

Abstract:

In the preceding paper [16] we presented several systolic algorithms for Factorial Data Analysis all running on the same triangular systolic array with orthogonal connections: **SARDA** (Systolic Array for Data Analysis). We restricted our study to matrix computations. In this paper we will now turn ourselves to the symmetric eigenvalue problem on SARDA and study especially tridiagonalization followed by multisection for the eigenvalues and inverse iteration method for the associated eigenvectors.

**A Programmable Systolic
Array for Factorial Analysis.
Part II: The Symmetric Eigenvalue Problem**

Tiba Porta

Research Report YALEU/DCS/RR-543

June 1987

This work was supported by AFOSR-86-0098 and done in part at LCS/Grenoble

1. Introduction:

Let us remind our purpose: we wanted to devise efficient algorithms for Factorial Data Analysis all running on the same triangular systolic array SARDA. We are now studying the symmetric eigenvalue problem on the symmetric positive definite matrix W of size k (for the other algorithms involved see part I [16]).

As we have a systolic array of size s with usually $k > s$, we use subspace iteration with block-partitioning (see [5],[15]) and with these two techniques we can restrict our study to the symmetric eigenvalue problem on W' of size s and obtain the t largest eigenvalues of W with $t \leq s$.

Actually a large variety of systolic algorithms have been studied for the symmetric eigenvalue problem [10]. We can distinguish two main classes of methods: the first is the Jacobi method [1],[20], the second is a tridiagonalization followed either by QR iteration with shift [8], or by bisection [21]. No method actually proposed is linear in time.

Other methods are possible when the symmetric matrix W is positive definite: LR Cholesky (chapter 8 of [22]) which is an adaptation of the LR algorithm to the symmetric case on the matrix W tridiagonalized before, or the singular value decomposition (SVD) of R upper triangular matrix got by Cholesky factorization of W ($W = R^T R$). (see [15])

Let us now describe briefly several of the methods listed above and some of their drawbacks.

- The Jacobi method [1] consists of applying planar rotations on the rows and columns of W to annihilate its non diagonal elements. This algorithm requires a square array of $s^2/4$ processors with octogonal connections (every processor has eight neighbors). After $O(s \log s)$ time steps we obtain the eigen elements of W' . This method is quite fast but it shows some difficulties to implement because at every time step, every processor has to communicate with its 8 nearest neighbors. So it requires a programming of data exchanges quite complex. Moreover as we have no diagonal connections between processors we have to simulate them which complicates the programming and delays the computations.
- The QR iteration for band matrices can be applied to the symmetric tridiagonalized matrix W'' (see [8],[9]). One iteration takes about $2s$ time steps but without shift the convergence is slow. (The non diagonal elements $w_{i-1,i}^k, i = 2, \dots, s$ of W_k'' converge to 0 at the speed of $w_{i-1,i}^k/w_{i-1,i}^{k-1} = O((\lambda_i/\lambda_{i-1})^k)$ where the eigenvalues $\lambda_i, i = 1, \dots, s$ are arranged by decreasing order). The problem is that in the QR iteration you need to have the k th shift σ_k to compute the k th QR factorization $W_k'' - \sigma_k I = Q_k R_k$ and you can only get it when you have finished the preceding iteration $W_{k-1}'' = R_{k-1} Q_{k-1} + \sigma_{k-1} I$. So you cannot fit the iterations into each other and there is a great loss of time.
- The LR Cholesky iteration can also be applied to the symmetric tridiagonal matrix W'' (see [22]). One iteration takes about $2s$ time steps and the iterations can fit into each other but the convergence is very slow especially when several eigenvalues are close to one another. (The non diagonal elements $w_{i-1,i}^k, i = 2, \dots, s$ of W_k'' converge to 0 at the speed of $w_{i-1,i}^k/w_{i-1,i}^{k-1} = O((\lambda_i/\lambda_{i-1})^{k/2})$).
- We did not use the methods related to the SVD of R , the Cholesky factor of W ($W = R^T R$), because they are of the same type as the Jacobi method described above. The Hestenes method consists of applying planar rotations to the columns of R to orthogonalize them ([2],[19]). It is analogous to the Jacobi method applied to $R^T R$. The SVD-Jacobi method applies once more planar rotations on the rows and columns of R to annihilate its non diagonal elements but as R is not symmetric the rotations applied to the rows and the columns of R are not the same ([3],[14]).

