Singular Value Computations with Systolic Arrays

Ilse C.F. Ipsen

Research Report YALEU/DCS/RR-291
November 1983

The work presented in this paper was supported by the Office of Naval Research under contract N000014-82-K-0184.

Abstract

Systolic arrays are constructed for bandwidth reduction and singular value decomposition of $m \times n$ matrices, wlog, $m \geq n$. The underlying algorithms are unconditionally stable.

Independently of the bandwidth, w, or the order of the matrix, a $4\bar{k}^2$ processor array accomplishes a reduction to bidiagonal form in time $O(wm^2/\bar{k}^2)$; subsequent determination of a singular value by the Golub-Reinsch SVD iteration takes $O(n)$ steps. With $O(m)$ $4w^2$ processor arrays the reduction time becomes sublinear, resulting in $O(m/w+sn)$ steps to compute s singular values, compared to sequential times of $O(m^2+sn)$ or $O(m^2w^2+sn)$. The array sizes, in contrast to many other designs, do not depend on the order, m or n, of the matrix rendering it possible to process problems of arbitrary size. Since input and output occurs, as in previous designs [Help83, KuLe78], by diagonals arrays can be directly appended to further reduce the computation time. Consequently, the designs will be most efficient for matrices with a fairly small and dense band.

Introduction

In response to the increasing demands for computing power as well as for real time processing, which exceed the capabilities of single processor systems, the implementation of systolic arrays in very large scale integrated (VLSI) circuits represents a cost effective way to exploit parallelism.

A systolic array can be described as an array of simple processors that routes data in a regular fashion, fully exploits parallel-pipelined processing and makes maximum use of data fetched from memory. The basic design philosophy is stated in [KuLe78, Kung81], according to which numerous designs for triangular decompositions, linear system solution and eigenvalue problems have been proposed (some of those are surveyed in [HeIp83]). The present paper introduces designs for bandwidth reduction and singular value computations of densely banded matrices.

Since systolic arrays rely heavily on pipelining, they are very efficient for long matrices. Hence, the number of processors per array (including I/O ports) should not depend on the order of the matrix. Aside from its order, $m \times n$, another topological characteristic of a matrix is the bandwidth, w. Frequently, w is small ($w = 3, 5$ for PDEs), so inexpensive arrays of w or w^2 processors can process arbitarily long matrices and achieve excellent hardware utilisation. Furthermore, input and output does not occur by rows or columns but rather codiagonals (subdiagonals, diagonal, superdiagonals). If input and output addressing schemes are identical, it becomes feasible to chain several (possibly different) arrays so as to directly pipeline the results from one into the other obviating the need for intermittent memory transfers. From a numerical point of view, the underlying algorithms must be unconditionally stable, on account of the largely data independent control, and permit easy decomposition of the problem in case of a disagreement between hardware and input sizes.

The already existing designs for singular value computations (SVD) possess a hardware complexity proportional to the order of the matrix, superlinear time complexity and little capability for handling problems that are larger than the given hardware. An $O(n^2)$ processor singular value array of [FiLP82] relies on a version of Hestenes' method where intermediate quantities are not properly updated; a formal convergence proof is not provided. [BrLu82] construct an array for a one-sided orthogonalisation method due to Hestenes which uses either a linearly connected mesh of $O(n)$ processors and $O(mn)$ steps to determine a singular value or else a two-dimensional $O(mn)$ processor array with a non-planar interconnection structure and time $O(n \log m)$. The method is quadratically convergent; experience suggests that 6 to 10 iterations per singular value provide for sufficient numerical accuracy. In [BrLV83a] a similar architecture of $O(n^2)$ processors implements a cyclic Jacobi method for the SVD in $O(m + n \log n)$ steps. With the addition of QR and matrix multiplication arrays, the generalised SVD for matrices $A \in R^{m \times n}$ and $B \in R^{p \times n}$ is computed in $O(l \log n)$ units of time on $O(l^2)$ processors, $l \geq m, n, p$. With reference to the previous works, Schreiber [Schr83b] suggests a method to cope with problems that do not match the array size. In [Schr83a] he proposes a kn processor design which reduces a matrix to upper triangular form with bandwidth $k+1$ in time $O(mn/k)$. A $k(k+1)$ processor array from [HeIp83] is used to implement a SVD iteration on $k+1$ diagonal matrices (analogous to the Golub-Reinsch iteration for bidiagonal matrices) in time $6n + O(k)$. For $k = O(\sqrt{n})$ this amounts to processor and hardware requirements of $O(n^{3/2})$.

In contrast, the final bandwidth reduction design presented here consists of l planar, orthogonally connected arrays of $4\bar{k}(\overline{ck} + 1)$ processors each with an array traversal time of $2(n + 4\overline{lk})$ steps, $\bar{c} \geq 1$. An iteration of the subsequent Golub-Reinsch SVD iteration for bidiagonal matrices can be done in $2n$ steps. On the average 3 iterations are required to find a singular value [Parl80]. The time to find s singular value comes to $O(wm^2/\overline{lk}^2 + sn)$ and $O(m^2/w + sn)$ if $\bar{k}$ is proportional to w. In the event that $\bar{l} = O(m)$ arrays are available the time is further reduced to $O(m/w + sn)$ as opposed to the sequential process with $O(m^2w^2 + sn)$ steps. The bandwidth reduction array is flexible in the sense that it can handle matrices of arbitrary size and bandwidth. The data addressing scheme is identical to the one employed in [HeIp83, KuLe78]; consequently a variety of

arrays can be chained to avoid costly memory transfers and repeated alignment of data. The basic operations in all algorithms are generation and application of Givens' plane rotations, resulting in unconditional stability of all computations.

The following is an overview of the methods to be implemented :

1. Algorithm BWO. A $2 \times (w-3)$ array removes the leading element of one subdiagonal and one superdiagonal. If $\bar{l}$ arrays are chained, a reduction to bidiagonal form takes $O(n^2 w/\bar{l})$ steps.
2. Algorithm BWK. A $4\bar{k} \times (\overline{ck}+1)$ array removes the leading $(w-1-\bar{k})$ elements of $\bar{k}$ subdiagonals or $\bar{k}$ superdiagonals. If $\bar{l}$ arrays are chained, bidiagonal form is achieved in $O(n^2 w/\overline{lk}^2)$ steps.
3. Algorithm SVE. A 4 processor matrix transposition array directly chained to a 4 processor matrix multiplication array forms the product $A^T A$ in time $2n+4$, while a 3 processor linearly connected mesh accomplishes one (explicitly shifted) QR iteration in $4n+2$ steps.
4. Algorithm SVI. A 5 processor array performs one (implicitly shifted) Golub-Reinsch iteration for bidiagonal matrices in time $2n+3$. An analysis of numerical accuracy, computation time and hardware flexibility reveals a combination of 2 and 4 as most advantegeous.

As for the organisation of this paper, the sequential versions of the algorithms are discussed in the above order, followed by a description of their systolic, parallel implementations. One paragraph on matrix transposition arrays is included.

The following assumptions will be made throughout the paper. The terms 'cell' and 'processor' are used interchangeably. Processors perform one of three operations : generation of plane rotations, application (propagation) of plane rotations or shifting of elements (output of an element unaltered to the left, right or top of the processor). A systolic array is made up of linearly connected meshes of processors (introduced in [Help83]) which eliminate at most one subdiagonal or one superdiagonal. Cells of a linearly connected mesh are numbered consecutively from left to right, starting with 1. Linear meshes within an array are numbered from bottom to top, again starting with 1. Overbars, e.g., $\bar{w}$, $\bar{k}$, indicate hardware parameters.

As a matter of consistency, time is measured with regard to plane rotations. One unit of time (or time step) is long enough to acommodate either the generation or the application of a plane rotation. For a detailed account on the execution time of addition, multiplication and square root operations involved in a rotation the reader is referred to [SaKu78]. Implementation of the algorithms is not affected by the synchronicity or asynchronicity of the arrays. However, the determination of computation speed is based on synchronous designs, where the computation time of the slowest processor is taken as a unit. The traversal time of an $m \times n$ matrix, $m \geq n$, through an array of $\bar{k}$ linear meshes is

$$T(m, \bar{k}) \equiv 2(m + \bar{k}) - 1.$$

In most of the figures, only the indices instead of the matrix elements proper will be displayed, since it is the position within the array that matters rather than the value of an element. One bit data lines are omitted in illustrations, their value is obvious from the direction of data flow.

Plane Rotations

The singular value decomposition of a matrix $A \in R^{m \times n}$ can be represented as

$$A = U \Sigma V^T,$$

where U and V are orthogonal matrices and Σ a nonnegative diagonal matrix. Its diagonal elements, the eigenvalues of $\sqrt{A^T A}$, are called the singular values of A.

In order to avoid an inordinate number of arithmetic operations the matrices to be considered are of bidiagonal form ($i > j$ or $i < j - 1 \Rightarrow a_{ij} = 0$). A stable reduction to this simpler form

$$B = PAQ^T$$

by orthogonal equivalence transformations does not alter the singular values of the original matrix A. The orthogonal matrices P and Q are products of Givens' plane rotations.

The standard Givens' rotation, P, is a computationally stable device for introducing a single zero element into a vector or matrix [Parl80, Wilk65].

$$P = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}, \qquad c^2 + s^2 = 1,$$

$$P \begin{pmatrix} x_1 & x_2 & \dots & x_k \\ y_1 & y_2 & \dots & y_k \end{pmatrix} = \begin{pmatrix} x_1' & x_2' & \dots & x_k' \\ 0 & y_2' & \dots & y_k' \end{pmatrix}, \qquad k \geq 1,$$

where

$$\begin{aligned} &\text{if} \quad y_1 = 0 \quad \text{then} \quad x_1' = x_1, \quad c = 1, \quad s = 0, \\ &\qquad\qquad \text{else} \quad x_1 = \sqrt{x_1^2 + y_1^2}, \quad c = \frac{x_1}{x_1'}, \quad s = \frac{y_1}{x_1'}, \end{aligned} \tag{P1}$$

$$x_i' = cx_i + sy_i, \quad y_i' = -sx_i + cy_i, \qquad 2 \leq i \leq k. \tag{P2}$$

A detailed treatment of plane rotations and their implementation in VLSI is found in [HeIp83, Ipse83]. Time complexity will be measured in terms of plane rotations generated or applied, i.e., (P1) or (P2), each, is assumed to count as one time step.

All of the algorithms to follow (including the singular value computations) rely solely on the reduction of the bandwidth through Givens' Rotations. Because these eliminations proceed codiagonalwise from without toward the diagonal and the elimination strategy for the ensuing fill-in, known as 'chasing the bulge' [Parl80], is similar to one by Rutishauser [Ruti63, Wilk65] the width of the band in any given row at all times is preserved. Consequently, the amount of hardware for the systolic networks will be independent of the matrix dimensions.

Sequential Bandwidth Reductions

Two bandwidth reduction methods, chosen to avoid temporary increase of the width of the band, are presented. The first one removes the outermost sub- and superdiagonal while the second one eliminates k sub- or superdiagonals at the same time. Assume for the time being that matrices are square of order n.

Removal of the outermost codiagonal pair, i.e., the qth subdiagonal and the pth superdiagonal, is accomplished by chasing the bulge from two sides as shown in Algorithm BWO [Ruti63].

Repeat until the matrix A is of order 1 :

1. Generate and apply $P_{q+1,1}$ and $P_{1,p+1}$ to annihilate elements $a_{q+1,1}$ and $a_{1,p+1}$, respectively, causing fill-in $a_{q,w}$ and $a_{w,p}$.
2. Removal of the leading subdiagonal element in step 1 entails rotations $P_{(q+1)+i(w-1),i(w-1)}$ in the upper and $P_{q+(i-1)(w-1),i(w-1)+1}$ in the lower triangle of A, $i \geq 1$, while elimination of the leading superdiagonal element in step 1 entails removal of fill-in by $P_{i(w-1)+1,p+(i-1)(w-1)}$ in the lower and $P_{i(w-1),(p+1)+i(w-1)}$ in the upper triangle of A, $i \geq 1$.
3. Disregard the leading row and column in all subsequent steps.

Algorithm BWO. Removal of the Outermost Codiagonal Pair of a Matrix A.

The sequence of steps 1 to 3 is executed $O(n)$ times as it annihilates one superdiagonal and one subdiagonal element per loop traversal. $O(n^2)$ rotations are generated, each of which takes $O(w)$ steps to be applied. The fill-in created and removed in step 2 occurs in subdiagonal $(q+1)$ and superdiagonal $(p+1)$. After each pass through the loop the matrix order is considered to

reduced by one. If for simplicity $q = p - 1$ is presumed, the bandwidth of the matrix is reduced by two after each removal of a codiagonal pair and the number of time steps amounts to

$$\begin{aligned} T_q &= \sum_{j=1}^{q} (w - 2(j-1)) \sum_{i=1}^{n-q+(j-1)} (n - i + j) \\ &= \frac{1}{2} \sum_{j=1}^{q} (w - 2(j-1))(n - q + j)(n - q + j - 1) \\ &= \frac{1}{2} q n^2 (w+1) - \frac{1}{4} n^2 q^2 + O(nw^3). \end{aligned}$$

For differing p and q, the computation time is proportional to $O(wn^2 \max\{p,q\})$; the dominant factor may double, since the width of the band is reduced $\mid q - p + 1 \mid$ times by one instead of two.

The second method is based on a different order of elimination; $k \geq 1$ codiagonals are removed by pre(post)multiplication, thereby introducing fill-in which is annihilated by post(pre)multiplication. The leading rows and columns now containing zeros in those codiagonals are deflated. Then the process is repeated on the remaining matrix, still having $(q+1)$ elements in its leading column and $(p+1)$ elements in its leading row. The fill-in which arises while eliminating one set of codiagonals is entirely removed before proceeding with the elimination of a further set. This is illustrated in algorithm BWK where one loop traversal causes the annihilation of the $(w-k-1)$ leading elements of each of the k outermost subdiagonals.

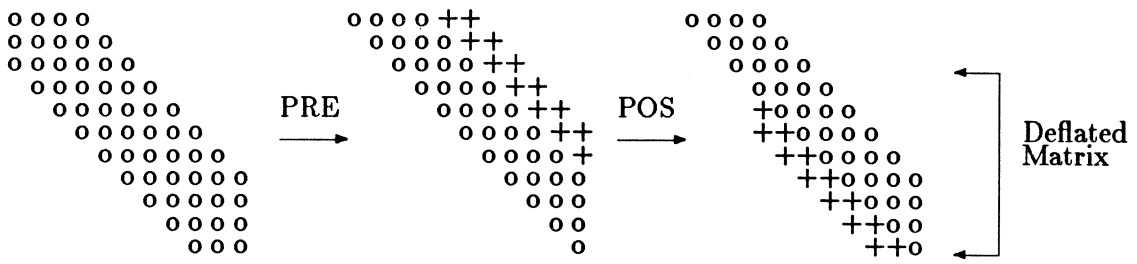
Repeat until the matrix A is of order 1 :

1. Generate Givens' rotations to eliminate by premultiplication the k outermost subdiagonals thereby causing fill-in of k additional superdiagonals.
2. Generate Givens' rotations to remove by postmultiplication this fill-in and thereby restore the previously removed subdiagonals (now containing zeros in their $(w-k-1)$ leading positions).
3. Disregard the $(w-k-1)$ leading rows and columns.

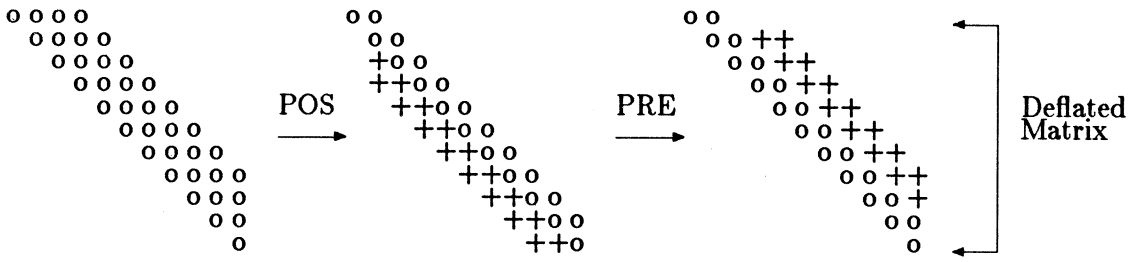
Algorithm BWK. Removal of $k \leq q$ Subdiagonals in a Matrix A.

Figure 1 gives an example of a reduction to bidiagonal form of a matrix with $q = 2$ and $p = 3$; in this case annihilation of subdiagonals takes place before annihilation of superdiagonals.

As will be shown subsequently, elimination of one subdiagonal implies deflation of the $(w-2)$ leading rows and columns in step 3. When starting the removal of the next subdiagonal, the bandwidth is essentially decreased to $w' = w - 1$. As before, the leading $(w'-2)$ rows and columns can be disregarded. Thus, annihilation of k subdiagonals requires deflation of $(w-k-1)$ rows and columns in step 3 of Algorithm BWK.



(a) Removal of $q = 2$ Subdiagonals.



(b) Subsequent Removal of $p - 1 = 2$ Superdiagonals.

