i.e., in position $(i - 1, j)$, is eliminated at time $t + 3$ and the element to the right, i.e., in position $(i, j + 1)$, is eliminated at time $t + 9$. This is illustrated in Fig. 2 for $k = 10$, $w = 12$. The general formula is the following. We eliminate element a_{ij} at time

$$t_{ij} = c_w + 3(w - i) + 9j, \tag{5.1}$$

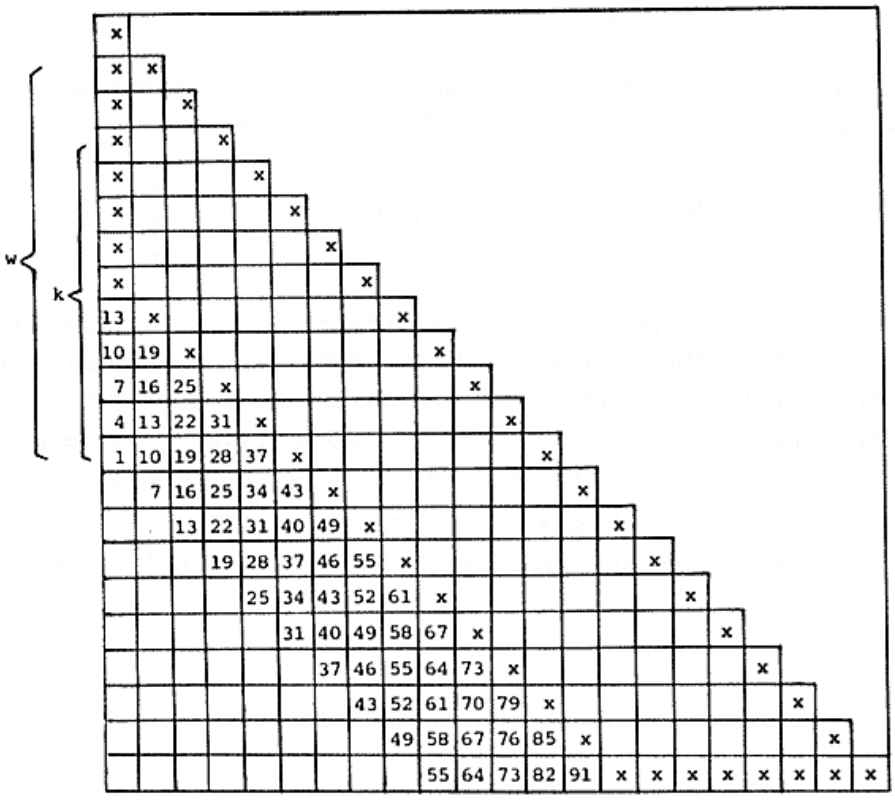

FIG. 2. Elimination ordering ($w = 12$, $k = 10$).

where c_w is chosen in such a way that there is no interference between computations for different bands (see Section 7).

6. Array Organization

We assume an $n \times n$ square network of processors. Every processor, except for those on a boundary, is connected to its eight nearest neighbors. The algorithm can clearly be modified to work on an array where each processor is connected to its four nearest neighbors. We distinguish three types of operation which processors are able to perform:

1. Determination of rotation parameters; i.e., given $[a, b]^T \in \mathbb{R}^2$, determine $c, s \in \mathbb{R}$ such that

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} (a^2 + b^2)^{1/2} \\ 0 \end{bmatrix}.$$

2. Single left or right rotation; i.e., given rotation parameters c and s and a column or row vector $[a, b]^T$ or $[a, b]$, compute

$$\begin{bmatrix} \bar{a} \\ \bar{b} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} \quad \text{or} \quad [\bar{a}, \bar{b}] = [a, b]\begin{bmatrix} c & -s \\ s & c \end{bmatrix}.$$

3. Simultaneous left and right rotation; i.e., given rotation parameters c_1, s_1 and c_2, s_2 and a 2×2 submatrix $\begin{bmatrix} u & v \\ z & y \end{bmatrix}$, compute

$$\begin{bmatrix} \bar{u} & \bar{v} \\ \bar{z} & \bar{y} \end{bmatrix} = \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}\begin{bmatrix} u & v \\ z & y \end{bmatrix}\begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix}.$$