After this short overview of the methods actually available for the symmetric eigenvalue problem, (see also table 1), we will now detail our choice: the tridiagonalization of W' , described in section 2, followed by the multisection method, described in section 3, for the eigenvalues of W' . In section 4 we explain how to get the eigenvectors of W' with the inverse iteration method.

2. Tridiagonalization of W' :

Let W' be a symmetric matrix of size s , we have to define a unitary matrix P such that $W_0 = PW'P^T$ is tridiagonal. To assure the numerical stability (see [22]) the matrix P can be either a product of *Householder transformations* [11], or a product of *rotations* [7],[8],[9]. The tridiagonalization is done column by column: we apply series of projections or rotations by premultiplication to annihilate a column of W' , then by postmultiplication with the transpose matrix of these operators, we annihilate the corresponding row. So we get a matrix unitary similar to W' with one column and row annihilated [17],[18]. This method requires $O(s^2)$ time steps, a triangular array of $s(s+1)/2$ processors and a systolic shifter which allows to transpose matrices without delays. We use Givens rotations to annihilate the elements of W' . Their utilization is up to date especially for the QR factorization [7],[9].

Let $[w_{i-1,j}, w_{i,j}]^T$ be a couple of elements of W' . In order to annihilate $w_{i,j}$ by using the pivot $w_{i-1,j}$ we build the Givens rotation $R_{i,j} = \begin{pmatrix} \cos \theta_{i,j} & \sin \theta_{i,j} \\ -\sin \theta_{i,j} & \cos \theta_{i,j} \end{pmatrix}$ defined by:
 $\rho = [w_{i-1,j}^2 + w_{i,j}^2]^{1/2}$, $\cos \theta_{i,j} = w_{i-1,j}/\rho$, $\sin \theta_{i,j} = w_{i,j}/\rho$.

We have then $R_{i,j}[w_{i-1,j}, w_{i,j}]^T = [\rho, 0]^T$.

Set down $W' = (w_{i,j})_{1 \leq i,j \leq s}$. We apply $(s-2)$ rotations by premultiplication to the $(s-1)$ last rows of W' in order to annihilate successively $w_{s,1}, w_{s-1,1}, \dots, w_{3,1}$.

$w_{s,1}$ is annihilated with the pivot $w_{s-1,1}$ and $R_{s,1}[w_{s-1,1}, w_{s,1}]^T = [w_{s-1,1}^1, 0]^T$.

$w_{s-1,1}^1$ is annihilated with the pivot $w_{s-2,1}$ and $R_{s-1,1}[w_{s-2,1}, w_{s-1,1}^1]^T = [w_{s-2,1}^1, 0]^T$.

...

After the $(s-2)$ th rotation we have the $(s-2)$ last elements of the first column of $Q_1^T W' = R_{3,1} \dots R_{s-1,1} R_{s,1} W'$ annihilated. This operation is called the first "forward pass". We then apply these $(s-2)$ rotations by postmultiplication to the $(s-1)$ last columns of $Q_1^T W'$ and we compute $Q_1^T (Q_1^T W')^T = Q_1^T W' Q_1$. This is the first "backward pass". In fact we only compute the product of Q_1^T with the $(s-1)$ last rows of $(Q_1^T W')^T$ because we already know that the first row of $Q_1^T W' Q_1$ is equal to the first column of $Q_1^T W'$. And we get:

$$Q_1^T W' Q_1 = \begin{pmatrix} w_{1,1} & w_{2,1}^1 & 0 & \dots & 0 \\ w_{2,1}^1 & w_{2,2}^2 & w_{2,3}^3 & \dots & w_{2,s}^2 \\ 0 & w_{3,2}^3 & w_{3,3}^3 & \dots & w_{3,s}^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & w_{s,2}^2 & w_{s,3}^3 & \dots & w_{s,s}^2 \end{pmatrix}$$

The upper indices of the elements of $Q_1^T W' Q_1$ represent the number of rotations applied to the elements during the first forward and backward passes.