+	:	Fill-in
o	:	Remaining matrix elements
PRE	:	Premultiplications
POS	:	Postmultiplications

Figure 1. Reduction to Bidiagonal Form of a Square Matrix, $n = 11$, $p = 3$, $q = 2$.

Consider the annihilation of the outermost, qth, subdiagonal of a matrix A with bandwidth $w = p + q + 1$, for instance. The removal of its ith element $a_{q+i,i}$, $1 \le i \le n - (p+q)$, is effected by

$$P_{q+i,i} \begin{pmatrix} a_{q+i-1,i} & \cdots & a_{q+i-1,p+q+i-1} & 0 \\ a_{q+i,i} & \cdots & a_{q+i,p+q+i-1} & a_{q+i,p+q+i} \end{pmatrix} = \begin{pmatrix} a'_{q+i-1,i} & a'_{q+i-1,i+1} & \cdots & a'_{q+i-1,p+q+i-1} & a'_{q+i-1,p+q+i} \\ 0 & a'_{q+i,i+1} & \cdots & a'_{q+i,p+q+i-1} & a'_{q+i,p+q+i} \end{pmatrix}.$$

The fill-in accumulates in superdiagonal $(p+1)$ as $a'_{q+i-1,p+q+i}$, $1 \le i \le n - (p+q)$. Elimination of $a_{q+i,i}$, where $n - (p+q) + 1 \le i \le n - q$, does not create new nonzero elements. Let $A = A'$. Now remove element $a_{q+i-1,p+q+i}$, $1 \le i \le n - q - (p+q) + 1$, of this new superdiagonal by

$$\begin{pmatrix} a_{q+i-1,p+q+i-1} & a_{q+i-1,p+q+i} \\ a_{q+i,p+q+i-1} & a_{q+i,p+q+i} \\ \vdots & \vdots \\ a_{q+p+q-1+i-1,p+q+i-1} & a_{q+p+q-1+i-1,p+q+i} \\ 0 & a_{q+p+q+i-1,p+q+i} \end{pmatrix} P^T_{q+i-1,p+q+i} = \begin{pmatrix} a'_{q+i-1,p+q+i-1} & 0 \\ a'_{q+i,p+q+i-1} & a'_{q+i,p+q+i} \\ \vdots & \vdots \\ a'_{q+p+q-1+i-1,p+q+i-1} & a'_{q+p+q-1+i-1,p+q+i} \\ a'_{q+p+q+i-1} & a'_{q+p+q+i-1,p+q+i} \end{pmatrix}$$

which at the same time restores the qth subdiagonal with nonzero elements $a_{q+p+q+i-1,p+q+i-1}$, $1 \le i \le n-q-(p+q)+1$. Removal of $a_{q+p+q+i-1,p+q+i-1}$ for $n-q-(p+q)+2 \le i \le n-q$ does not result in the creation of further nonzero elements.

The first nonzero element of subdiagonal q, $\tilde{a}_{q,1}$, occupies position $(q+p+q, p+q)$. The deflated matrix should be of the same shape as the original one, i.e., contain a dense $(q+1) \times (p+1)$ leading submatrix. Moreover, $\tilde{a}_{q,1}$ should now be in row $q+1$ and column 1. Consequently, $(q+p+q)-(q+1) = w-2$ rows and $(p+q)-1 = w-2$ columns must be deflated.

Removal of superdiagonals takes place in an analogous manner. The systolic networks to be presented require the set of k codiagonals to consist of either subdiagonals or superdiagonals, but not both. Furthermore, it seems to be inconsequential whether subdiagonals are deleted first or superdiagonals [Ipse83].

The removal of k codiagonals (subdiagonals or superdiagonals) entails $\lceil \frac{n}{w-k-1} \rceil$ passes through the loop of Algorithm BWK, thus a reduction to bidiagonal form requires

$$\left(\left\lceil \frac{q}{k} \right\rceil + \left\lceil \frac{p-1}{k} \right\rceil \right) \left\lceil \frac{n}{w-k-1} \right\rceil = O\left(\frac{nw}{k(w-k-1)} \right)$$

passes, assuming that k is fixed. To obtain the total number of arithmetic operations in a reduction to bidiagonal form, observe that after each pass the order of the matrix is reduced by $(w-k-1)$, while after $\lceil \frac{n}{w-k-1} \rceil$ passes the bandwidth is decreased by k. Hence the elimination of q subdiagonals takes time

$$\begin{aligned} T_k &= \sum_{j=1}^{q/k} (w-(j-1)k) \sum_{i=1}^{n/(w-1-jk)} n-(i-1)(w-1-jk) \\ &= \frac{1}{2} \sum_{j=1}^{q/k} (w-(j-1)k) \left(\frac{n^2}{w-1-jk} + n \right) \\ &= \frac{1}{2}\frac{q}{k} n^2 + \frac{1}{2}\frac{q}{k} n(w-q) + \frac{1}{2} n^2 (k+1) \sum_{j=1}^{q/k} \frac{1}{w-1-jk}. \end{aligned}$$

With

$$l = \frac{w-1}{k} - j, \quad a = \frac{w-1}{k} - \frac{q}{k}, \quad b = \frac{w-1}{k} - 1,$$

the last term becomes

$$\sum_{j=1}^{q/k} \frac{1}{w-1-k} = \sum_{l=a}^{b} \frac{1}{jk} \le \frac{1}{k}(b-a+1) = \frac{q}{k^2}.$$

Thus,

$$\begin{aligned} T_k &\le \frac{1}{2}\frac{q}{k} n^2 + \frac{1}{2}\frac{q}{k} n(w-q) + \frac{1}{2} n^2 \frac{q(k+1)}{k^2} \\ &= \frac{q}{k} n^2 + \frac{1}{2}\frac{q}{k^2} n^2 + \frac{1}{2}\frac{q}{k} n(w-q), \end{aligned}$$

and

$$\begin{aligned} T_1 &= \frac{3}{2} n^2 q + \frac{1}{2} nq(w-q), \\ T_q &= n^2 + \frac{1}{2} n(w-q) + \frac{1}{2q} n^2. \end{aligned}$$

Consequently, a reduction to bidiagonal form takes $O(n^2w)$ steps for $k = 1$ and $O(n^2)$ for $k = O(w)$.

In general, reduction to bidiagonal form of rectangular matrices $A \in R^{m\times n}$ requires time on the order of $v(\max\{m,n\})$, where $v = w$ or $v = w^2$:

$m \geq n$ Let $A = (A_1 A_2)^T$, where A_1 is a $n \times n$ and A_2 a $l \times n$ matrix, and $l = m - n \geq 0$. Since the only nonzero elements of A_2 constitute subdiagonals, their removal reduces the first dimension, l, of A_2 while elimination of superdiagonals does not affect its order. Hence, $O(vm^2)$ steps are needed, resulting in a $n \times n$ matrix for the singular value decomposition.

$m \leq n$ Proceed as above with the transpose, A^T, rendering a computation time of $O(vn^2)$ as well as a $m \times m$ matrix.

Sequential Singular Value Computations

Computation of singular values with the QR algorithm can be done explicitly and implicitly. The explicit, straightforward, approach for computing the singular values of a matrix $A \in R^{m\times n}$ consists of using the explicitly shifted QR-algorithm to find the eigenvalues of A^TA, as illustrated in Algorithm SVE. In sequential computations the origin shift s_i is an eigenvalue of the trailing principal 2×2 submatrix of $A_i^TA_i$.

---

1. $B_i = A_i^TA_i \qquad (A_0 = A)$.
2. Compute $B_i - s_iI = Q_iR_i$.
3. Determine $B_{i+1}^T = Q_i^TR_i^T + s_iI$.

Algorithm SVE. One Step of the Explicitly Shifted QR-Algorithm in the SVD Decomposition of a Matrix A.

Orthogonal equivalence transformations allow both, a reduction of A to bidiagonal form before commencing SVE or else a reduction of B_0 to tridiagonal form after step 1 of SVE. Let $A \in R^{m\times n}$ have bandwidth w, $l = \min\{m,n\}$ and $L = \max\{m,n\}$. As for the first possibility, the bandwidth reduction yields a bidiagonal matrix $A' \in R^{l\times l}$ in time $O(w^2L^2)$. Steps 1,2 and 3 deal with matrices of bandwidth 2 or 3 in time $O(l)$. In the second case, matrix multiplication brings about an $n \times n$ matrix B' of bandwidth $2w - 1$ in time $O(w^2L)$, followed by a reduction to tridiagonal form of B' in time $O(w^2n^2)$. Steps 2 and 3 need $O(n)$ operations. Unless $l = n$, where the second alternative requires less time, the first one is to be preferred. A delayed reduction doubles the bandwidth, and hence the hardware, as well as the number of passes through the loop in Algorithms BWO and BWK.

Information about the smaller singular values of A may be lost when restricting attention to the rounded A^TA. The following method avoids the explicit formation of A^TA.

The Golub-Reinsch SVD iteration applies a variant of the implicitly shifted QR algorithm to a bidiagonal matrix $A \in R^{n\times n}$ [GoRe71] (since an upper bidiagonal matrix is zero below the nth row, the matrix is assumed to be square).
Find P_i, a Givens' rotation, so that

$$(P_i(A_i^TA_i - s_iI))_{21} = 0.$$

Set

$$B_i = A_iP_i^T$$

and compute A_{i+1} by reducing B_i to bidiagonal form via orthogonal equivalence transformations, requires $O(n)$ arithmetic operations. The algorithm is exhibited in detail as Algorithm SVI below. If

$$A_i^TA_i = Q_iR_i,$$

then P_i^T has the same first column as Q_i, hence B_i differs from A_i only in the first two rows and columns. Accelerated convergence, associated with origin shifts, is achieved without having to subtract the shift from the diagonal and later restore it (as in the explicitly shifted version) but by means of concentrating the effect of the shift s_i in the matrix P_i.

---

1. Find P_i so that $(P_i(A_i^T A_i - s_i I))_{21} = 0$.
2. Determine $B_i = A_i P_i^T$.
3. $\overline{B}_2 = B_i$.
4. For $j = 2, 3, \ldots, n-1$ do
   begin
   4.1 Generate a premultiplication $P_{j,j-1}$ to annihilate the element of $\overline{B}_j$ in position $(j, j-1)$.
   4.2 Apply $P_{j,j-1}$ and generate a fill-in at position $(j-1, j+1)$ of $P_{j,j-1}\overline{B}_j$.
   4.3 Generate a postmultiplication $P^T_{j-1,j+1}$ to eliminate the element in position $(j-1, j+1)$ and compute $\overline{B}_{j+1} = (P_{j,j-1}\overline{B}_j)P^T_{j-1,j+1}$, while causing a fill-in at position $(j+1, j)$ of $\overline{B}_{j+1}$.
   end.
5. $A_{i+1} = \overline{B}_n$.

Algorithm SVI. One Step of the Golub-Reinsch SVD Iteration for Bidiagonal Matrices A.

When given a lower bidiagonal matrix A its transpose A^T will be considered, as the singular values of A and A^T are the same. For a diagonal matrix A only identity rotations are formed.

Roundoff Error in Algorithms BWO and BWK

Not only the time complexity but also a roundoff error analysis indicates that BWK is to be preferred over BWO, since fewer rotations are generated and hence higher accuracy is attained.

Gentleman [Gent75] has shown that the application of plane rotations to a matrix $A \in R^{n \times n}$ has a roundoff error bounded above by

$$\eta s \parallel A \parallel_F (1+\eta)^{s-1},$$

where η is a multiple of the machine error and s is the maximal number of rotations applied to any element. The following observations determine s for BWK and BWO.

BWK : In step 1 of BWK each row is affected by at most $2k$ different premultiplications (k to eliminate elements in that very row and k for the row beneath). In step 2 an element of a row is affected by at most $2k$ postmultiplications, giving a total of $4k$ rotations per element per loop traversal. There are $\frac{n}{w-1-k}$ traversals, so

$$s_{BWK} = \frac{4kn}{w-1-k}.$$

BWO : The larger the bandwidth the smaller the percentage of elements which contribute to the rounding error. Hence, the worst case (generation of rotations in adjacent rows or columns) occurs in BWO when reducing a matrix with $q = 1$ and $p = 2$ to bidiagonal form. By an argument similar to the previous one, each element is altered by at most 4 rotations. To eliminate a codiagonal pair $n-1$ loop traversals are necessary resulting in

$$s_{BWO} = 4(n-1).$$

For matters of comparison assume that k codiagonals are removed : BWO annihilates $k/2$ subdiagonals and $k/2$ superdiagonals, while BWK deletes k subdiagonals or k superdiagonals. BWO renders higher accuracy than BWK only if

$$2k(n-1) = \frac{k}{2} s_{BWO} < s_{BWK} = \frac{4kn}{w-1-k},$$

which is the case for

$$w - 3 - \frac{2}{n-1} < k,$$

namely a reduction to bidiagonal form,

$$s_{BWO} = s_{BWK} - 2(w-2)(n+1),$$

or a transformation to tridiagonal form

$$s_{BWO} = s_{BWK} - 2(w-3).$$

The above comparison presumes BWK to remove all codiagonals at once implying that A has $q = w-2$ or $p = w-1$. In all but two situations BWK is executed separately for subdiagonals and superdiagonals and thus more accurate than BWO.

Processors

The processors employed are simple extensions of cells 1, 1', 2, 2' and 3 in [Help83, Ipse83] with the same data flow. Processor B1 generates and/or applies rotations while B2 only applies rotations. Cell B is a conglomerate of two B1 and two B2 cells, whereas S is needed to direct (shift) data to appropriate processors. In addition, each processor is augmented with two one-bit data lines which determine the direction of dataflow elemnts

$left = 1$ shifting of matrix elements to the left for subdiagonal elimination (premultiplications).

$right = 1$ shifting of matrix elements to the right for superdiagonal elimination (postmultiplications). Processors B1 and B are additionally attached to a third bit line, *on*. The three bit lines determine the task performed by a processor.

In particular, cell B1 generates rotations if $on = 1$ (behaving like cell 1 or 1') and applies them otherwise (cell 2 or 2'). If $left = 1$ then the rotations generated are premultiplications (as in cell 1, or 2) and postmultiplications if $right = 1$ (cell 1', or 2'), see Figure 2. Cell B2 propagates rotations to the right and matrix elements to the left if $left = 1$ (like cell 2) and if $right = 1$ rotations to the left and elements to the right, shown in Figure 3. Figure 4 depicts cell S which directs matrix elements straight upwards with a delay of one step if $on = 0$ (i.e., it functions as cell 3 with identity rotations [Help83]). If $on = 1$, it displaces them by one cell to the left if $left = 1$ (similar to cell 2 with identity rotations [Help83]) and to the right if $right = 1$ (cell 2' with identity rotation). Figures 5, 6 and 7 depict I/O format and data flow through linear meshes for subdiagonal elimination ($left = 1$), superdiagonal elimination ($right = 1$) and shifting upwards ($left = right = 0$). The former are identical to the ones in [Help83].

Cell B is crucial for the correct execution of algorithms BWO and SVI, since it assures consistent application of rotations as follows. The reduction of a matrix A by pre- and postmultiplications results in a matrix B, where

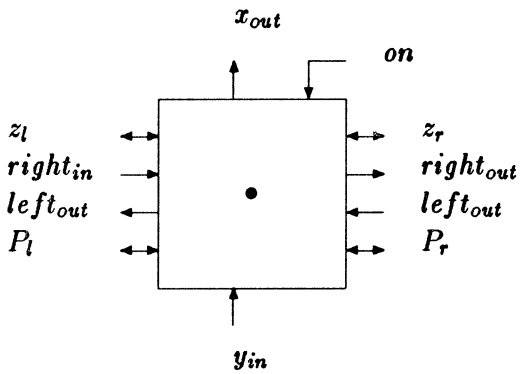
$$B = \left(\prod_{i,j} P_{i,j} \right) A \left(\prod_{i,j} Q_{i,j} \right),$$

and $P_{i,j}$ and $Q_{i,j}$ are Givens' rotations. Parallelism is exploited by taking advantage of associativity in matrix multiplication : it is immaterial whether premultiplications or postmultiplications are performed first, however

1. the order of premultiplications in $\left(\prod_{i,j} P_{i,j}\right)$ and postmultiplications in $\left(\prod_{i,j} Q_{i,j}\right)$ must be preserved,
2. if $P_{i,j}$ and $Q_{k,l}$ are applied at the same time, they must not affect common elements; when acting on the same elements (at different times), one must be completed before the other commences.

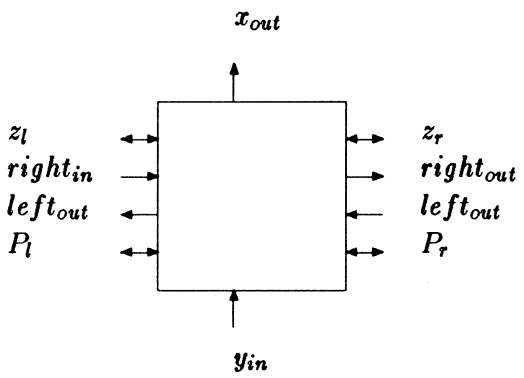
In order to ascertain condition 2, B handles application of a rotation to three different element pairs at the same time and thus enables premultiplications and postmultiplications to take place in succeeding time intervals; so fill-in is generated in one before being removed in the next step. It joins two linear meshes and forwards postmultiplications from top to bottom and premultiplications from bottom to top. The signals on_{PRE} and on_{POS} determine whether rotations are to be generated and if so, of which type they are.