In order to perform operation 1 or 2 processors form conceptual clusters of two cells with a vertical connection for the left transformation and a horizontal connection for the right transformation; similarly, they form clusters of four cells when operation 3 is to be performed (see Fig. 3). For example, the cluster in Fig. 3a consists of two processors which initially (before operation 2) store a vector $\begin{bmatrix} a \\ b \end{bmatrix}$ and finally (after operation 2) store $\begin{bmatrix} \bar{a} \\ \bar{b} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}$. The details of exactly how this could be implemented will not be specified pre-

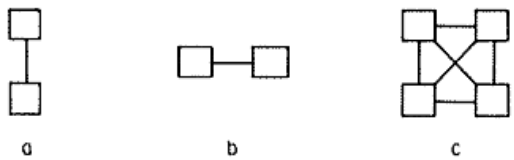

FIG. 3. Possible configurations of clusters.

cisely, as it is enough for our purposes if the reader is convinced that an implementation is possible.

We assume that operation 1 or 2 takes one unit of time while operation 3 takes two units of time (including data transfer time). At the beginning of the computation each processor contains exactly one element of the matrix A. In the course of the computation modified elements of the transformed matrix A stay in the corresponding processors while generated rotation parameters travel along horizontal and vertical connections. We assume that the network works in a synchronous manner and that the propagation delay is one unit of time. This means that rotation parameters cannot be broadcast but that their transfer from one processor to neighboring processors in one unit of time is possible.

Assume that the current bandwidth is w and, according to the ordering described in Section 5, the next element to be annihilated at time t_s is at position $(i + 1, j)$. At time t_s processors in positions (i, j) and $(i + 1, j)$ form a cluster of type (a) as shown in Fig. 3 and perform an operation of type 1. Then rotation parameters are sent to the right where a cluster of two or four processors is formed. In the former case a left transformation is performed, taking one unit of time, and rotation parameters are sent further to the right. The latter case corresponds to the situation when left and right transformations are to be performed simultaneously on a 2×2 submatrix of the matrix A; i.e., a cluster of four processors simultaneously receives rotation parameters from the left and top. This operation takes two units of time. Next rotation parameters propagate to the right and to the bottom clusters. Define a "channel" to be a pair of adjacent rows or columns of processors. Rotation parameters that propagate along a horizontal channel eventually reach processors on a diagonal. Here, a cluster of four processors is formed which executes simultaneous left and right rotations using the same rotation parameters for both left and right transformations. Now, rotation parameters stop traveling to the right. Instead they are reflected from the diagonal and move downward, becoming a source of right transformations, until the lowest nonzero codiagonal is reached. Here, when a new nonzero is created, the old rotation parameters are no longer needed and can be discarded. As we started at time t_s from a cluster of processors in positions (i, j) and $(i + 1, j)$, and moved with unit speed, we had to arrive in position $(i + w + 1, i)$ at time $t_e = t_s + (i + 1 - j) + w$. At time t_e, a new nonzero is introduced in position $(i + w + 1, i)$. The new nonzero is annihilated at the next unit of time, i.e., at time $t_e + 1$, by a cluster of processors in positions $(i + w, i)$ and $(i + w + 1, i)$. New rotation parameters start to propagate to the right along a horizontal channel. When they reach the diagonal they are reflected and propagate down a vertical channel. This process is repeated until the bulge is chased out of the matrix; see Fig. 4.

As annihilation of an element causes emergence of a bulge we must ask whether some rotations that chase bulges interfere with each other or with