Then we apply $(s-3)$ rotations by premultiplication to the $(s-2)$ last rows of $W^1 = Q_1^T W' Q_1$ in order to annihilate successively $w_{s,2}^2, w_{s-1,2}^3, \dots, w_{4,2}^3$. If Q_2^T is the product of these $(s-3)$ rotations the second forward pass consists of $Q_2^T W^1$ and the second backward pass, of $Q_2^T (Q_2^T W^1)^T = Q_2^T Q_1^T W' Q_1 Q_2$.

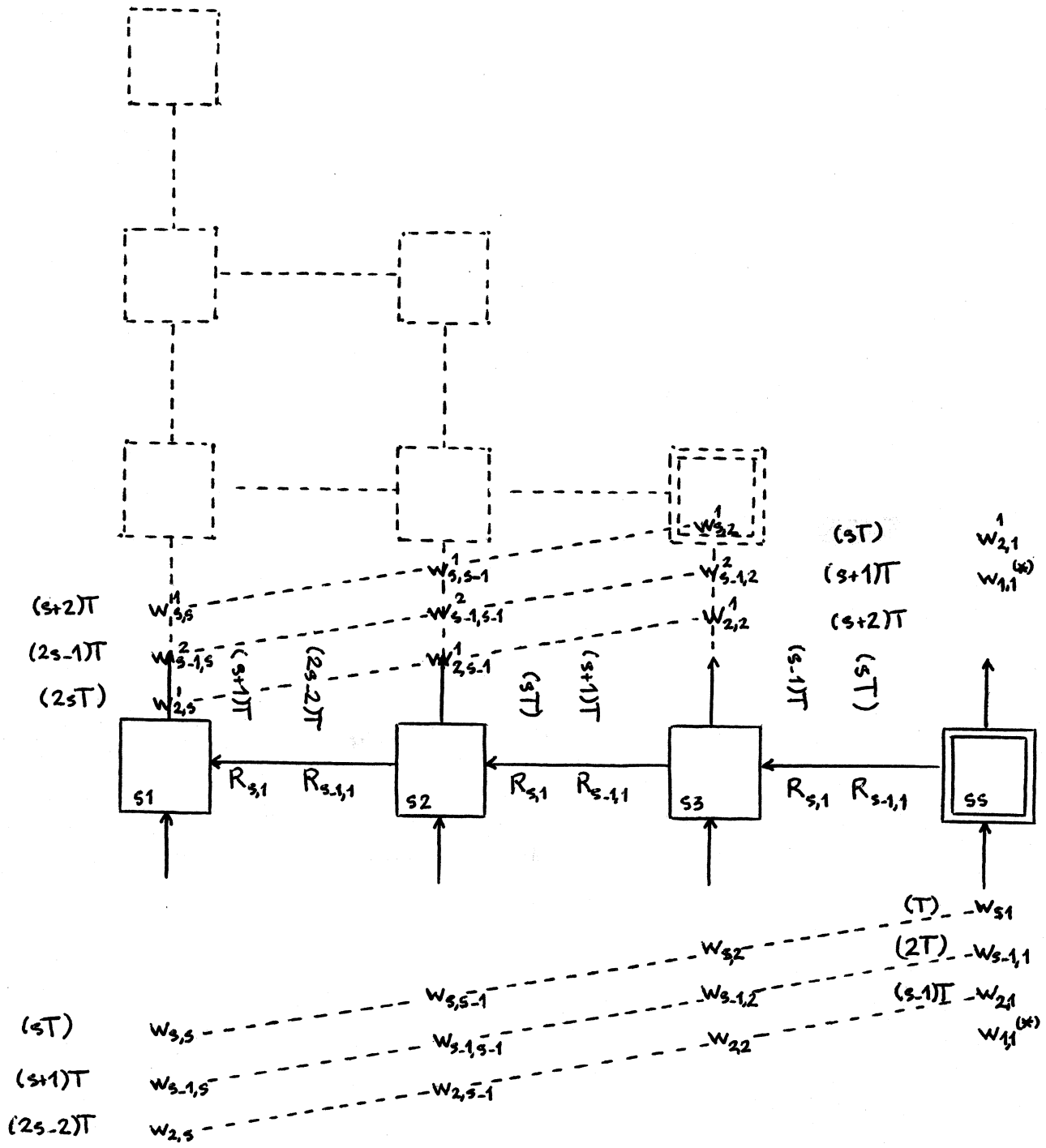


Figure 1: Moving of the $(s - 1)$ last rows of W' in the array for the matrix product $Q_1^T W'$ ($s = 4$).
 first forward pass: $Q_1^T = R_{s-1,1} R_{s,1}$.

The duration of the first forward pass is $2s$ time steps. The first backward pass can only begin from the instant $(2s + 1)T$ when we have computed the last element of $Q_1^T W'$.