For ease of understanding (not necessarily conforming to physical reality) the data flow 'inside' cell B will be depicted as similar to one in a combination of cells B1 and B2. Figure 8 presents the programme of cell B along with an illustration of the element pairs that are affected. Diagonal elements are input to the lower left corner and elements of the first superdiagonal to the lower right. The neighbouring processor to the lower left delivers elements of the first subdiagonal and the one to the lower right (if present) elements of the second superdiagonal. Five time steps are necessary for a particular value to 'traverse' the processor. Because a diagonal element is the first element of a matrix to enter the array under the current I/O format, a postmultiplication (POS) takes place before a premultiplication (PRE), either of which may be identity.



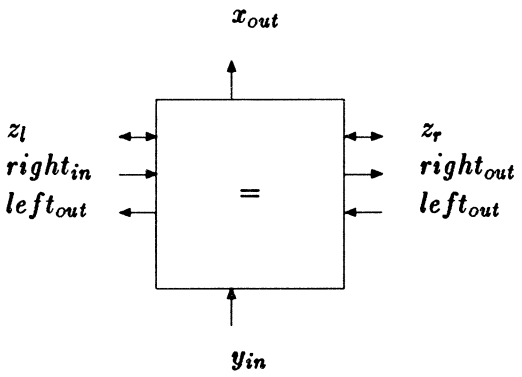
{Internal registers are P, x and y}
$right_{out} := right_{in}$;
$left_{out} := left_{in}$;
$y := y_{in}$;
if $on = 1$ **then**
if $left_{in} = 1$ **then**
begin {Generate premultiplications}
$x := z_r$; {Input}
$\binom{x}{0} := P\binom{x}{y}$; {Equations (P1)}
$P_r := P$; $z_l := 0$; {Output}
end
else $\{right_{in} = 1\}$
begin {Generate postmultiplications}
$x := z_l$; {Input}
$(x\ 0) := (x\ y)P^T$; {Equations (P1)}
$P_l := P$; $z_r := 0$; {Output}
end
else $\{on = 0\}$
if $left_{in} = 1$ **then**
begin {Propagate premultiplications}
$P := P_l$; $x := z_r$; {Input}
$\binom{x}{y} := P\binom{x}{y}$; {Equations (P2)}
$P_r := P$; $z_l := y$; {Output}
end
else $\{right_{in} = 1\}$
begin {Propagate postmultiplications}
$P := P_r$; $x := z_l$; {Input}
$(x\ y) := (x\ y)P^T$; {Equations (P2)}
$P_l := P$; $z_r := y$; {Output}
end
$x_{out} := x$.

Figure 2. Cell B1.



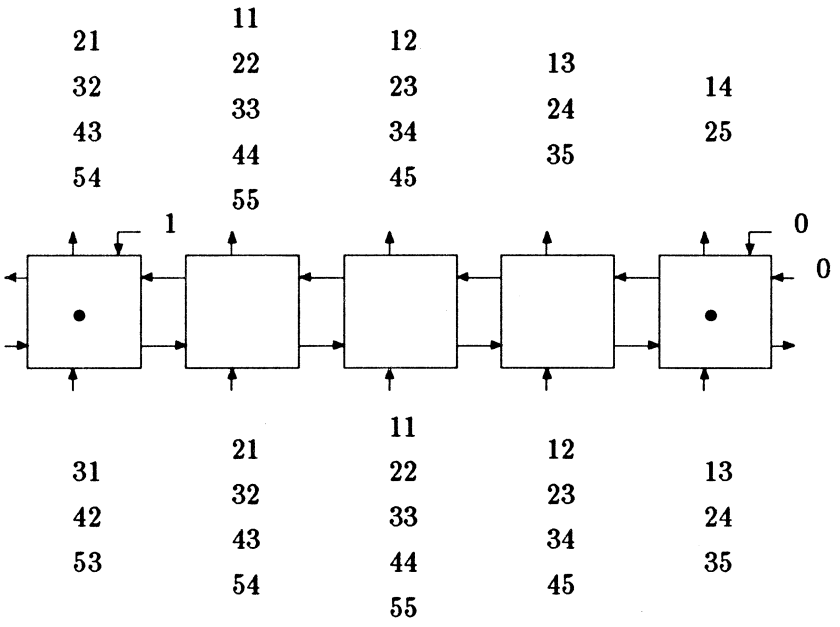
{Internal registers are P, x and y}
$right_{out} := right_{in}$;
$left_{out} := left_{in}$;
$y := y_{in}$;
if $left_{in} = 1$ **then**
begin {Propagate premultiplications}
$P := P_l$; $x := z_r$; {Input}
$\binom{x}{y} := P\binom{x}{y}$; {Equations (P2)}
$P_r := P$; $z_l := y$; {Output}
end
else $\{right_{in} = 1\}$
begin {Propagate postmultiplications}
$P := P_r$; $x := z_l$; {Input}
$(x\ y) := (x\ y)P^T$; {Equations (P2)}
$P_l := P$; $z_r := y$; {Output}
end
$x_{out} := x$.

Figure 3. Cell B2.



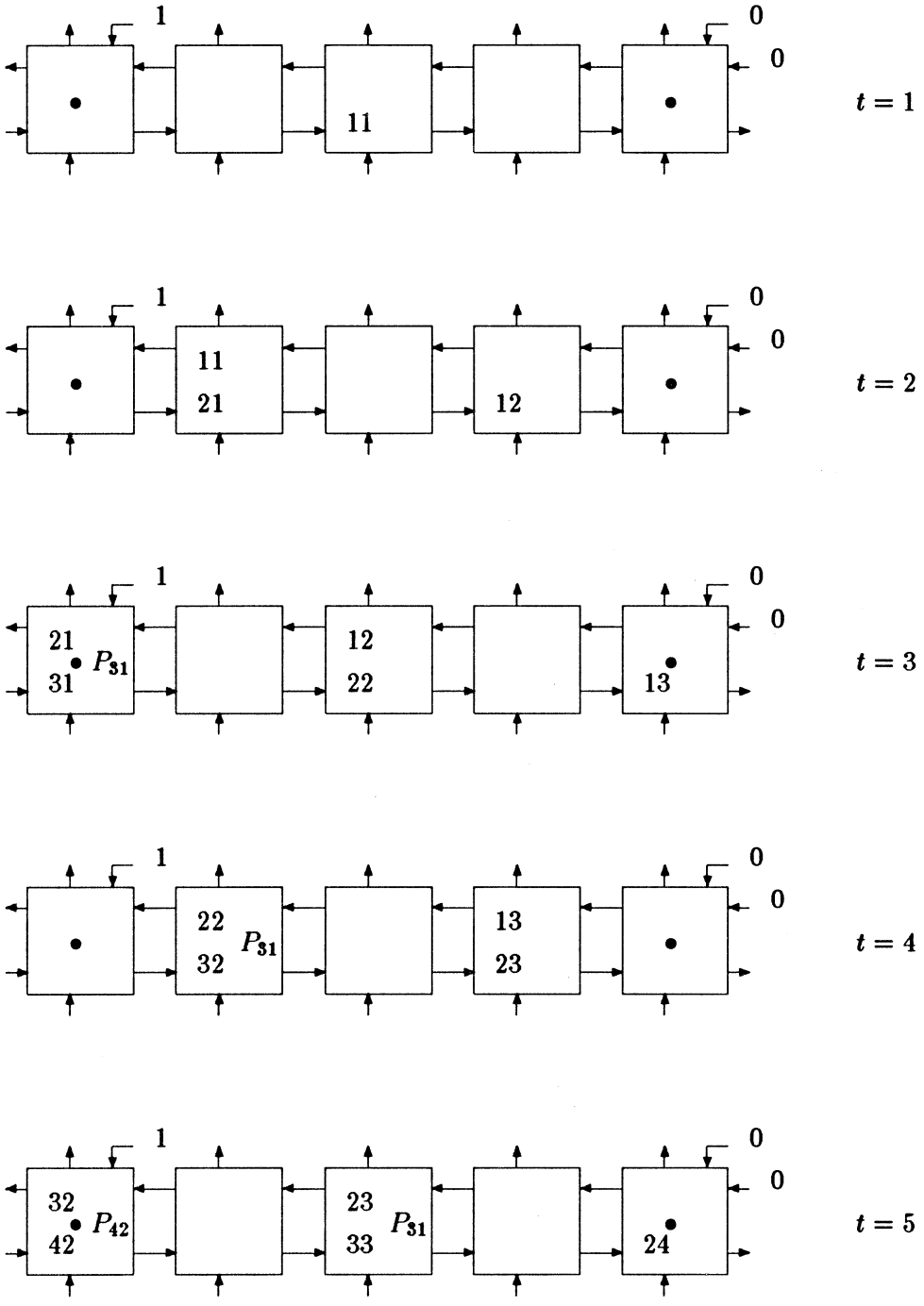

```
{Internal registers are x and y}
right_out := right_in;
left_out := left_in
if left_in = 1 then
                begin {Shift left}
                      x_out := z_r; z_l := y_in;
                end
           else if right_in = 1 then
                                begin {Shift right}
                                      x_out := z_l; z_r := y_in;
                                end
                           else {right_in = left_in = 0}
                                begin {Shift upwards}
                                      x_out := x; x := y; y := y_in;
                                end
```

Figure 4. Cell S.



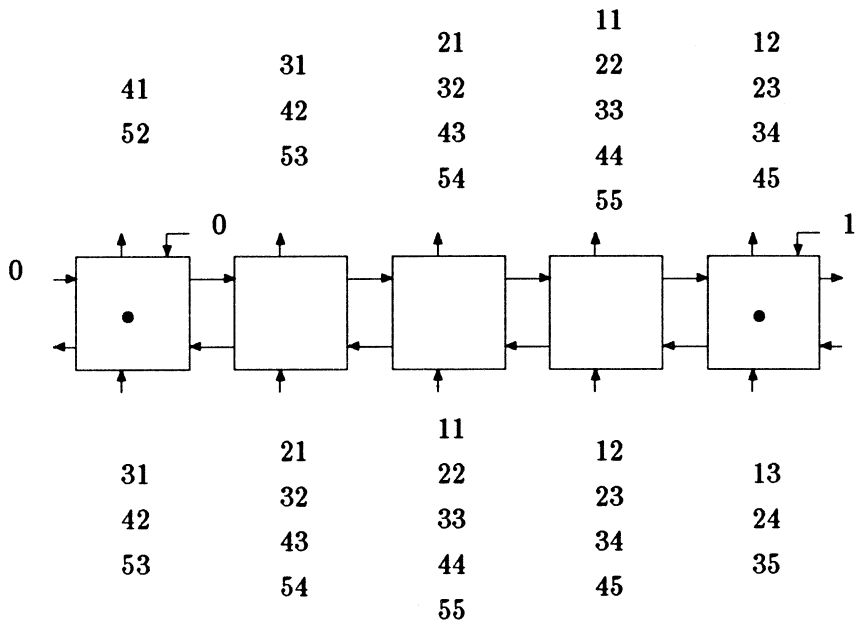
(a) Input/Output Format.

Figure 5. Linearly Connected Mesh for Subdiagonal Elimination ($left = 1$, $right = 0$, $\bar{w} = 5$) and Input $p = q = 2$.



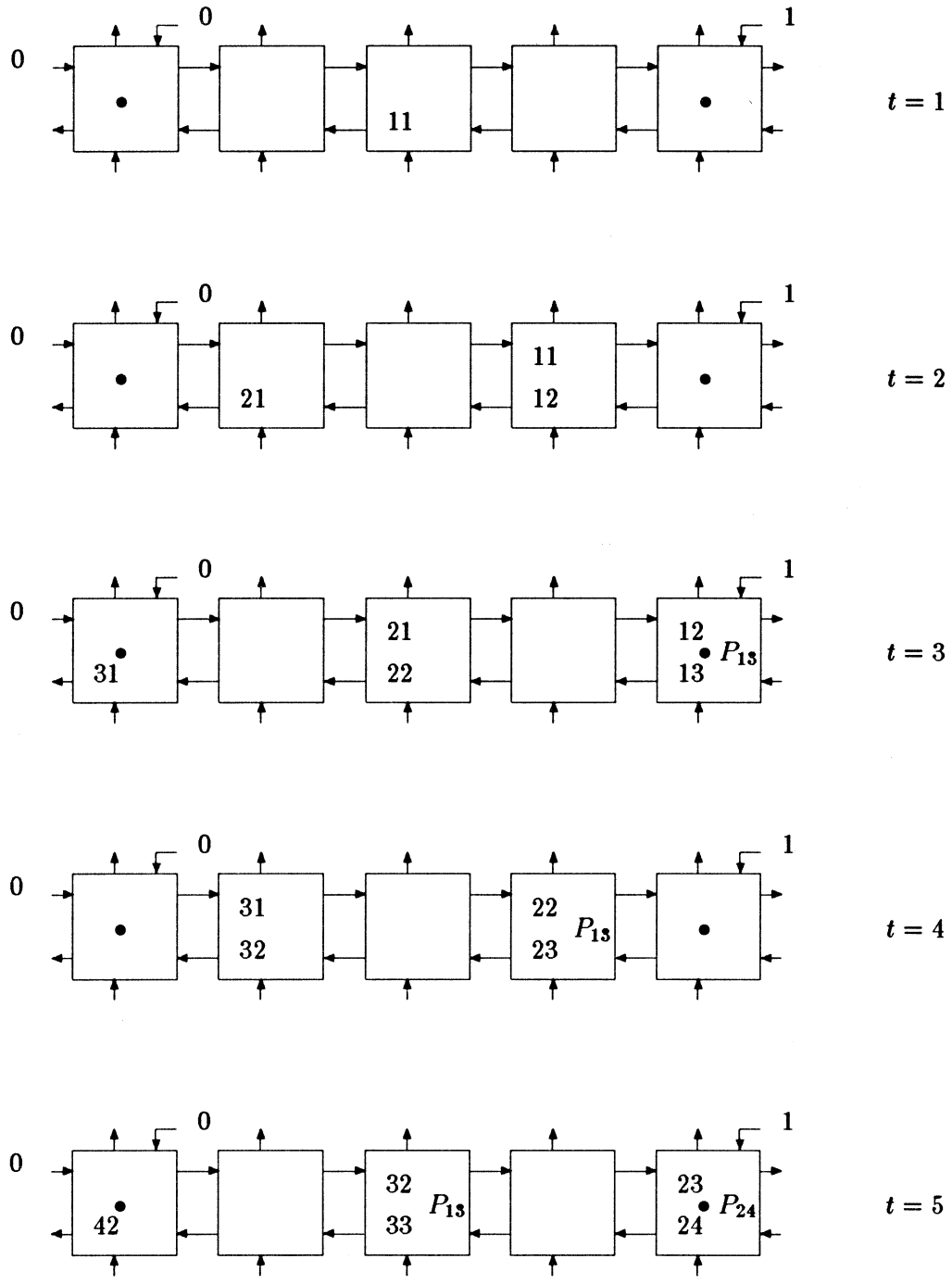
(b) Partial Execution Trace.

Figure 5. Continuation.



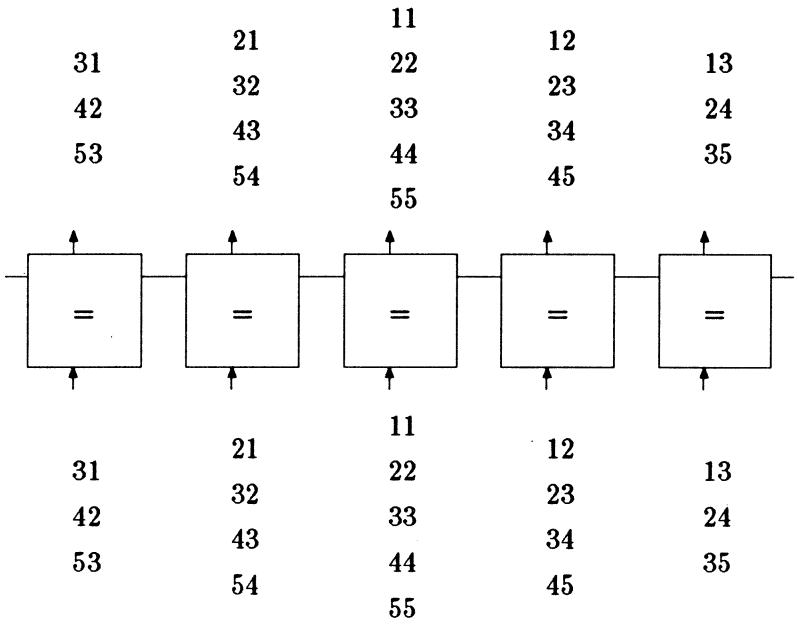
(a) Input/Output Format.

Figure 6. Linearly Connected Mesh for Superdiagonal Elimination ($left = 0$, $right = 1$, $\bar{w} = 5$) and Input $p = q = 2$.