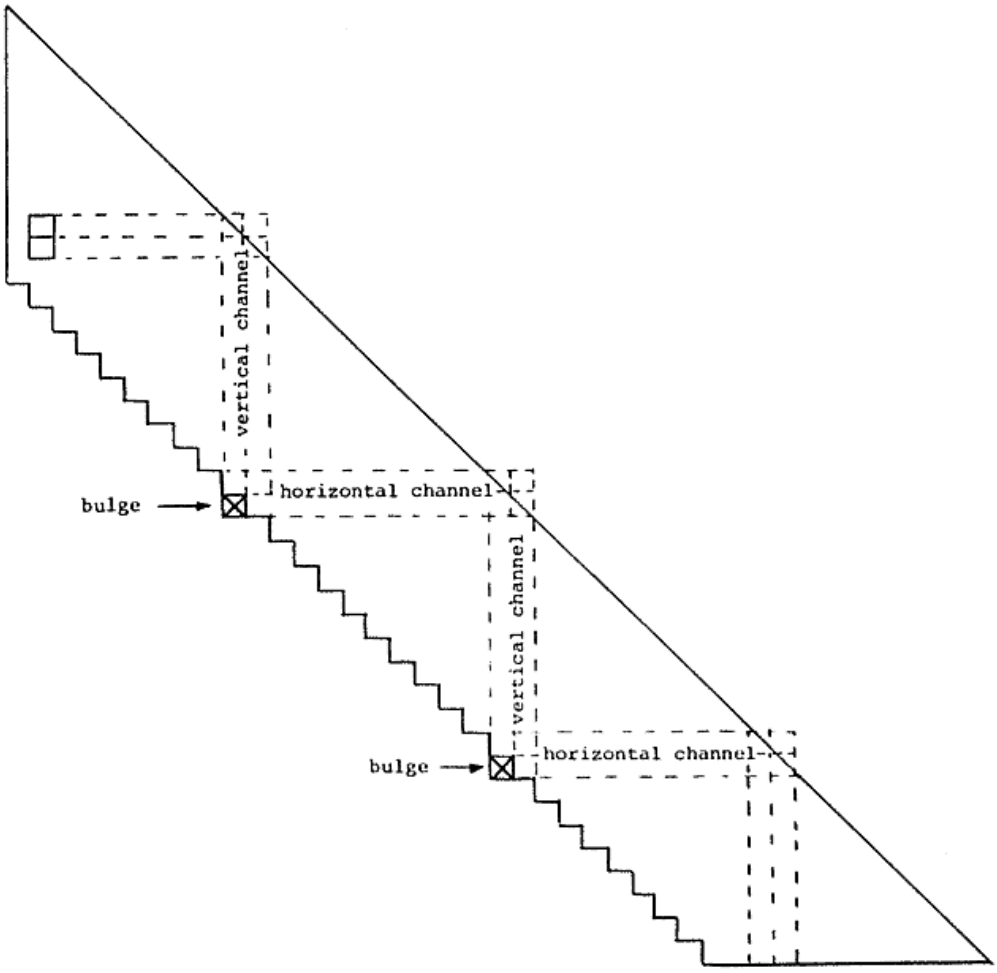

FIG. 4. Chasing a bulge.

original rotations. In Section 7 we show that it is possible to perform many rotations in parallel without interference.

7. OPERATION OF THE ARRAY

In the process of elimination left and right transformations, i.e., pre- and postmultiplications, may interfere with each other. We shall refer to the interference between left and right transformations such that some matrix element is involved in left and right transformations at the same unit of time as a *conflict*.

We have to ensure that transformations are performed in the correct order and that conflicts are resolved accordingly. There are four kinds of conflict possible. As left and right rotations cross at some 2×2 submatrix we describe conflicts in terms of the 2×2 submatrix

$$S = \begin{bmatrix} a_{ij} & a_{i,j+1} \\ a_{i+1,j} & a_{i+1,j+1} \end{bmatrix},$$

left rotation

$$T_l = \begin{bmatrix} c_l & s_l \\ -s_l & c_l \end{bmatrix},$$

and right rotation

$$T_r = \begin{bmatrix} c_r & -s_r \\ s_r & c_r \end{bmatrix}.$$

Consider the transformation

$$\begin{bmatrix} \bar{a}_{ij} & \bar{a}_{i,j+1} \\ \bar{a}_{i+1,j} & \bar{a}_{i+1,j+1} \end{bmatrix} = \begin{bmatrix} c_l & s_l \\ -s_l & c_s \end{bmatrix} \begin{bmatrix} a_{i,j} & a_{i,j+1} \\ a_{i+1,j} & a_{i+1,j+1} \end{bmatrix} \begin{bmatrix} c_r & -s_r \\ s_r & c_r \end{bmatrix}.$$

A conflict of the first kind occurs when a left rotation T_l on $[a_{ij}, a_{i+1,j}]^T$ and right rotation T_r on $[a_{ij}, a_{i,j+1}]$ are to be performed at the same unit of time.

The second kind of conflict involves a left rotation on $[a_{ij}, a_{i+1}, j]^T$ and right rotation on $[a_{i+1,j}, a_{i+1,j+1}]$ while the third kind occurs when vectors $[a_{i,j+1}, a_{i+1,j+1}]^T$ and $[a_{i,j}, a_{i,j+1}]$ are to be transformed simultaneously by rotations T_l and T_r, respectively. Finally, a conflict occurs when T_l rotates vector $[a_{i,j+1}, a_{i+1,j+1}]^T$ and T_r rotates vector $[a_{i+1,j}, a_{i+1,j+1}]$, but this implies that one time unit earlier a conflict of the first kind took place. For this reason we need to consider only three kinds of conflict; see Fig. 5.