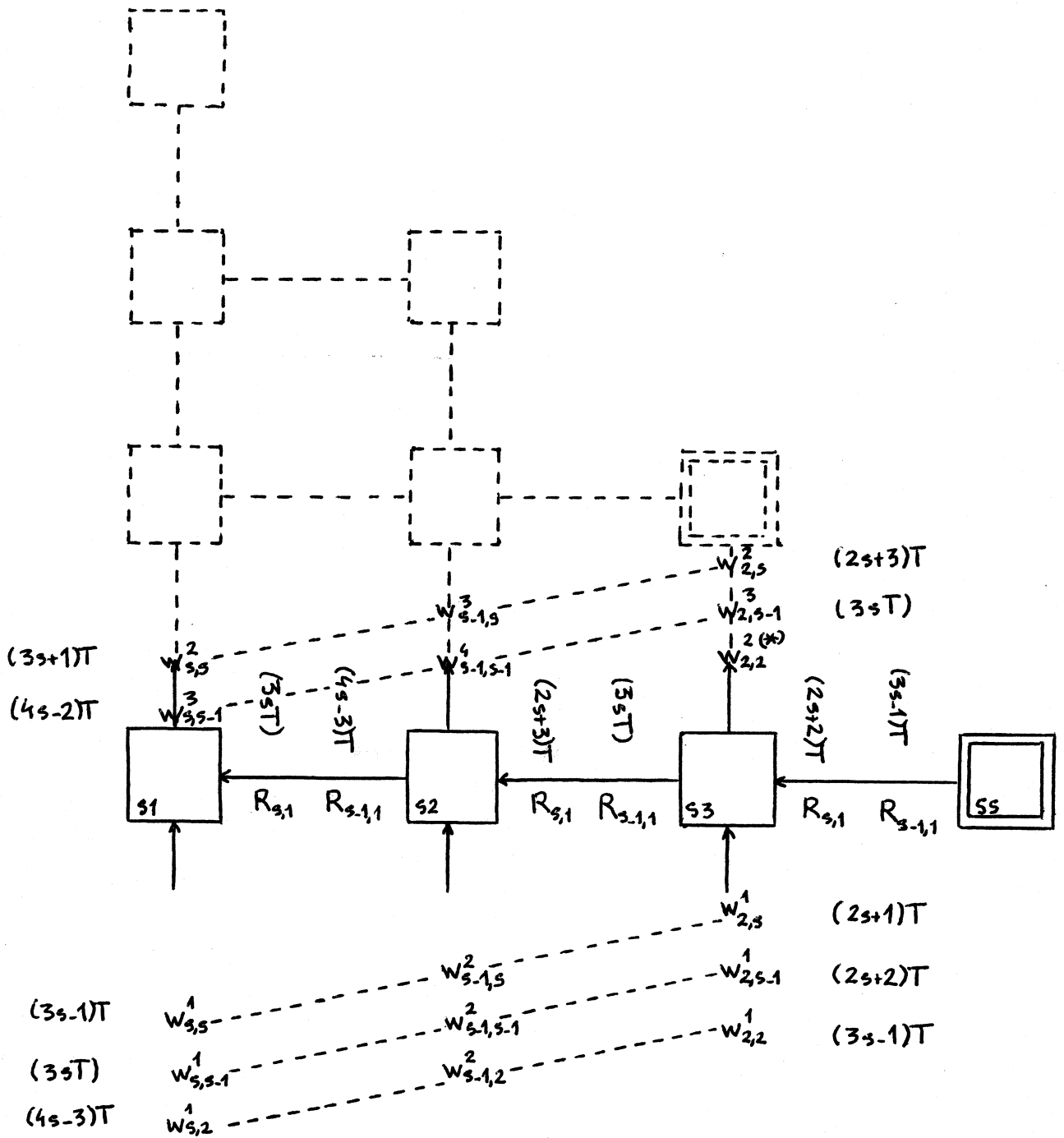


Figure 2: Moving of the $(s-1)$ last rows of $(Q_1^T W')^T$ in the array for the matrix product $Q_1^T (Q_1^T W')^T$ ($s=4$).
first backward pass.

The duration of the first backward pass is $2(s-1)$ time steps. The second forward pass can begin as soon as we get the first elements of $W^1 = Q_1^T W' Q_1$, so from the instant $(2s+3)T$.

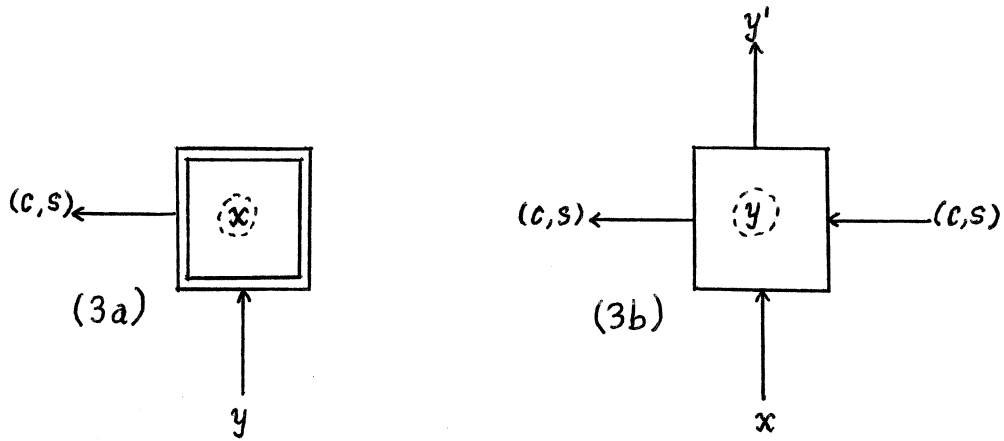


Figure 3: (3a) program of a diagonal cell: The rotation R which annihilates y with the pivot x is computed:

$$\begin{pmatrix} \rho \\ 0 \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \text{ where } \rho = (x^2 + y^2)^{1/2}, c = x/\rho, \\ s = y/\rho. \text{ Then } x = \rho \text{ is memorized in the array.}$$

(3b) program of a non diagonal cell: The rotation R is applied:

$$\begin{pmatrix} y \\ y' \end{pmatrix} = R \begin{pmatrix} x \\ y \end{pmatrix}. \text{ Then } y \text{ is memorized in the cell.}$$

Let us now evaluate the duration of the tridiagonalization. The first forward pass takes $2s$ time steps. The second forward pass begins at the instant $(2s+3)T = [2(s+1)+1]T$ and ends at the instant $[(2s+3)+2(s-1)-1]T = 4sT$. The third forward pass begins at the instant $[2(s+1)+2s+1]T$ and ends at the instant $[(4s+3)+2(s-2)-1]T$.