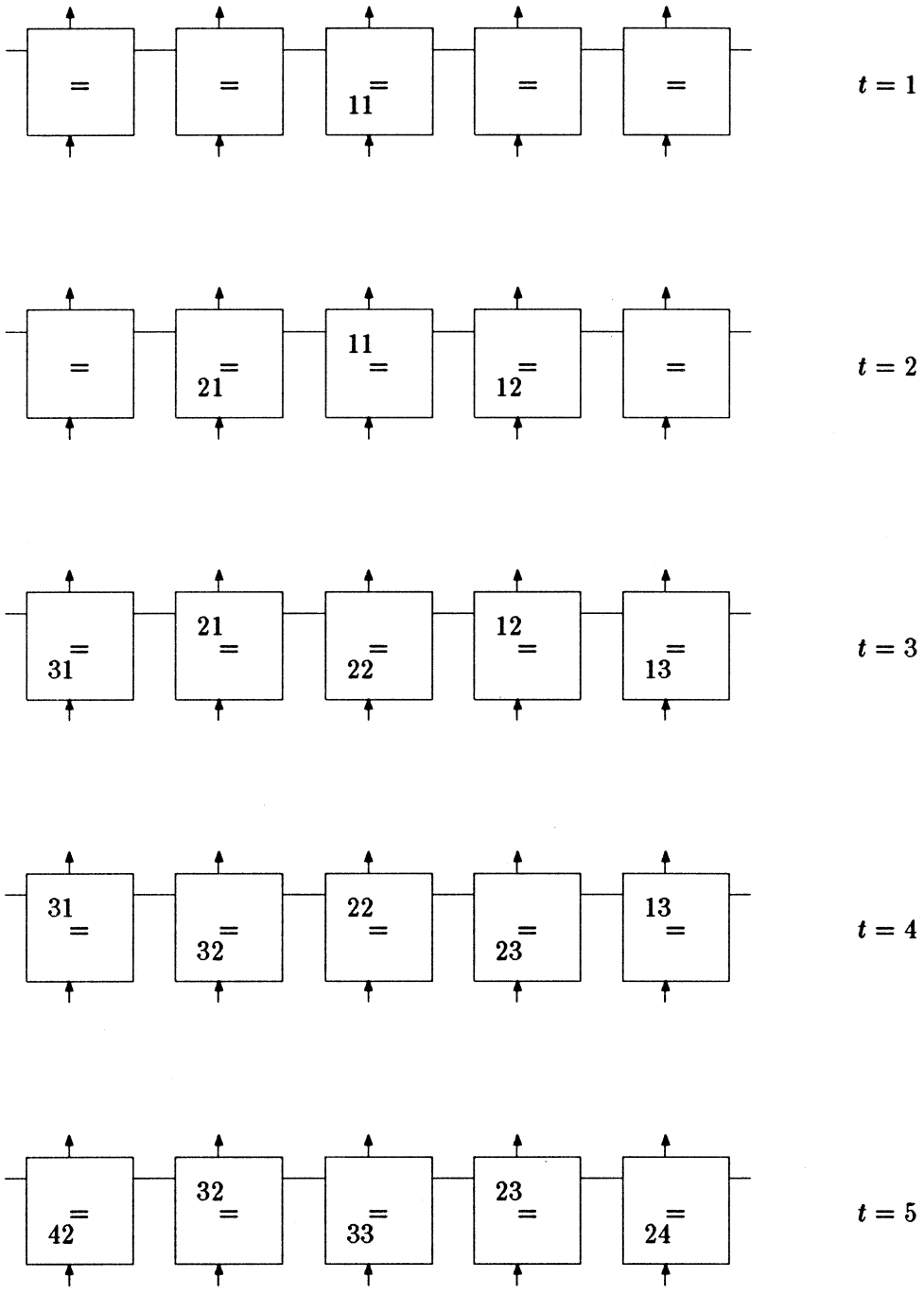
(b) Partial Execution Trace.

Figure 6. Continuation.



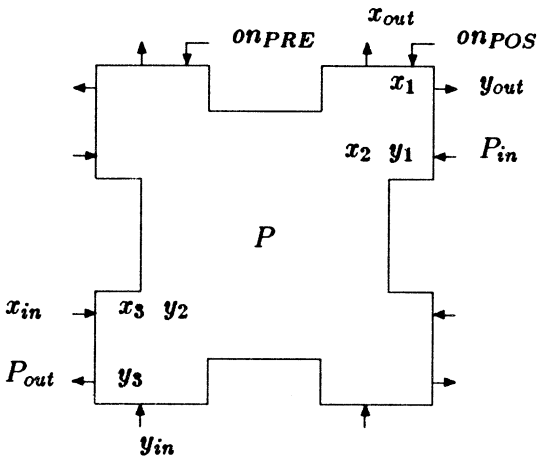
(a) Input/Output Format.

Figure 7. Linearly Connected Mesh for Shifting ($left = right = 0$, $\bar{w} = 5$) and Input $p = q = 2$.



(b) Partial Execution Trace.

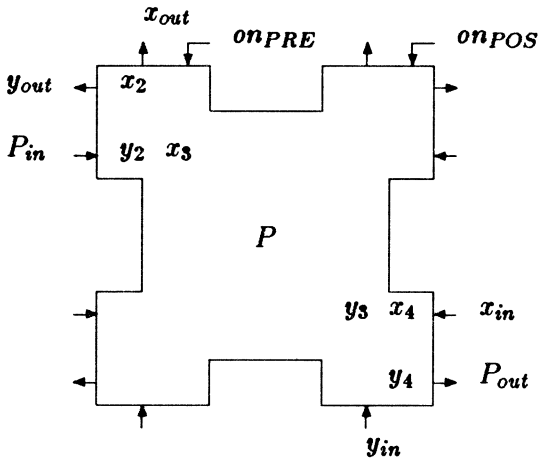
Figure 7. Continuation.



{Internal registers are P, x_i and y_i, $1 \leq i \leq 3$ }
$x_3 := x_{in}$; $y_3 := y_{in}$;
if $on_{POS} = 1$ **then** {Generate rotations}
$(x_1\ 0) := (x_1\ y_1)P^T$;
else {Input rotations}
$P := P_{in}$;
$\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} := \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} P^T$;
$x_{out} := x_1$; $y_{out} := y_1$; $P_{out} := P$; {Output}
$\begin{pmatrix} x_2 & y_2 \\ x_3 & y_3 \end{pmatrix} := \begin{pmatrix} x_2 & x_3 \\ y_2 & y_3 \end{pmatrix}$. {Transpose}

(a) Postmultiplication (POS).

Figure 8. Cell B.



{Internal registers are P, x_i and y_i, $1 \le i \le 4$}
$x_4 := x_{in}$; $y_4 := y_{in}$;
if $on_{PRE} = 1$ **then** {Generate rotations}
$\binom{x_2}{0} := P\binom{x_2}{y_2}$;
else {Input rotations}
$P := P_{in}$;
$\begin{pmatrix} x_2 & x_3 & x_4 \\ y_2 & y_3 & y_4 \end{pmatrix} := P\begin{pmatrix} x_2 & x_3 & x_4 \\ y_2 & y_3 & y_4 \end{pmatrix}$;
$x_{out} := x_2$; $y_{out} := y_2$; $P_{out} := P$; {Output}
$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} := \begin{pmatrix} x_3 & x_4 \\ y_3 & y_4 \end{pmatrix}$. {Transpose and rename}

(b) Premultiplication (PRE).

Figure 8. Continuation.

A Systolic Array for Algorithm BWO

The concurrent order for the generation of plane rotations is described in [SaKu78, HeIp83]. Consider subdiagonal annihilation : after the ith element of the $q - j + 1$st subdiagonal has been removed, rows $1 \ldots q + j + i - 2$ will not be affected by further eliminations in this subdiagonal and removal of the ith element of subdiagonal $q - j$ can commence, $1 \le j \le k$, $1 \le i \le n - q - i + 1$. Annihilation of subdiagonal elements in corresponding positions is staggered so as to avoid interference of rotations. The analogue holds for superdiagonal eliminations.

In compliance with Algorithm BWO the array removes the leading elements of the outermost sub- and superdiagonal along with the ensuing fill-in. It consists of two linear meshes, joined through cell B (which in view of area and complexity accounts for four cells), bringing the total number of processors to $2w - 3$. The $w - 1$ processors in the bottom mesh are B2 cells. The ones to the left of cell B have $right = 1$ and to the right $left = 1$. When $q \ge 2$ and $p \ge 3$ the two leftmost and rightmost cells in the upper mesh are B1 cells, all others B2 and $on_{pre} = on_{post} = 0$ for cell B (rotations are generated by the B1 processors not B). Cells to the left of B have $left = 1$ and to the right of B $right = 1$. The inner B1 cells remove the leading nonzero element of an outermost codiagonal (subdiagonal q or superdiagonal p) by generating an appropriate rotation ($on = 1$). For all subsequent elements they forward rotations ($on = 0$), generated by the outer B1 cells, to chase the bulge from subdiagonal $q + 1$ or superdiagonal $p + 1$. Figure 9 presents a bandwidth reduction array with I/O format for $\bar{q} = 4$ and $\bar{w} = 8$ while Figure 10 displays the data

flow for the example $\bar{q} = 2$ and $\bar{w} = 6$. The cases $q < 2$ or $p < 3$ are treated with an array similar to the one used for the singular value decomposition, where B generates rotations and some of the B1 cells are dispensable.

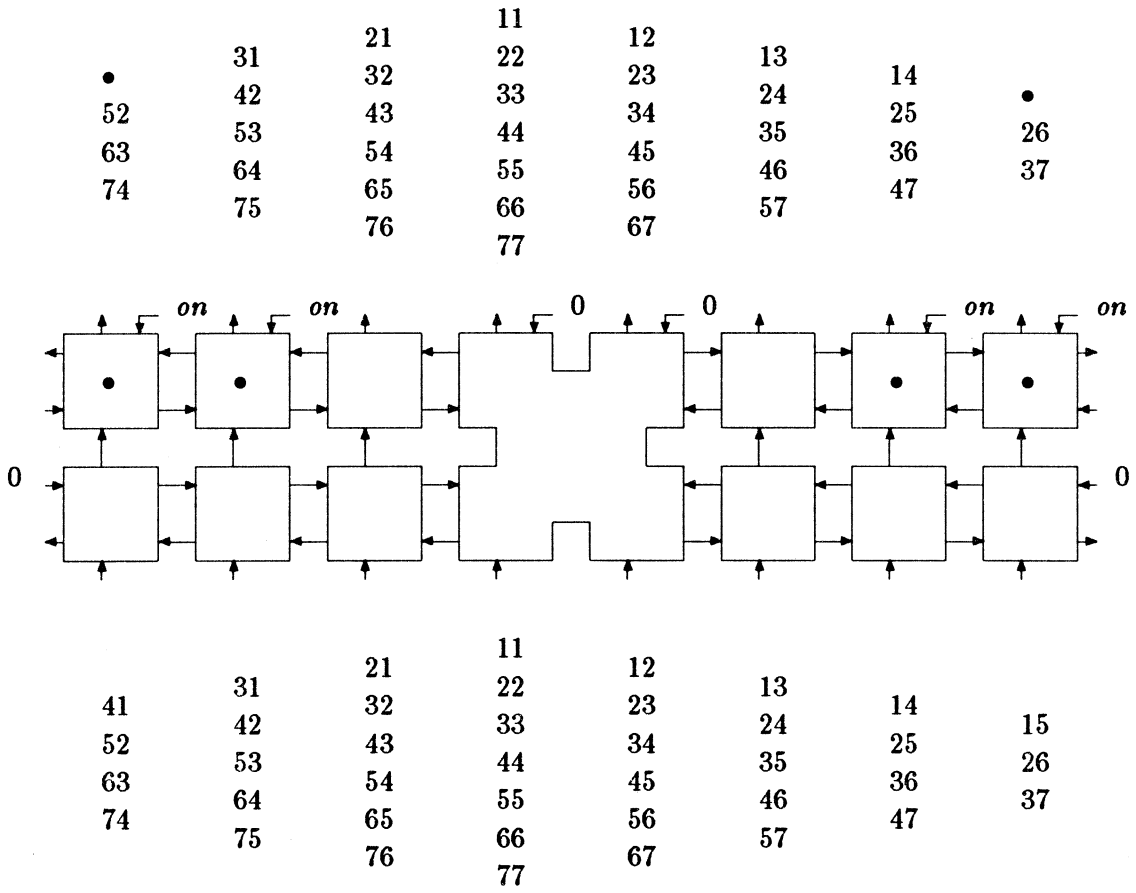


Figure 9. Bandwidth Reduction Array for $\bar{w} = 8$, $\bar{q} = 4$, $\bar{p} = 5$.

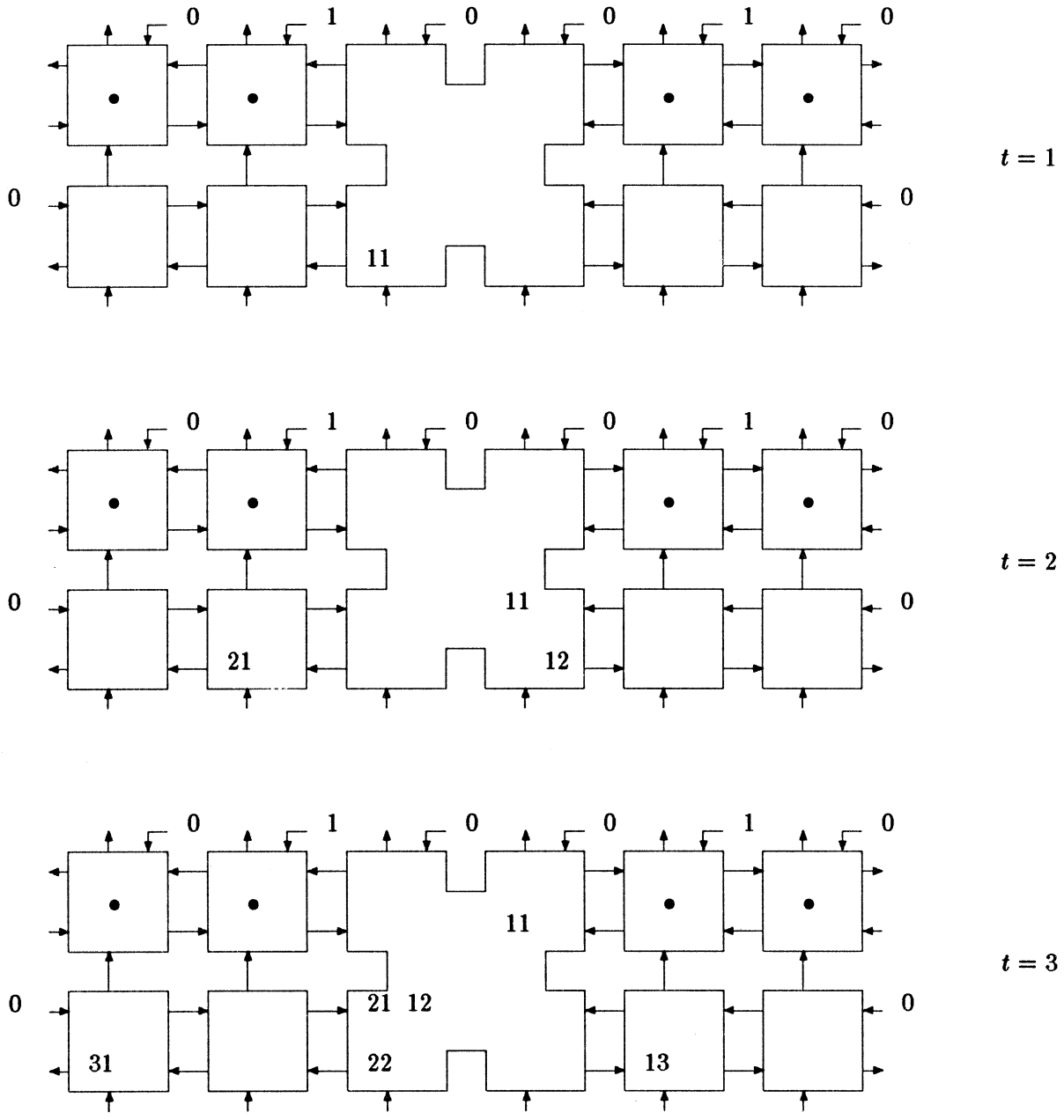


Figure 10. Partial Execution Trace for a Bandwidth Reduction Array with $\bar{q} = 2$ and $\bar{p} = 3$.