LEMMA 1. *Let T_{is} and T_{ju} be transformations that zero elements a_{is} and a_{ju} within the current band. If $i - j \geq k$ then T_{is} does not interfere with T_{ju}. If $i - j < k$ then only a conflict of the first kind can occur.*

Proof. Let T_{is} and T_{ju} be such that $i - j = k$. The earliest element to be annihilated in row i is the element in column $i - w$. The last element to be annihilated in row j is the element in column $j - (w - k/2 + 1)$; see Fig. 6. Consider the case when $s = i - w$ and $t = j - (w - k/2 + 1)$. Since transformation T_{ju} propagates with unit speed—first along the horizontal, then along the vertical channel—it has to arrive in position $(j + w, j - 1)$ where

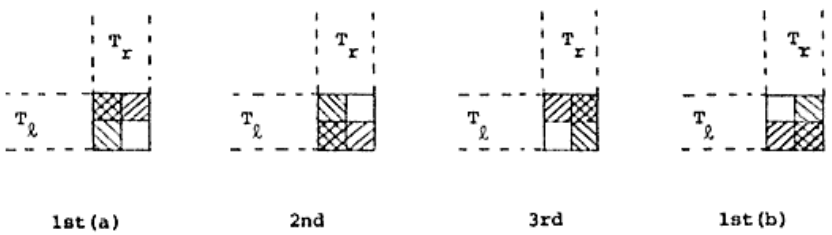

FIG. 5. Different kinds of conflict.

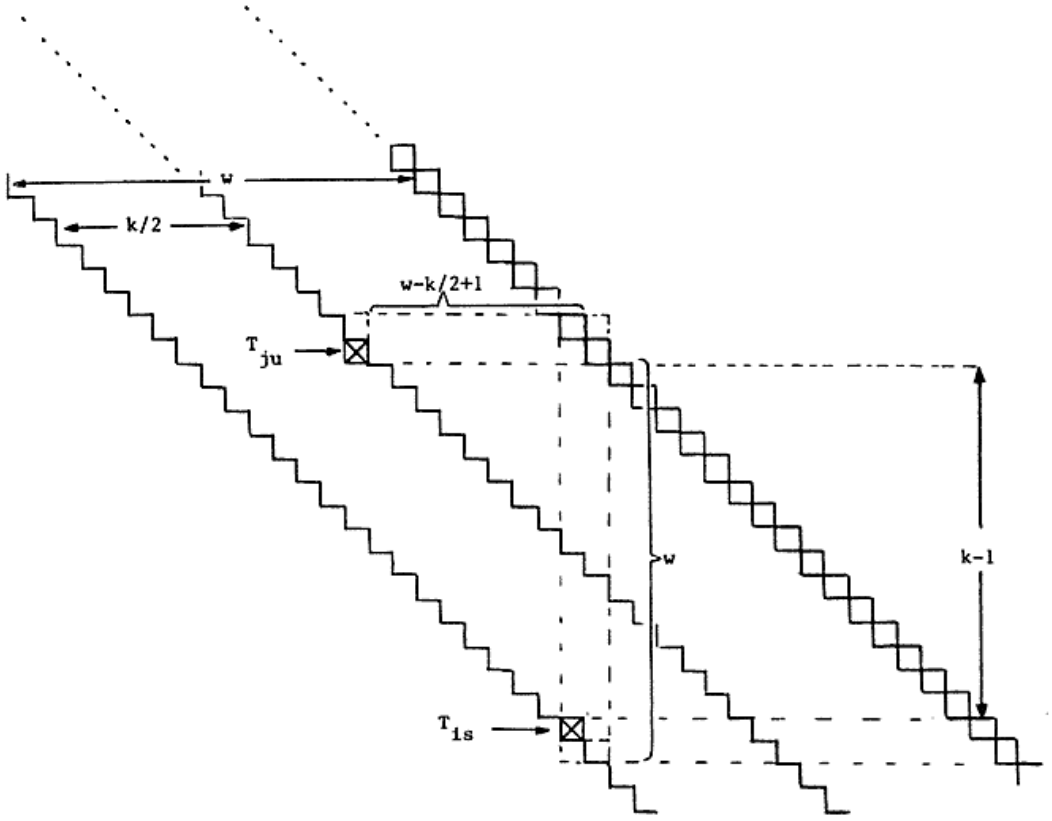

FIG. 6. See Lemma 1.