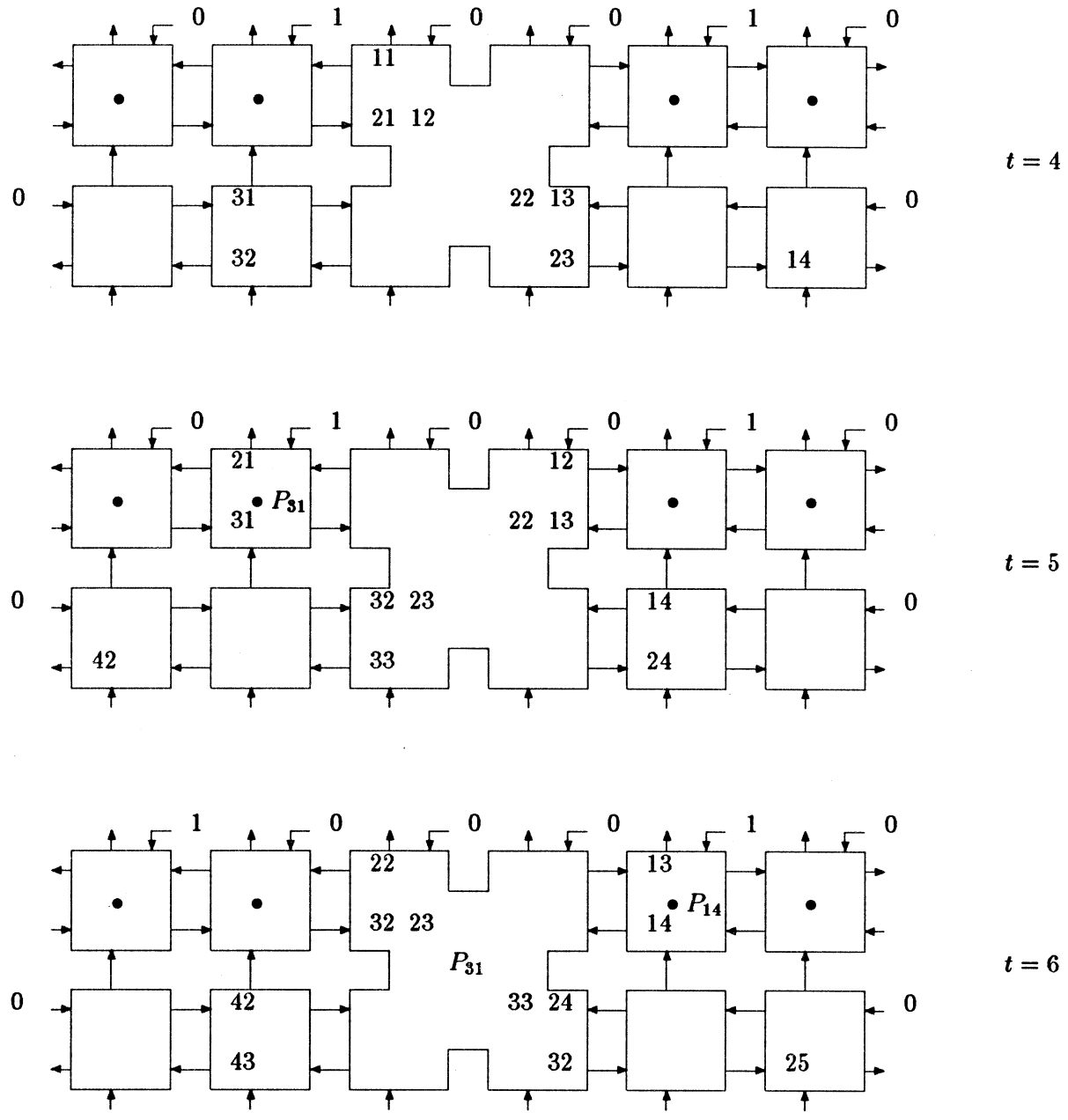


Figure 10. Continuation.

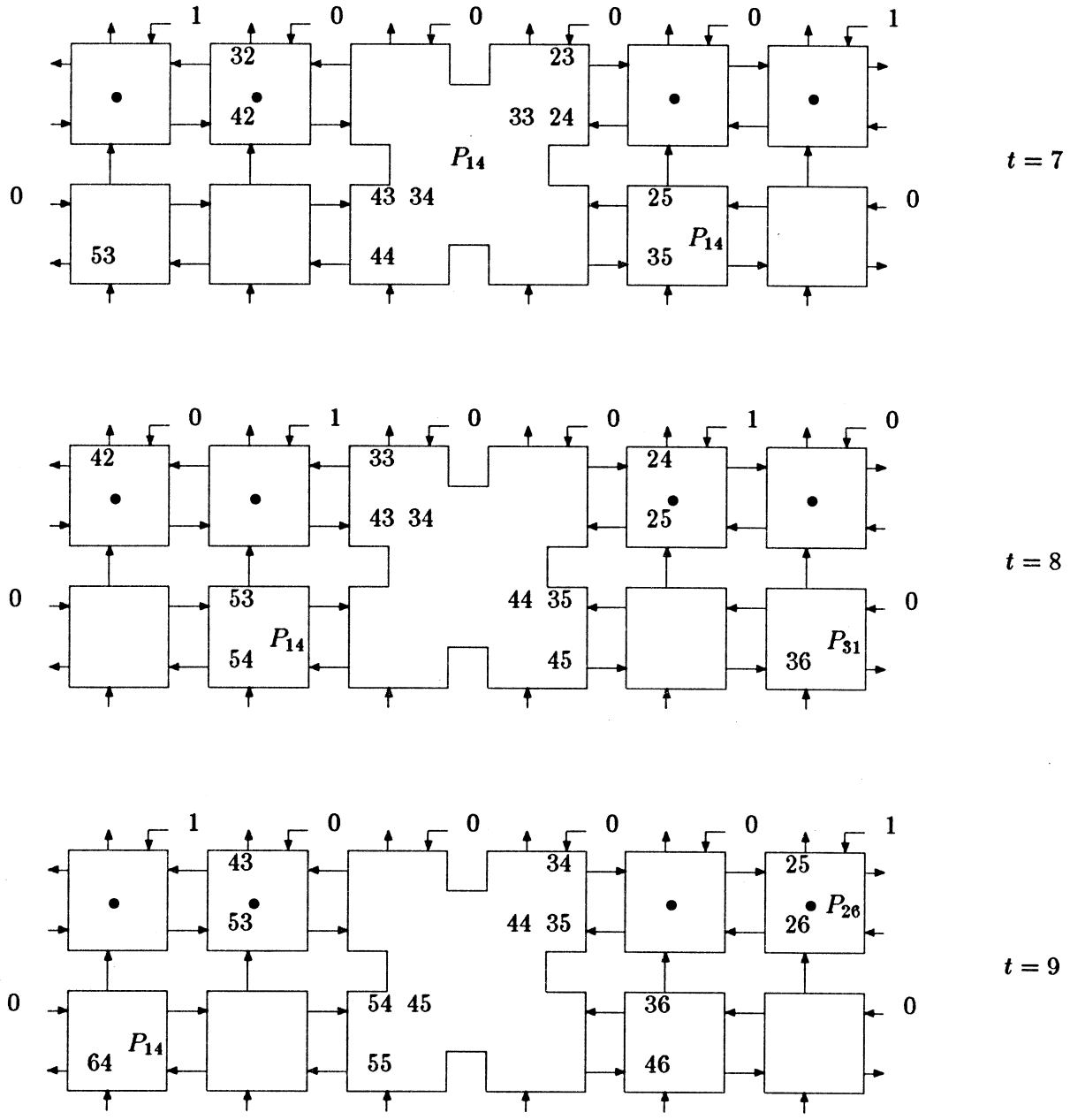


Figure 10. Continuation.

A traversal through the array is done in

$$T(n,2) = 2n + 3$$

steps. If the leading row and column is deflated after each pass through the array, the number of steps for removal of the qth subdiagonal and pth superdiagonal comes to

$$\begin{aligned}\sum_{i=1}^{n-r} T(n-i+1,2) &= \sum_{i=1}^{n-r} 2(n-i+3) - 1 \\ &= (n-r)(2n+5) - (n-r)(n-r+1) \\ &= n^2 - r^2 + O(n),\end{aligned}$$

where $r = \min\{p,q\}$. As for the generation of rotations, the left inner B1 cell has $on = 1$ at $t = 1 \ldots q+2$ and $on = 0$ thereafter, when $on = 1$ for the outer B1 cell. Similarly, the right outer B1 cell is set to $on = 1$ at $t = 1 \ldots p+2$, after which on becomes 0 and the outer B1 processor generates rotations.

If $\bar{k} \leq n - r$ such arrays are chained and for simplicity the $\bar{k}$ leading rows and columns are deflated after each pass through the $\bar{k}$ arrays, the removal of a codiagonal pair takes time

$$\begin{aligned}\sum_{i=1}^{(n-r)/\bar{k}} T(n-(i-1)\bar{k}, 2\bar{k}) &= \sum_{i=1}^{(n-r)/\bar{k}} 2(n - i\bar{k} + 3\bar{k}) - 1 \\ &= 2\frac{n(n-r)}{\bar{k}} + \frac{(n-r)^2}{\bar{k}} + O(n) = O\left(\frac{n^2}{\bar{k}}\right).\end{aligned}$$

The setting of bit lines now also depends on the position of the array in the $\bar{k}$ array network.

Only if the number of arrays, $\bar{k}$, is proportional to the order of the matrix, is the resulting computation time linear in n. A reduction to bidiagonal form takes $O\left(\frac{wn^2}{\bar{k}}\right)$ steps. For small $\bar{k}$ this represents a reduction in execution time of order w as compared to the sequential case.

The last (reduction from $w = 3$ or 4 to $w = 2$) in a stage of bandwidth reduction arrays is nearly identical to the array implementing Algorithm SVI. The resulting agreement in hardware features and execution speeds of both arrays greatly facilitates their chaining. Unfortunately, within a bandwidth array the complexity of different cell types varies greatly and so do their execution times. If t_x is the time for cell x to complete one step, then

$$t_{B2} \leq t_{B1} \leq t_B.$$

As opposed to the SVI array which basically consists of one processor that generates rotations in each step, $2w - 4$ simpler processors work at the speed of one big, complex cell. Moreover, on account of the large area occupied by cells B1 and B, the ratio of bandwidth to area becomes only appreciable for large bandwidths. Lastly, on account of the fixed positions for those cells not only $w = \bar{w}$ is necessary but also $p = \bar{p}$ and $q = \bar{q}$. Thus, $\max\{p,q\}$ differently sized arrays must be available in order to accomodate the decreasing bandwidths.

Linear Meshes for Algorithm BWK

The design for algorithm BWK requires simpler hardware, already introduced in [Help83], than that for BWO and allows processing of arbitrarily sized matrices. Two types of linear meshes, each comprising $\bar{w}$ cells, are employed; one performs actual computations, i.e., elimination of codiagonals, and a second one properly positions (shifts) matrix elements.

A computation mesh consists of $(\bar{w}-2)$ B1 and/or B2 cells, enclosed by a B1 cell to the right and to the left. Each mesh contains at most one B1 cell with $on = 1$. The computation mesh, an extension of the one in [Help83] and obtained by replacing cells 1 and 1' by B1 and 2 and 2' by B2, can perform both, pre- and postmultiplications; which of the two, is determined by the value of the bits $left$ and $right$, and on in the B1 cells.

If $left = 1$ a mesh performs premultiplications (Figure 5) and is called a QR mesh (q such meshes combined effect a QR decomposition). If the matrix elements are routed so that the outermost subdiagonal enters a B1 cell with $on = 1$, this subdiagonal is eliminated, the resulting rotation propagated to the left and thus a new superdiagonal (fill-in) generated. Because the bandwidth is preserved, the remaining elements are displaced one cell to the left before output from the mesh. Identity rotations are generated for a zero outermost subdiagonal. A B1 cell with $on = 1$ must be among the $\bar{w}-(w-1)$ leftmost cells of the mesh to provide for enough space to the right for the other $\bar{w}-1$ codiagonals. If all B1 cells have $on = 0$, nonidentity rotations may be input to the leftmost B1 cell, which applies and then forwards them as before.

If $right = 1$, the mesh performs postmultiplications (Figure 6) and is called QL mesh (p of those accomplish a QL factorisation). The outermost superdiagonal is annihilated when entering a B1 cell with $on = 1$,the resulting rotations forwarded to the left and another subdiagonal is formed. Again, preservation of bandwidth dictates that the remaining elements be displaced one cell to the right before leaving the mesh. A zero superdiagonal causes formation of identity rotations. To accomodate the $(\bar{w}-1)$ codiagonals to the left of it, a B1 cell with $on = 1$ must be among the rightmost $\bar{w}-(w-1)$ cells of the mesh.

The cases $left = right = 1$ or $left = right = 0$ are not allowed to occur. As a matrix elements is displaced to either the left or right before exiting the array, it needs three time steps to traverse the mesh.

A linear shift mesh consists of S cells (Figure 7). If $left = 1$ matrix elements are shifted one cell to the left (as in a QR mesh with identity rotations), or one to the right for $right = 1$ (cf. the QL mesh with identity rotations). Here too, the situation $left = right = 1$ is not permitted. However, $left = right = 0$ causes shifting of elements straight upwards in that particular mesh. Because of the delay element in S cells, the traversal time is the same as for a linear mesh, thereby ascertaining that both mesh types work in a synchronised manner.

A Systolic Array for Algorithm BWK

An array ('module') is constructed which executes one pass through the loop of BWK; during one module traversal the $(w-\bar{k}-1)$ leading elements of the $\bar{k}$ outermost subdiagonals are removed. For the removal of $1 \le k \le q$ subdiagonals, $\lceil k/\bar{k} \rceil$ passes through the module are necessary. Hence, $k \le \bar{k}$ may be assumed for the subsequent analysis; moreover, as will be explained later, $\bar{w} \ge w+\bar{k}$ is required (otherwise the matrix must be partitioned, for according strategies see [Ipse83]).

A module consists of four sets of linear meshes. The $\bar{k}$ QR meshes at the bottom of the module, where input is entered, are followed by a group of shift meshes that direct matrix elements to the proper positions for input to the $\bar{k}$ QL meshes. These in turn are succeeded by another group of shift meshes which, upon output, will have placed matrix elements in the positions assumed at input to the module. A sketch of the module is found in Figure 11.

The QR part causes removal of k subdiagonals along with fill-in of k superdiagonals. The QL part erases this fill-in and restores the k previously eliminated subdiagonals except for their $(w-k-1)$ leading nonzero elements. Furthermore, the outer codiagonals are eliminated by the lower meshes while the inner ones are removed by the upper meshes. More formally, mesh j of the QR part removes the $(q-j+1)$st subdiagonal, $\bar{k}-k+1 \le j \le \bar{k}$, by appending $(\bar{k}-k)$ zero subdiagonals to the left of the band (hence, for $k = 0$, $\bar{w} \ge w+\bar{k}$ is required). Then subdiagonal $(q-k)$ will emerge from the leftmost cell of the top mesh in the QR part. Similarly, superdiagonal $(p+k-j+1)$ is removed in mesh j of the QL part, $\bar{k}-k+1 \le j \le \bar{k}$, and superdiagonal p appears in the rightmost B1 cell. In case the number of subdiagonals to be removed is less than

the number of meshes, $k < \bar{k}$, the lower $(\bar{k} - k)$ meshes of the QR and QL parts do not generate rotations ($on = 0$ in all their B1 cells). Tables 1 and 2 (with $\tilde{w} = \bar{w}$) show processor time schedules for the QR and QL parts.

On account of the shift groups, input and output patterns of a module are identical, so chaining of modules is feasible. It remains to be determined how many meshes are to be present in a shift group. Assuming, that data alignment is left justified in the QR and right justified in the QL part, Tables 1 and 2, with $\tilde{w} = \bar{w}$, show the diagonal, for instance, to leave cell $(\bar{k} - k) + (q + 1) - \bar{k}$ of mesh $\bar{k}$ in the QR part. For proper superdiagonal elimination it has to enter cell $(\bar{w} - (\bar{k} - k) - p)$ in mesh 1 of the QL group. Since $\bar{w} \geq w$, the first shift group has to direct matrix elements to the right. Analogously, cell $(\bar{w} + k - p)$ in the top mesh of the QL part delivers the diagonal, which has to return to its original position in order to enter cell $(\bar{k} - k) + (q + 1)$ in the QR part. Hence, the second group performs shifts to the right. In both cases, data have to be displaced by the same number of places, $(\bar{w} - w) - (\bar{k} - k)$.

For matching hardware and input dimensions, $\bar{k} = k$ and $\bar{w} = w + \bar{k}$, the displacement comes to $\bar{k}$, the difference between w and $\bar{w}$. However, for $\bar{w} > w + \bar{k}$ the difference between w and $\bar{w}$ can be arbitrary and so the amount of displacement. To avoid data dependent control or hardware dimensions, both shift groups are modified as follows : a B1 cell occupies position $ik + 1$, $i \geq 0$, in every mesh. To avoid idle cells, $\bar{w} = \overline{ck} + 1$ for some $\bar{c} \geq 1$ bringing the number of B1 cells per mesh to $\bar{c} + 1$. Instead of guiding elements toward the right boundary, the second shiftgroup now directs elements toward the nearest possible B1 cell which is never more than $\bar{k}$ cells away. On account of the initial left justified data alignment, the first shift group still shifts toward cell 1. Consequently, there are $\bar{k}$ meshes in each shift part and each element is shifted by $\left((\bar{w} - w) - (\bar{k} - k)\right) \mod (\bar{k}+1)$. The preceeding ideas are applied to an example in Figure 12. Processor time schedules for all four groups are given in Tables 1 to 4, $\tilde{w} = \left((\bar{w} - w) - (\bar{k} - k)\right) \mod (\bar{k} + 1)$ in Table 2. Hence, the number of steps for the traversal of a $4\bar{k} \times (\overline{ck} + 1)$ processor module amounts to

$$T(n, 4\bar{k}) = 2(n + 4\bar{k}) - 1.$$

For the sake of simpler control deflation may be omitted, since only identity rotations are generated for leading zero elements. Otherwise the leading $(w - k - 1)$ rows and columns are removed after pass through the module.

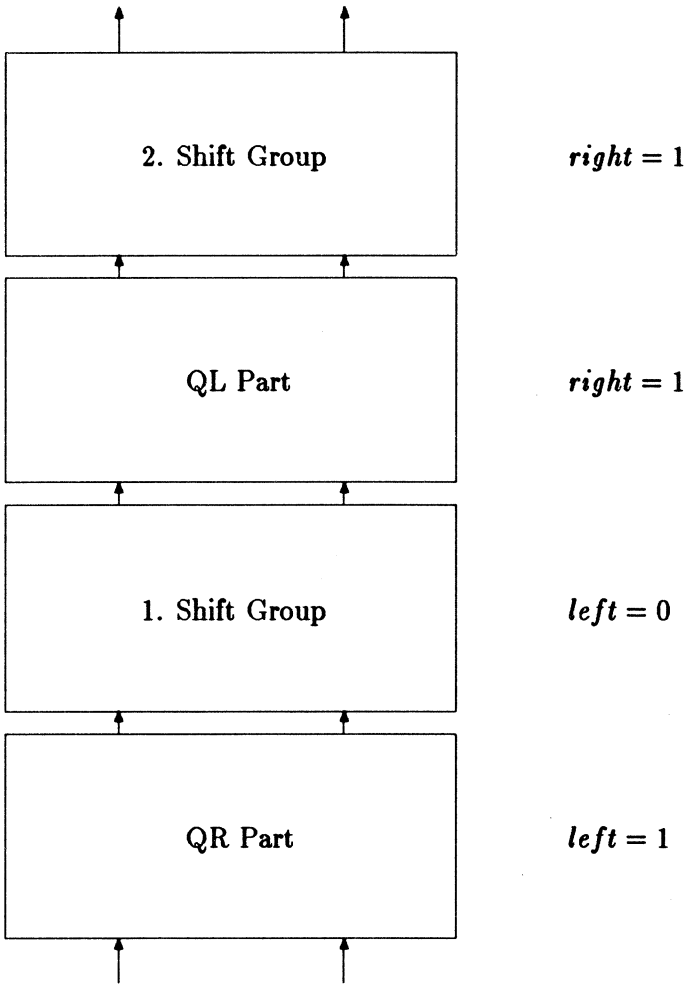
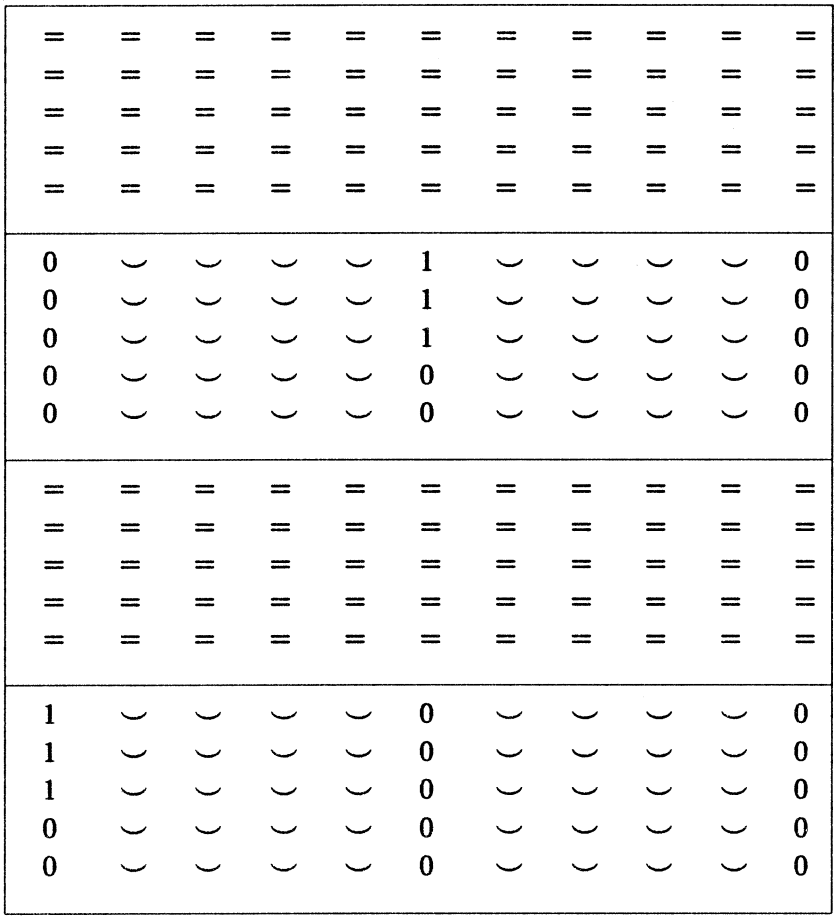


Figure 11. Structure of a Module for Algorithm BWK (Subdiagonal Elimination).



= : S cell
0 : B1 cell ($on = 0$)
1 : B1 cell ($on = 1$)
⌣ : B2 cell

Figure 12. Schematised Module for Subdiagonal Elimination, $\bar{k} = 5$, $\bar{c} = 2$, Programmed for $q = k = 3$, $p = 2$ and $w = 6$.

Codiagonal	Element	enters Cell	of Mesh	at Time
Diagonal	i $\{1 \le i \le n\}$	$x+q+1$	1	$2i-1$
Subdiagonal $q-l+1$ $\{1 \le l \le q\}$	i $\{1 \le i \le n-(q-l+1)\}$	$x+l$	1	$(q-l+1)+(2i-1)$
Superdiagonal $p-l+1$ $\{1 \le l \le p\}$	i $\{1 \le i \le n-(p-l+1)\}$	$x+w-l+1$	1	$(p-l+1)+(2i-1)$
Diagonal	i $\{1 \le i \le n\}$	$x+(q+1)-(j-1)$	j $\{1 \le j \le \bar{k}\}$	t_{ij}
Subdiagonal $q-l+1$ $\{k+1 \le l \le q\}$	i $\{1 \le i \le n-(q-l+1)\}$	$x+l-(j-1)$	j $\{1 \le j \le \bar{k}\}$	$(q-l+1)+t_{ij}$
Subdiagonal $q-l+1$ $\{j \le l \le k\}$	i $\{1 \le i \le n-(q-l+1)\}$	$x+l-(j-1)$	$\bar{k}-k+j$ $\{1 \le j \le k\}$	$(q-l+1)+t_{ij}$
Superdiagonal $p-l+1$ $\{1 \le l \le p\}$	i $\{1 \le i \le n-(p-l+1)\}$	$x+w-l+j$	j $\{1 \le j \le \bar{k}\}$	$(p-l+1)+t_{ij}$
Superdiagonal $p+l$ $\{1 \le l \le j\}$	i $\{1 \le i \le n-(p-l)\}$	$x+w+l-(j-1)$	$(\bar{k}-k)+j$ $\{1 \le j \le k\}$	$(p+l-1)+t_{ij}$

Assume,

1. $1 \le k \le q$, $k \le \bar{k}$ and $w + \bar{k} \le \bar{w}$;
2. element 1 of the diagonal enters at time $t = 1$;
3. $t_{ij} = (2i-1) + 2(j-1)$;
4. $x = \bar{k} - k$.

Table 1. Processor Time Schedule for Meshes in the QR Group of a Module for Subdiagonal Elimination.

Codiagonal	Element	enters Cell	of Mesh	at Time
Diagonal	i $\{1 \le i \le n\}$	$x-p$	1	$2i-1$
Subdiagonal $q-l+1$ $\{k+1 \le l \le q\}$	i $\{1 \le i \le n-(q-l+1)\}$	$x-w+l$	1	$(q-l+1)+(2i-1)$
Superdiagonal $p+k-l+1$ $\{1 \le l \le p+k\}$	i $\{1 \le i \le n-(p+k-l+1)\}$	$x+k-l+1$	1	$(p-l+1)+(2i-1)$
Diagonal	i $\{1 \le i \le n\}$	$x-p+(j-1)$	j $\{1 \le j \le \bar{k}\}$	t_{ij}
Subdiagonal $q-l+1$ $\{k+1 \le l \le q\}$	i $\{1 \le i \le n-(q-l+1)\}$	$x-w+l+(j-1)$	j $\{1 \le j \le \bar{k}\}$	$(q-l+1)+t_{ij}$
Subdiagonal $q-k+l$ $\{1 \le l \le j\}$	i $\{1 \le i \le n-(q-k+l)\}$	$x-w-l+j$	$\bar{k}-k+j$ $\{1 \le j \le k\}$	$(q-k+l)+t_{ij}$
Superdiagonal $p-l+1$ $\{1 \le l \le p\}$	i $\{1 \le i \le n-(p-l+1)\}$	$x-l+j$	j $\{1 \le j \le \bar{k}\}$	$(p-l+1)+t_{ij}$
Superdiagonal $p+l$ $\{1 \le l \le k\}$	i $\{1 \le i \le n-(p+l)\}$	$x-l+(j-1)$	j $\{1 \le j \le \bar{k}-k\}$	$(p+l)+t_{ij}$
Superdiagonal $p+l$ $\{1 \le l \le k-j+1\}$	i $\{1 \le i \le n-(p+l)\}$	$x-l+(j-1)$	$\bar{k}-k+j$ $\{1 \le j \le k\}$	$(p+l)+t_{ij}$

Assume,

1. $1 \le k \le q$, $k \le \bar{k}$ and $w + \bar{k} \le \bar{w}$;
2. element 1 of the diagonal enters at time $t = 1$;
3. $t_{ij} = (2i-1) + 2(j-1)$;
4. $x = \tilde{w} - (\bar{k}-k) - k$;
5. $\tilde{w} = \bar{w}$ or $\left((\bar{w}-w) - (\bar{k}-k)\right) \mod k+1$.

Table 2. Processor Time Schedule for Meshes in the QL Group of a Module for Subdiagonal Elimination.

Codiagonal	Element	enters Cell	of Mesh	at Time
Diagonal	i $\{1 \le i \le n\}$	$q-k+1$ $q-k+j$	j $\{1 \le j \le \bar{k}-z\}$ $\bar{k}-z+j$ $\{1 \le j \le z\}$	t_{ij}
Subdiagonal $q-l+1$ $\{k+1 \le l \le q\}$	i $\{1 \le i \le n-(q-l+1)\}$	l $l+(j-1)$	j $\{1 \le j \le \bar{k}-z\}$ $\bar{k}-z+j$ $\{1 \le j \le z\}$	$(q-l+1)+t_{ij}$
Superdiagonal $p+k-l+1$ $\{1 \le l \le p+k\}$	i $\{1 \le i \le n-(p+k-l+1)\}$	$w-l+1$ $w-l+j$	j $\{1 \le j \le \bar{k}-z\}$ $\bar{k}-z+j$ $\{1 \le j \le z\}$	$(p+k-l+1)+t_{ij}$

Assume,

1. $1 \le k \le q$, $k \le \bar{k}$ and $w + \bar{k} \le \bar{w}$;
2. element 1 of the diagonal enters at time $t = 1$;
3. $t_{ij} = (2i-1) + 2(j-1)$;
4. meshes $\bar{k}-z+1, \ldots, \bar{k}$ shift to the right, all others upwards, $0 \le z \le \bar{k}$.

Table 3. Processor Time Schedule for Meshes in the First Shift Group of a Module for Subdiagonal Elimination.

Codiagonal	Element	enters Cell	of Mesh	at Time
Diagonal	i $\{1 \le i \le n\}$	$\bar{w} - p$ $\bar{w} - p - (j-1)$	j $\{1 \le j \le \bar{k} - z\}$ $\bar{k} - z + j$ $\{1 \le j \le z\}$	t_{ij}
Subdiagonal $q - l + 1$ $\{1 \le l \le q\}$	i $\{1 \le i \le n - (q - l + 1)\}$	$\bar{w} - w - l$ $\bar{w} - w - l - (j-1)$	j $\{1 \le j \le \bar{k} - z\}$ $\bar{k} - z + j$ $\{1 \le j \le z\}$	$(q - l + 1) + t_{ij}$
Superdiagonal $p - l + 1$ $\{1 \le l \le p\}$	i $\{1 \le i \le n - (p - l + 1)\}$	$\bar{w} - l + 1$ $\bar{w} - l + j + 2$	j $\{1 \le j \le \bar{k} - z\}$ $\bar{k} - z + j$ $\{1 \le j \le z\}$	$(p - l + 1) + t_{ij}$

Assume,

1. $1 \le k \le q$, $k \le \bar{k}$ and $w + \bar{k} \le \bar{w}$;
2. element 1 of the diagonal enters at time $t = 1$;
3. $t_{ij} = (2i - 1) + 2(j - 1)$;
4. meshes $\bar{k} - z + 1, \ldots, \bar{k}$ shift to the right, all others upwards, $0 \le z \le \bar{k}$.

Table 4. Processor Time Schedule for Meshes in the Second Shift Group of a Module for Subdiagonal Elimination.

Programming of a Module

Given a module for subdiagonal elimination with parameters $\bar{c}$ and $\bar{k}$, the following requirements must be satisfied :

$$\bar{w} = \overline{ck} + 1, \qquad \bar{k} \geq 1, \quad \bar{c} \geq 1.$$

Meshes $1 \ldots \bar{k}, 2\bar{k}+1 \ldots 3\bar{k}$: cell $i\bar{k}+1$ is a B1 cell, all others are B2 cells, $0 \leq i \leq \bar{c}$.
Meshes $\bar{k}+1 \ldots 2\bar{k}, 3\bar{k}+1 \ldots 4\bar{k}$: all cells are S cells.

The initial (default) state of the module is a follows :

1. Bit Data Lines
   Meshes $1 \ldots \bar{k}$ (QR group) : $left = 1$, $right = 0$.
   Meshes $\bar{k}+1 \ldots 2\bar{k}$ (1. shift group) : $left = right = 0$.
   Meshes $2\bar{k}+1 \ldots 3\bar{k}$ (QL group) : $left = 0$, $right = 1$.
   Meshes $3\bar{k}+1 \ldots 4\bar{k}$ (2. shift group) : $left = right = 0$.
   Meshes $1 \ldots \bar{k}, 2\bar{k}+1 \ldots 3\bar{k}$: cells $i\bar{k}+1$ (B1 cells), $1 \leq i \leq \bar{c}$, have $on = 0$.
2. Matrix and Rotation Lines
   Mesh 1 : all matrix input lines are 0.
   Meshes $1 \ldots \bar{k}$: the left rotation line of cell 1 contains the identity rotation.
   Meshes $2\bar{k}+1 \ldots 3\bar{k}$: the right rotation line of cell $\bar{w}$ contains the identity rotation.
   Meshes $1 \ldots \bar{k}, 3\bar{k}+1 \ldots 4\bar{k}$: the right matrix input to cell $\bar{w}$ is 0.
   Meshes $\bar{k}+1 \ldots 3\bar{k}$: the left matrix input to cell 1 is 0.

If the input matrix satisfies $\bar{w} < w + \bar{k}$, it is partitioned into submatrices of size $n' \times n'$, where $n' = \lfloor (\bar{w} - \bar{k} + 1)/2 \rfloor$; for details see [Ipse83]. Thus, assume $\bar{w} \geq w + \bar{k}$. The programme in Figure 13 describes elimination of subdiagonals. If superdiagonals are to be annihilated, the functions of For superdiagonal elimination the groups QR and QL are exchanged, as well as the shift groups 1 and 2 by accordingly modifying the bitlines *left*, *right* and *on*.

A Systolic Array for Algorithm SVE

One iteration of SVE for bidiagonal matrices is implemented by taking advantage of simple arrays for matrix transposition [Ipse83], to be discussed in the next section, and QR decomposition [HeIp83, Ipse83]. Formation of transposes 'on the fly' enhances the degree of pipelining and avoids time consuming storage of data during a computation for the purpose of changes in address patterns.

The initial formation of the bulge (step 1, $i = 0$) is done with the help of a four processor matrix transposition array (Figure 14) and a four processor matrix multiplication array [KuLe78] in $2n - 1$ and $3n + 2$ steps, respectively. Steps 2 and 3 employ a linear three processor QR decomposition array (Figure 15) with one B1 and two B2 cells. Its inputs are $(B_i - s_i I)$ and R_i^T, respectively, and the execution time is $2(n-1)$. Cell B1 generates rotations in step 2 and forwards the recycled rotations in step 3. Formation of the transpose R_i^T before step 3 and B_i^T after step 3 is accomplished with fifteen and twelve processor networks (Figure 16) in $2(n+6)$ and $2(n+4)$ steps, respectively. Because of hardware requirements for large bandwidths, w, as well as demands on accuracy, the implicitly shifted method is recommended.