a new nonzero is created after

$$w + (w - k/2 + 1) = 2w - k/2 + 1 \tag{7.1}$$

units of time. The position $(j + w, j - 1)$ corresponds to the position $(i + 1, s)$ when

$$k = w - 1 \tag{7.2}$$

or to the position $(i + 2, s + 1)$ when

$$k = w - 2. \tag{7.3}$$

In the former case (7.1) becomes $\frac{3}{2}k + 3$ while in the latter case (7.1) becomes $\frac{3}{2}k + 5$.

From (5.1) we get

$$\begin{aligned} t_{is} - t_{ju} &= 3(j - i) + 9(s - u) \\ &= 6(i - j) - \tfrac{9}{2}k + 9 \\ &= \tfrac{3}{2}k + 9. \end{aligned} \tag{7.4}$$

Comparing (7.1) with (7.4) for the cases (7.2) and (7.3) we see that transformation T_{ju} precedes transformation T_{is} by at least four units of time. Thus transformations T_{ju} and T_{is} do not interfere for $u = j - (w - k/2 + 1)$, $s = i - w$.

Now any other transformation T_{lr} with $l \leq j$ and $r \leq$ u precedes transformation T_{ju} and in consequence must precede T_{is}. Similarly, any transformation T_{qp} with $q \geq i$ and $p \geq s$ is preceded by transformation T_{is}. Thus no transformation T_{is} that zeros element a_{is} interferes with transformation T_{ju} that zeros element a_{ju} as long as $i - j \geq k$.

Now consider the case $i - j < k$.

Let us assume that a conflict has been caused by left transformations T_{is} and T_{ju}, where $j < i$ and $u < s$ (it is easy to see that for $u \geq$ s conflict cannot occur). If transformations T_{is} and T_{ju} start at time $t(T_{is})$ and $t(T_{ju})$, respectively, and arrive simultaneously at location 1°, 2°, 3°, or 4° (see Fig. 7) then as they propagate with constant unit speed we have the relation

$$\begin{aligned} t(T_{is}) - t(T_{ju}) &= (s - u) + (i - j) && \text{for case 1° or 4°} \\ &= (s - u) + (i - j) + 1 && \text{for case 2°} \\ &= (s - u) + (i - j) - 1 && \text{for case 3°.} \end{aligned}$$

On the other hand because of the ordering imposed by (5.1), we have another relation,

$$t(T_{is}) - t(T_{ju}) = 9(s - u) - 3(i - j).$$

Comparing these two relations we conclude that conflict can occur if and only if

$$\begin{aligned} 8(s - u) - 4(i - j) &= 0 && \text{for case 1° or 4°} \\ &= 1 && \text{for case 2°} \\ &= -1 && \text{for case 3°.} \end{aligned}$$

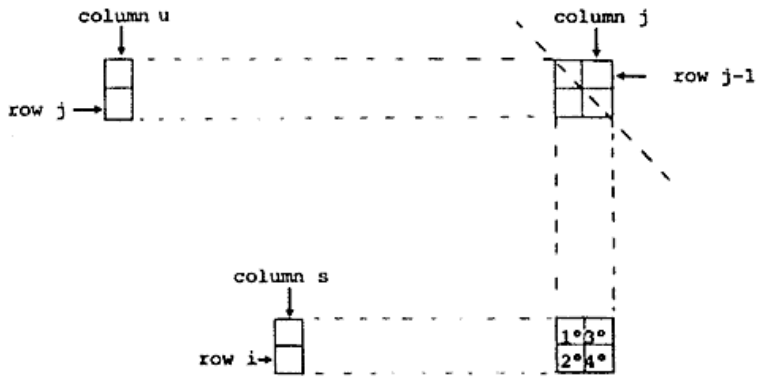

FIG. 7. See Lemma 1.