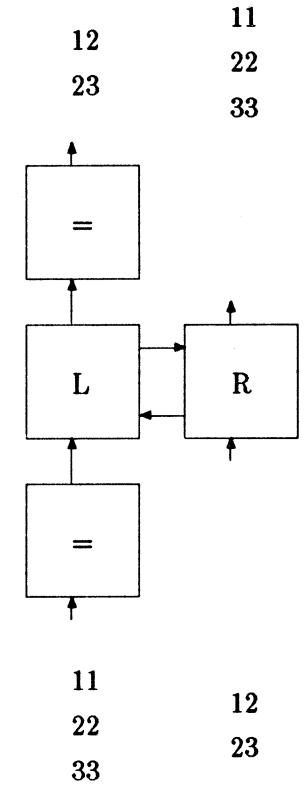


Figure 14. Matrix Transposition Array for an Upper Bidiagonal Matrix.

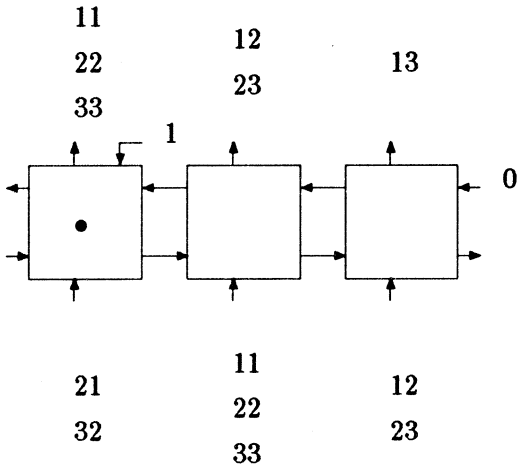
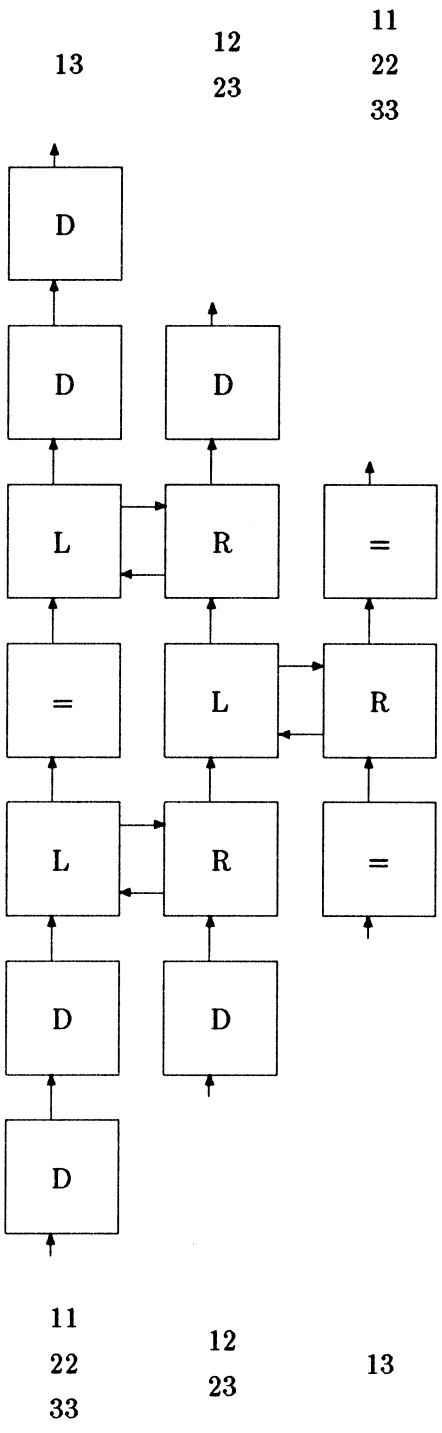
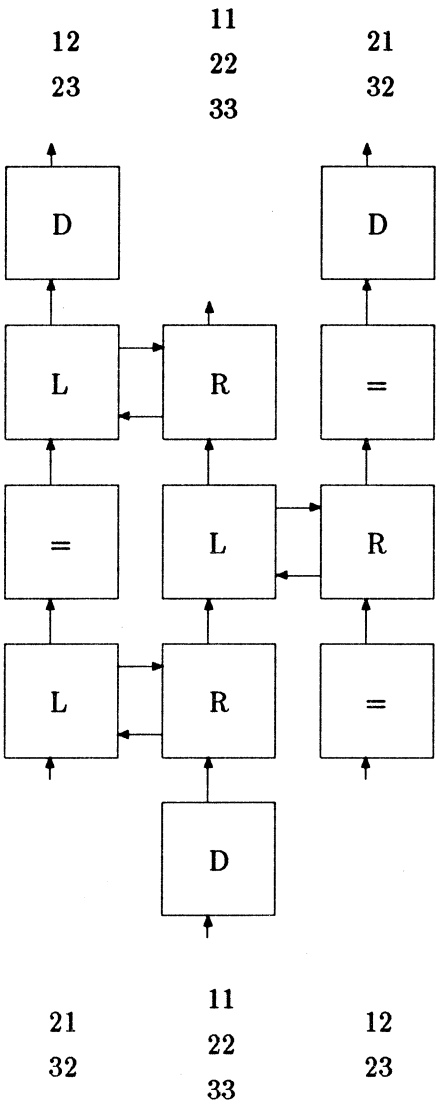


Figure 15. QR Array for a Tridiagonal Matrix.



(a) Upper Triangular Matrices.

Figure 16. Matrix Transposition Arrays for Matrices of Bandwidth $w = 3$.



(b) Tridiagonal Matrices.

Figure 16. Continuation.

Systolic Arrays for Matrix Transposition

The transpose of a matrix, in particular a banded matrix, can be obtained by reflecting the codiagonals about the diagonal, whereby the ordering between successive elements of a codiagonal is preserved. That is, the ith subdiagonal becomes the ith superdiagonal, $1 \leq i \leq q$, and the jth superdiagonal turns into the jth subdiagonal, $1 \leq j \leq p$. The square array of $O(w^2)$ delay elements has a low hardware cost for matrices of small bandwidth, in contrast to $O(n^2)$ cells for dense matrices (the case $w = 2$ is an exception, as only 2 not 4 cells are needed).

Four kinds of delay elements, shown in Figure 17, direct the matrix elements. A pair of cells, (L, R), acts as a 2×2 crossbar switch if both, L and R, receive their inputs at the same time. As illustrated in Figure 18 for the case $w = 6$, the L and R cells are arranged in a chequered fashion as a square orthogonally connected w^2 processor array. At the left and right edges (I/O takes place at the bottom and top edges) where they do not pair, a S cell (as before, but $left = right = 0$, Figure 17) is used instead. For the (L, R) pair to properly function, data must be delivered and output by non-time skewed codiagonals. It takes two time steps to traverse a mesh and values may are input no sooner than every other cycle. This implies that the ith element of each codiagonal enters the the jth linear mesh at time

$$T(i,j) - 1 = 2(i + j - 1) - 1, \qquad 1 \leq i \leq n, \quad 1 \leq j \leq w,$$

bringing the total traversal time to

$$T(n,w) = 2(n + w) - 1.$$

Absent values at the end of the 'shorter' codigonals are represented by a default value of zero. The array size depends only on w not on the particular values of p and q.

In case of a disagreement between matrix and hardware dimensions let the array be of dimension $\bar{w} \times \bar{w}$. If the bandwidth of the matrix is too small, $w < \bar{w}$, then prior to input $\bar{w} - w$ codiagonals are arbitrarily attached to the outside of the band and removed on output. For the opposite, $\bar{w} < w$, there is only a simple solution if a $\bar{w} \times \bar{k}$ array is available and $\bar{k} \geq w$ is a multiple of $\bar{w}$. The transpose is available after $\lceil \bar{k}/\bar{w} \rceil$ passes through the array.

A non-time skewed codiagonal address pattern calls for a pre- and postprocessor to ensure compatibility with the codiagonalwise I/O pattern of all the other arrays (Figures 14 and 16). Let k be the larger of p and q. Consisting of D cells the preprocessor has the ith codiagonal traverse $k - i$ cells upwards (k for the diagonal) so corresponding elements of all codiagonals enter the transposition array at the same time. Similarly, the ith codiagonal crosses i S cells in the postprocessor (none for the diagonal) to re-establish the time skewed codiagonals. The number of cells in the two adjustment arrays comes to

$$\left(\sum_{i=1}^{p} (k - i) + \sum_{i=1}^{q} (k - i) + k \right) + \left(\sum_{i=1}^{p} i + \sum_{i=1}^{q} i \right) = wk,$$

and including the transposition array this makes $w(w + k)$ cells. If the first element, a_{11}, enters the preprocessor at time $t = 1$ and a_{nn} is the last element to leave the postprocessor, having to traverse $k + w$ cells, then a transposition is completed in time

$$T(n, k + w) = 2(n + k + w) - 1 \leq 2(n + 2w).$$

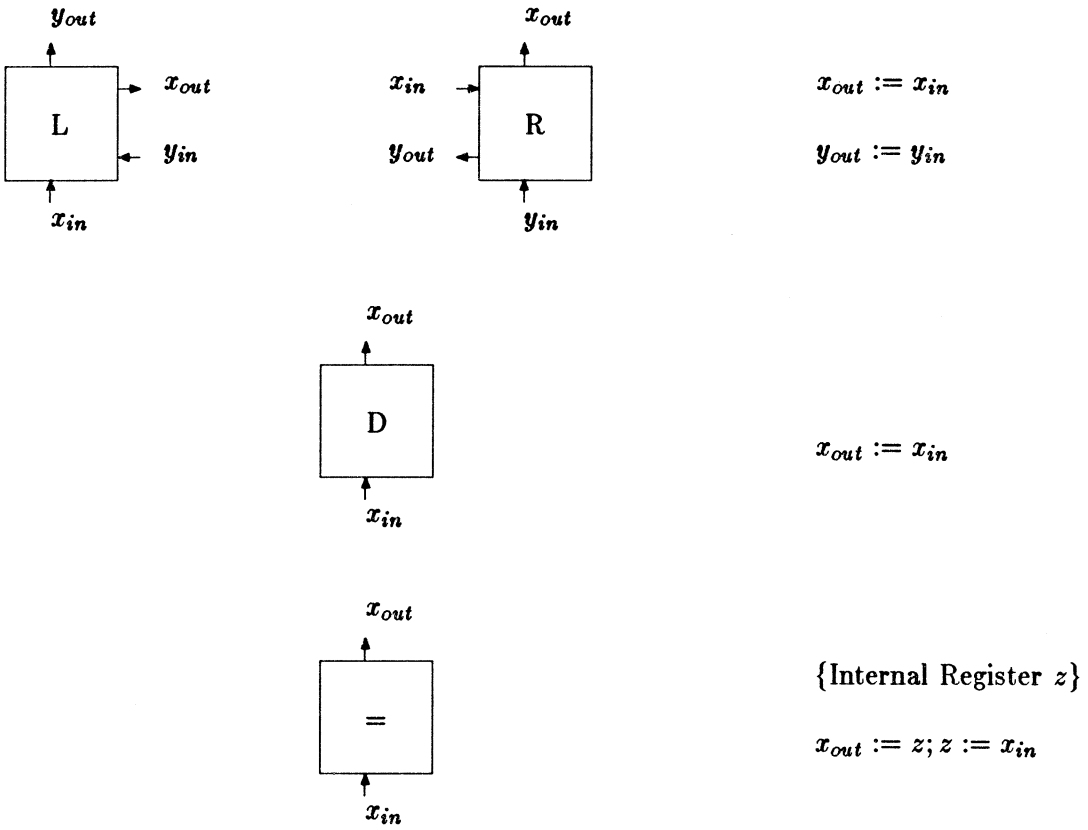


Figure 17. Processors for Matrix Transposition.

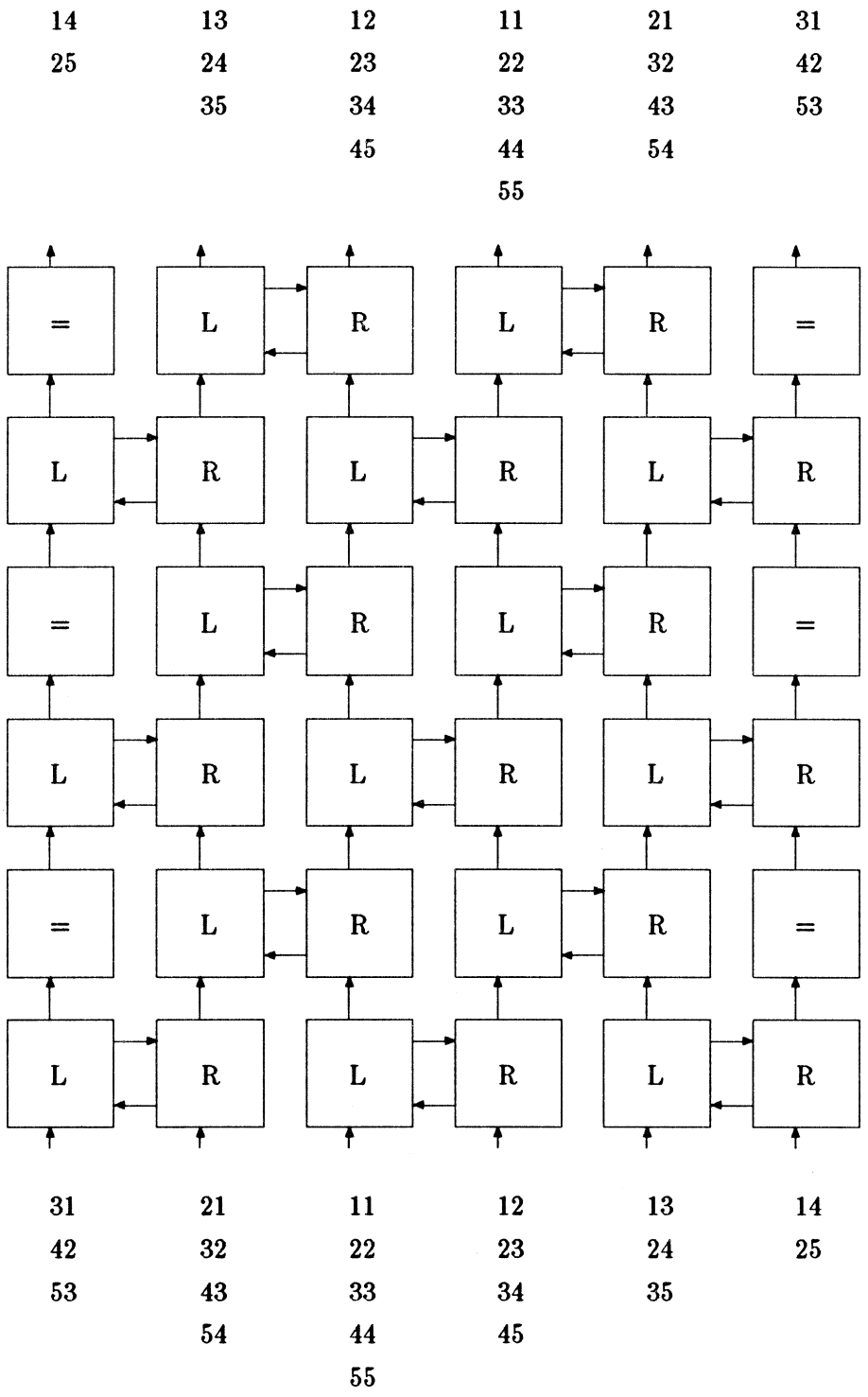


Figure 18. Matrix Transposition Array with Non-Time Skewed Codiagonal I/O Format, $\bar{w} = 6$.

A Systolic Array for Algorithm SVI

The processor array which implements SVI (except for step 1) is a special case of the bandwidth reduction array ($q = 1, p = 2$) and consists of three linearly connected meshes, a total of 5 processors, shown in Figure 19(a). The hardware count is independent of n. The bottom mesh, consisting of B2 cells with $right = 1$, is responsible for the formation of

$$B_i = A_i P_i^T,$$

that is the initial creation of the bulge (step 2 in SVI). P_i, computed in a scalar unit (step 1 of SVI), and the first element of A_i, a_{11}, enter the network at the same time. Alternatively, since B_i differs from A_i only in the leading two rows and columns, it also could be determined by a scalar unit and the bottom mesh omitted (Figure 19(b)). This layer alters matrix elements during the first three time steps, all rotations following P_i^T are identities.