Since the left-hand side is divisible by 4, only conflicts of the first kind can take place. ■

COROLLARY. *Conflicts are caused only by left transformations T_{is} and T_{ju} for which*

$$2(s - u) = i - j, \qquad j < i \text{ and } u < s.$$

Now we identify the pattern of potential conflicts in a given unit of time. If a 2×2 submatrix

$$S_{ij} = \begin{bmatrix} a_{ij} & a_{i,j+1} \\ a_{i+1,j} & a_{i+1,j+1} \end{bmatrix}$$

is involved in a conflict caused by transformations T_{is} and T_{ju} then from the assumed annihilation ordering the following 2×2 submatrices are also possibly in conflict as long as they are contained in the band:

$$S_{(i-k)-2l,(j-3k)+2l} \qquad \text{for } k = 0, \pm 1, \pm 2, \ldots, \quad l = 0, \pm 1, \pm 2, \ldots. \tag{7.5}$$

Also, because of the assumed ordering of annihilation, at any given two successive units of time only processors corresponding to matrices (7.5) are active. Thus conflicts occur in disjoint and well-separated submatrices and can recur every two units of time. This suggests the following solution to handle conflicts.

When a conflict occurs and we know that it can only be a conflict of the first kind, the four processors involved form a cluster as described in Section 6. Within this cluster first a left transformation on the whole 2×2 submatrix is performed, then a right transformation also on the whole 2×2 submatrix. Here it is assumed that a cluster of four processors is capable of performing left or right transformations on 2×2 submatrices in one unit of time. This means that after two units of time the transformed 2×2 submatrix has the correct value.

After computation involving matrices described by (7.5) have been completed the new activities and new possible conflicts occur in submatrices described by (7.5) where j is replaced by $j + 2$ (as the left rotation moves two positions right and the right rotation moves two positions down).

LEMMA 2. *Let T_{is} be a transformation that zeros element a_{is} in the current band and let T_{ju} be a transformation that zeros some bulge. Transformations T_{is} and T_{ju} never interfere with each other.*

Proof. Every transformation that annihilates bulges is a successor of some transformation that annihilates elements within the band. Thus by