Reduction of B_i (step 4 of SVI) is done entirely by cell B which in turn performs computations POS.2.1 and PRE.2.1, caused by $on_{pre} = on_{post} = 1$. POS.2.2 and PRE.2.2 are never encountered during this iteration and can thus be disregarded. Another B2 cell in the second mesh delivers elements of the bulge to cell B. The total time to determine $A_{i+1} = \overline{B}_n$ comes to

$$T(n, 2) = 2n + 3,$$

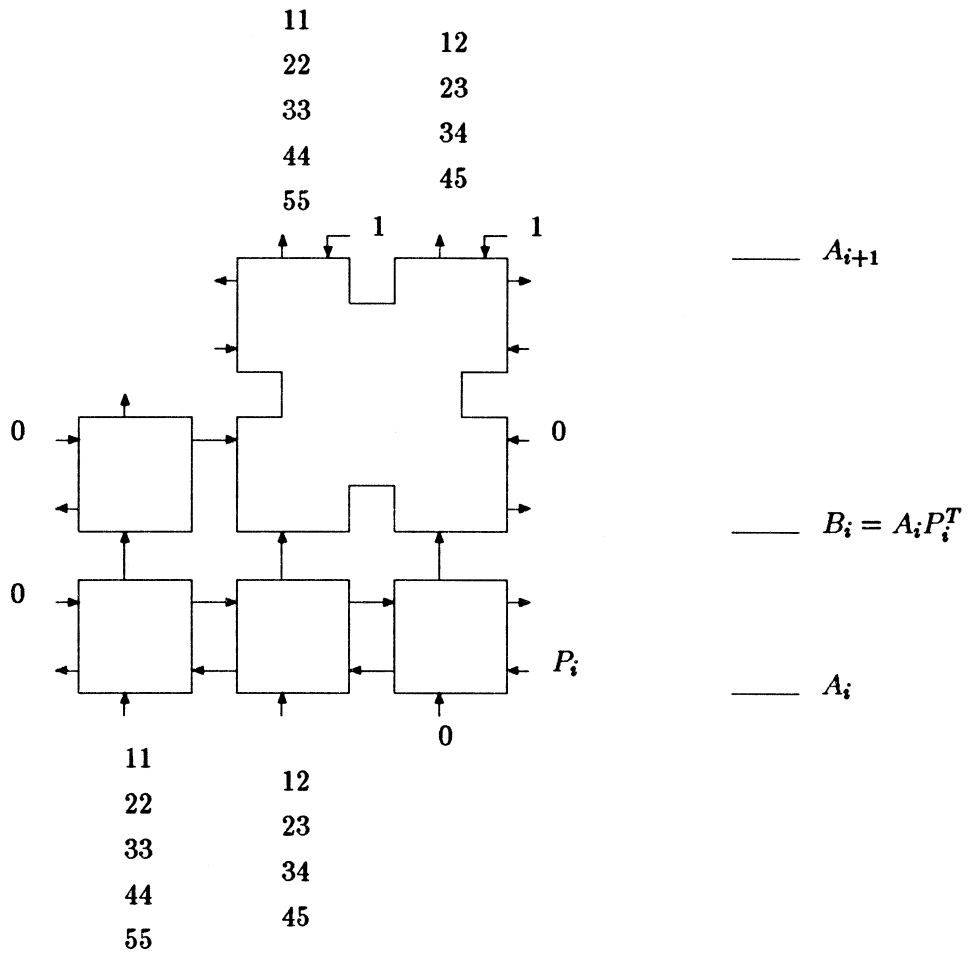
and excluding the computation of B_i to $2n + 1$. The network, I/O format and execution trace are displayed in Figures 19 and 20. After each singular value computation, the trailing row and column of A_i are disregarded.

The plain (unshifted) algorithm is linearly convergent. If Wilkinson's Shift [Wilk68] is used, the algorithm is ultimately quadratically convergent and in practice seems to be cubically convergent; three iterations on the average are enough to compute a singular value [Parl80]. Since the shift is determined from the trailing end of the matrix and applied to the leading part, it obstructs the pipelining process and necessitates transfers to memory between iterations. Under these circumstances, the average number of (arithmetic) steps for one singular value amounts to

$$3T(n, 2) = 6n + 5.$$

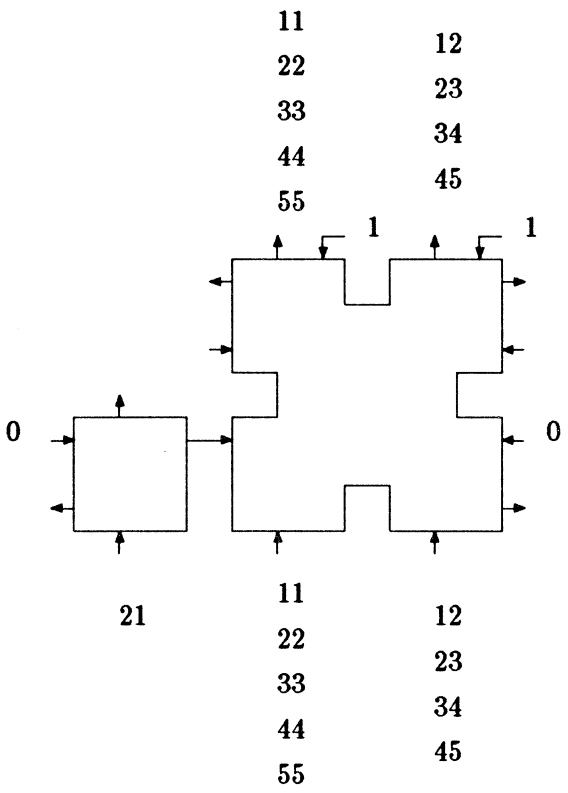
Research is under way to find means of accelerating convergence (obviating the need for intermittent memory transfers) and hence a computation time of

$$T(n, 2\bar{k}) = 2(n + 2\bar{k}) - 1 \qquad \text{for some } \bar{k} \geq 1.$$



(a) Computation of B_i Included in the Network and Indication to the Right of Intermediate Quantities formed.

Figure 19. Systolic Arrays for Algorithm SVI.



(b) Reduction of B_i to Bidiagonal Form A_{i+1}.

Figure 19. Continuation.

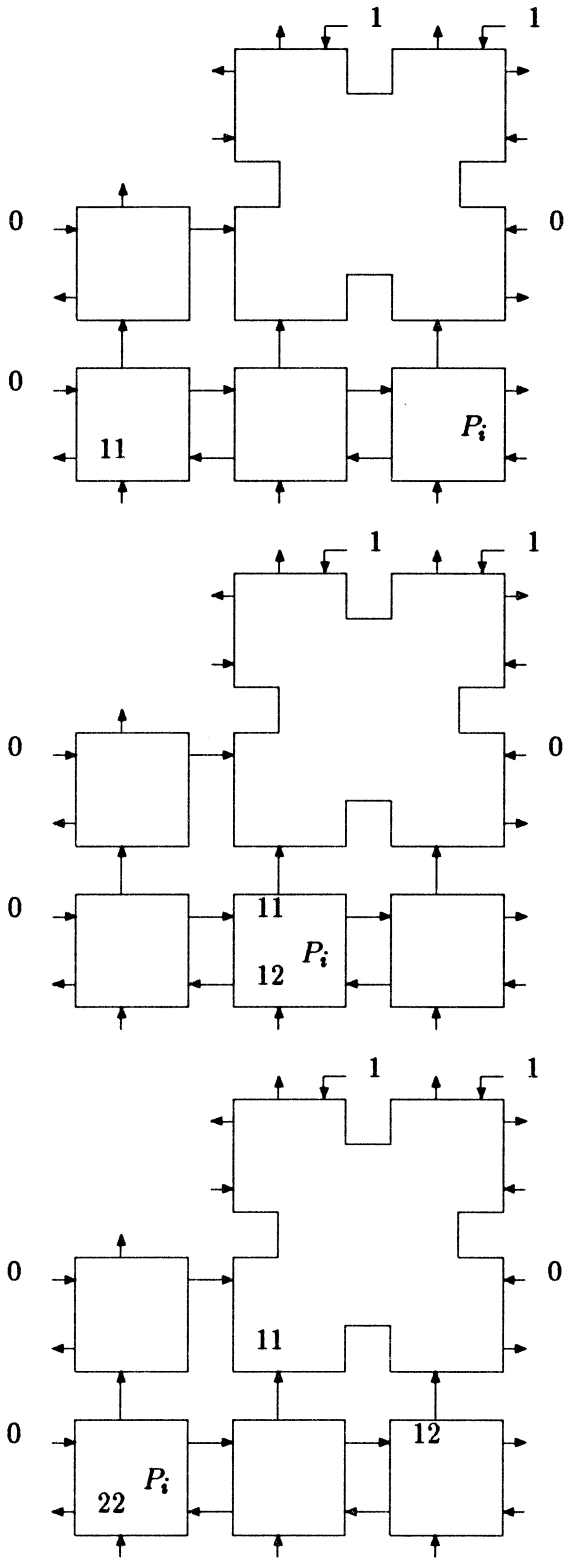


Figure 20. Partial Execution Trace for the Network in Figure 19(a).

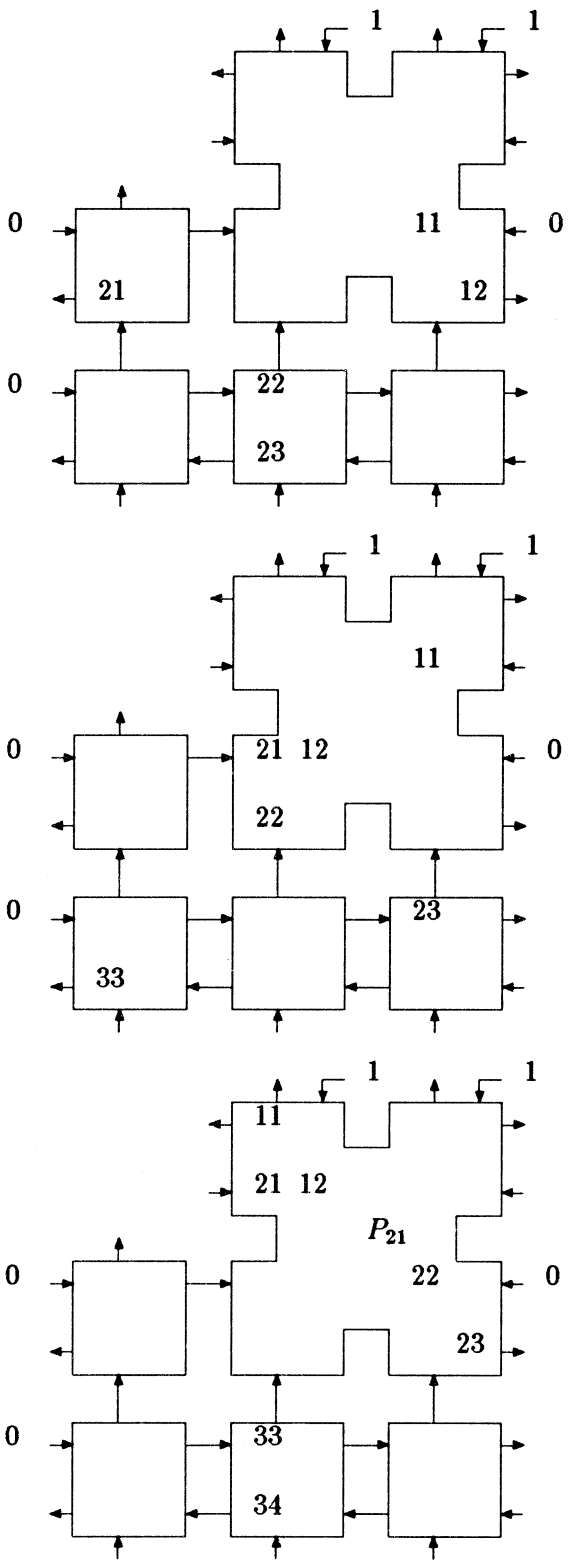
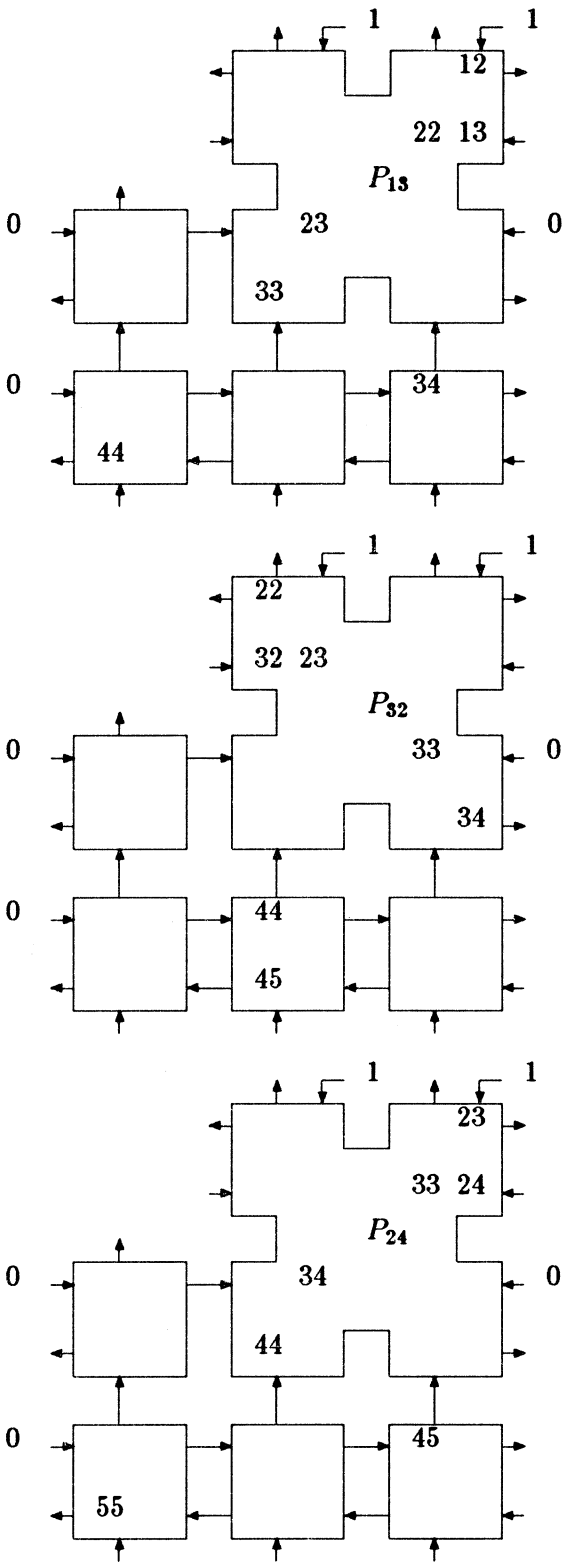


Figure 20. Continuation.



$t = 7$

$t = 8$

$t = 9$

Figure 20. Continuation.

Bibliography

[BrLu82] Brent, R.P. and Luk, F.T., *"A Systolic Architecture for the Singular Value Decomposition,"* Report TR-CS-82-09, Dept of Computer Science, The Australian National University, September 1982.

[BrLV83a] Brent, R.P., Luk, F.T., and Van Loan, C., *"Computation of the Singular Value Decomposition Using Mesh-Connected Processors,"* Report TR 82-528, Dept of Computer Science, Cornell University, March 1983.

[BrLV83b] Brent, R.P., Luk, F.T., and Van Loan, C., *"Computation of the Generalized Singular Value Decomposition Using Mesh-Connected Processors,"* Report TR 83-563, Dept of Computer Science, Cornell University, July 1983.

[FiLP82] Finn, A.M., Luk, F.T. and Pottle, C., "Systolic Array Computation of the Singular Value Decomposition," *Proc. SPIE Symposium*, 341 (Real Time Processing V), pp 35-43, 1982.

[Gent75] Gentleman, W.M., "Error Analysis of QR Decompositions by Givens Transformations," *Linear Algebra and its Applications*, 10, pp 189-197, 1975.

[GoRe70] Golub, G.H. and Reinsch, C., "Singular Value Decomposition and Least Squares Solutions," *Num. Math.*, 14, pp 403-420, 1970.
Handbook for Automatic Computation, II, Linear Algebra Springer Verlag, New York, 1971

[Help83] Heller, D.E. and Ipsen, I.C.F., "Systolic Networks for Orthogonal Decompositions," *SIAM J. Sci. Stat. Comp.*, 4, pp 261-269, 1983.

[Ipse83] Ipsen, I.C.F., *Stable Matrix Computations in VLSI*, Ph.D Thesis, The Pennsylvania State University, August 1983.

[Kung81] Kung, H.T., "Why Systolic Architectures ?," *IEEE Computer*, 15 No 1, pp 37-46, January 1982.

[KuLe78] Kung, H.T. and Leiserson, C.E., "Systolic Arrays (for VLSI)," *Sparse Matrix Proceedings*, SIAM, pp 256-282, 1978.

[Parl80] Parlett, B.N., *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, New Jersey, 1980.

[Ruti63] Rutishauser, H., "On Jacobi Plane Rotations," *Symposium in Applied Mathematics*, 15, pp 219-239, 1963.

[SaKu78] Sameh, A.H. and Kuck, D.J., "On Stable Linear System Solvers," *JACM*, 25, pp 81-91, 1978.

[Schr83a] Schreiber, R., *"A Systolic Architecture for the Singular Value Decomposition,"* Proc. 1er Colloque International sur les Méthodes Vectorielles et Paralèles en Calcul Scientifique, Electricité de France, Bulletin de la Direction des Etudes et Recherches, Série C No 1, 1983.

[Schr83b] Schreiber, R., *"On the Systolic Arrays of Brent, Luk and Van Loan for the Symmetric Eigenvalue and Singular Value Problems,"* Report TRITA-NA-8311, 1983, NADA, KTH Stockholm.

[Wilk65] Wilkinson, J.H., *The Algebraic Eigenvalue Problem*, Oxford University Press, 1965.

[Wilk68] Wilkinson, J.H., "Global Convergence of the Tridiagonal QR Algorithm with Origin Shifts," *Linear Algebra and its Applications*, 1, pp 409-420, 1968.