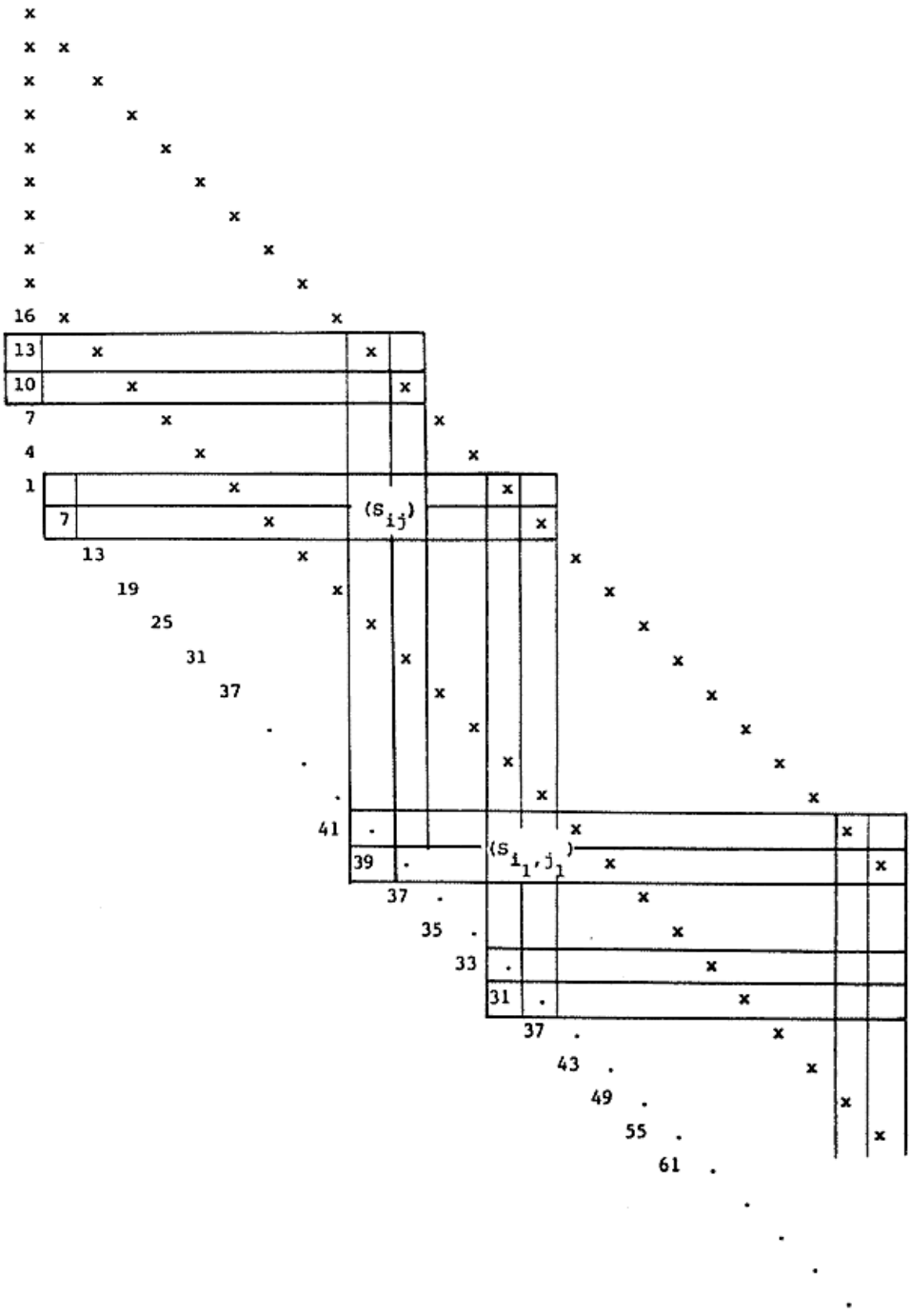

FIG. 8. See Lemma 2.

Lemma 1 it is sufficient to consider only such transformations T_{is} and T_{ju} whose predecessors annihilate elements that lie in rows which are separated by less than k rows. Assume that a conflict has occurred in some submatrix S_{i_1,j_1} (see Fig. 8). This can only happen when left and right transformations are to be performed simultaneously on some element of submatrix S_{i_1,j_1}. *Assume now that the left transformation* $T_l(i_1, j_1)$ is a reflection from the lowest codiagonal

of the right transformation $T_r(i,j)$ that transforms submatrix S_{ij}. Consider the left transformation $T_l(i,j)$, that is involved in conflict with $T_r(i,j)$ in submatrix S_{ij}. The left transformation reflects along the diagonal and becomes a right transformation $T_r(i_1,j_1)$ that passes through submatrix S_{i_1,j_1}. Because right transformation $T_r(i,j)$ introduces a new nonzero below the codiagonal this nonzero has to be annihilated, as soon as possible under the constraint that we can move with unit speed, by left transformation $T_l(i_1,j_1)$. This additional annihilation causes transformation $T_l(i_1,j_1)$ to arrive at submatrix S_{i_1,j_1} two units of time later than right transformation $T_r(i_1,j_1)$. Thus there is no conflict between $T_r(i_1,j_1)$ and $T_l(i_1,j_1)$.

By examining the neighbor clusters of processors that are active at the same time as the cluster corresponding to submatrix S_{ij}, we conclude that a conflict in position described by submatrix S_{i_1,j_1} is not possible. As S_{i_1,j_1} has been chosen arbitrarily, conflict cannot occur. ■

Next we show that at any given unit of time bulges that are introduced beneath the lowest subdiagonal can lie only on a single codiagonal next to the lower boundary of the band before they are annihilated. Observe that the first wave of bulges which arrive on the codiagonal next to the lower boundary is due to annihilation of elements that lie along the left and lower boundary of the band (see Fig. 8). The length of this new codiagonal is $k - 1$ with the middle bulge corresponding to annihilation of the lower left corner (this corner is eliminated first). The middle bulge emerges first, then every two units of time successive bulges arrive above and every six units of time successive bulges arrive below. According to the annihilation scheme, the second wave arrives eight units of time later than the first wave. Thus there is enough time to annihilate any two successive bulges before arrival of the next wave. This ensures that bulges cannot accumulate. We can conclude that all transformations are performed in a correct order.

Recall that we eliminate element a_{ij} in the lower half of the current band at time

$$t_{ij} = c_w + 3(w - i) + 9j \qquad \text{for } w - k/2 + 1 \leq i - j \leq w.$$

From the above formula it follows that annihilation of the lowest $k/2$ subdiagonals in the current band takes $6n - w + O(1)$ units of time. Clearly there is no interference between computations for two successive cycles, i.e., for band B_w, and the next band $B_{\bar{w}}$, if

$$c_{\bar{w}} - c_w \geq 6n.$$

Note, however, that computations for two successive bands can be overlapped. In fact it is sufficient to have $c_{\bar{w}} - c_w \geq 4n$, as is explained in the following lemma.

LEMMA 3. *Let B_w and $B_{\bar{w}}$ be two successive bands, $\bar{w} < w$. If*

$$c_{\bar{w}} - c_w \geq 4n,$$

then computations for band B_w do not interfere with computations for band $B_{\bar{w}}$.

Proof. The time to eliminate from $a_{w+1,1}$ to $a_{n,n-\bar{w}-1}$ is $6(n - w - 1) + 9(w - \bar{w} - 1) = 6n + 3w - 9\bar{w} + O(1)$. The next band has $\bar{w} = \lceil w/2 \rceil$ for w odd or $\bar{w} = \lceil w/2 \rceil + 1$ for w even and an elimination that started at $a_{\bar{w}+1,1}$ takes time at most $2(n - \bar{w})$ to propagate down to position $(n, n - \bar{w} - 1)$. Thus we can overlap by $2(n - \bar{w})$ and start eliminating $a_{\bar{w}+1,1}$ at time

$$6n - 3w - 9\bar{w} - 2(n - \bar{w}) < 4n$$

after eliminating $a_{w+1,1}$. ■

The main result of this paper immediately follows from Lemma 3 and the observation that tridiagonalization of the five-diagonal matrix, i.e., the case $w = 2$, can be achieved in time $O(n)$ by means of the standard sequential algorithm. Hence we have:

Theorem. Tridiagonalization of an $n \times n$ symmetric matrix can be achieved in time $4n \log_2 n + O(n)$ using an array of n^2 mesh-connected processors.

8. CONCLUDING REMARKS

It is clear that, by taking advantage of symmetry, we need only a triangular array of $n^2/2 + O(n)$ processors, rather than a square array. Also, it is possible to accumulate the transformations (i.e., to compute the matrix Q of Section 2) using the same array of processors.

The concept of band halving can be applied to the reduction of a general $n \times n$ matrix to upper Hessenberg form as well as to the reduction of a triangular matrix to bidiagonal form. By a straightforward extension of our method both problems can be solved in time $O(n \log n)$ on a two-dimensional array of n^2 processors.

We have shown that certain matrix reductions which are usually performed as a preliminary step before application of the QR algorithm can be done in time $O(n \log n)$ on a systolic array. However, as mentioned in Section 1, it is not clear if this is the best approach to the solution of eigenvalue problems on systolic arrays. Iterative methods such as the Jacobi method of [2, 3], which do not require any preliminary reductions, are certainly much simpler and thus more easily implemented.

We conjecture that the bound $O(n \log n)$ is best possible, i.e., that any parallel algorithm which tridiagonalizes a symmetric matrix by the application of plane rotations must take time at least $Kn \log n$ for some positive constant K. However, we have not been able to prove this conjecture.

ACKNOWLEDGMENT

We thank Professor H. T. Kung for suggesting the problem and for helpful discussions during November 1982. We also thank the referees for their comments, which have helped to improve the clarity of the paper.

REFERENCES

1. Bojanczyk, A., Brent, R. P., and Kung, H. T. Numerically stable solution of dense systems of linear equations using mesh-connected processors. *SIAM J. Sci. Statist. Comput.* **5** (1984), 95–104.
2. Brent, R. P., Kung, H. T., and Luk, F. T. Some linear-time algorithms for systolic arrays. In Mason, (Ed.). *Information Processing 83*. North-Holland, Amsterdam, 1983, pp. 865–876.
3. Brent, R. P., and Luk, F. T. The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM J. Sci. Statist. Comput.* **6** (1985), 69–84.
4. Heller, D. E., and Ipsen, I. C. F. Systolic networks for orthogonal equivalence transformations and their applications. *Proc. 1982 Conf. on Advanced Research in VLSI*, MIT, Cambridge, Mass., 1982, pp. 113–122.
5. Kung, S. Y., and Gal-Ezer, R. J. Linear or square array for eigenvalue and singular value decomposition. *Proc. USC Workshop on VLSI and Modern Signal Processing*, Nov. 1982, pp. 89–98.
6. Schreiber, R. Systolic arrays for eigenvalue computation. *Proc. SPIE Symp. East 1982*, Vol. 341, *Real-Time Signal Processing V*, Society of Photo-optical Instrumentation Engineers, 1982.
7. Ullman, J. D. *Computational Aspects of VLSI*, Computer Science Press, Rockville, Md., 1984, Chap. 5.
8. Wilkinson, J. H. *The Algebraic Eigenvalue Problem*. Oxford Univ. Press (Clarendon), London/New York, 1965.