...

The $(s-2)$ th forward pass begins at the instant $[2(s+1)+2s+\dots+2((s+1)-(s-2)+2)+1]T = (2[(s+1)+s+\dots+5]+1)T$, the $(s-2)$ th forward and backward passes take 10 time steps. So the tridiagonalization ends at the instant $(2[(s+1)(s+2)/2-10]+10+1)T = [(s+1)(s+2)-9]T$ and it takes $((s+1)(s+2)-9)$ time steps.

Nevertheless for the tridiagonalization of a symmetric matrix of size s on his triangular array of $s(s+1)/2$ cells Schreiber [17],[18] assumes the existence of a systolic shifter which transposes the matrices between each forward and backward passes without delay time. It is possible to avoid the use of this shifter by creating in the memories of each non diagonal cell internal stacks (LIFO Last In First Out) allowing the storage of intermediate results of the matrices W^i between the forward and backward passes. Thus we create three internal stacks for the cells $i, i-1$ for $i = 3, \dots, s$, one internal stack for the cells $i, 1$ for $i = 3, \dots, s$ and two internal stacks for all the other non diagonal cells.

Then the computations are done as following: the forward passes are executed as described before, the matrix W^i is introduced vertically, column by column, but the elements of each column are memorized in the first LIFO of each non diagonal cell $s-i, j$ called $F_{s-i, j}$ instead of being sent in the shifter (see figure 5). Except the elements of the next to last column of W^i which are stored in $F_{s-i, s-i-1}$ and then reintroduced vertically in the cell $s-i, s-i-1$, the other columns j of W^i are reintroduced vertically in the cells $s-i, j$ and then horizontally to the right neighbor

cells $s - i, j + 1$ for the backward pass and stored in the second LIFOs $B_{s-i,j+1}$ from where they will be taken out for the next forward pass (see figure 6). These second LIFOs $B_{s-i,j+1}$ are linked up directly with the cells $s - i - 1, j$ of the $(s - i - 1)$ th row of the array which requires a slight modification of the array (see figures 5-6-7).

As the Systolimag machine, (see part I [16]), has no physical diagonal connections we can simulate them as following: see figures 4a, 4b. We can consider that there is no delay time when we use the diagram 4b, instead of the diagram 4a if we work by data flow. In fact in that case as soon as the processor $i - 1, j$ receives the data from the processor i, j it sends it to the processor $i - 1, j - 1$ without delay time. It follows that the backward passes are not executed in the same way as in the array of Schreiber. In his model the rotations computed by the diagonal cell $s - i, s - i$ during the $(i + 1)$ th forward pass are memorized and then resent in the $(s - i)$ th row of the array at the good time to execute the backward pass. These rotations reach every processor of the $(s - i)$ th row one by one and then are applied to the rows of $Q_{i+1}^T W^i$. In our model the rotations sent during the forward pass along the $(s - i)$ th row have to be memorized in the processors from their first utilization, the j th rotation memorized in the $(j + 1)$ th processor of the row in order to be reused during the backward pass. Without using systolic shifter, the times to execute the forward and backward passes are doubled. The loss of time comes from the transposition of the matrix W^i between every forward and backward passes.

The first forward and backward passes take $[2s + 2(s - 1)]$ time steps. The second forward and backward passes take $[2(s - 1) + 2(s - 2)]$ time steps.

...

The $(s - 2)$ th forward and backward passes take $[2(s - (s - 3)) + 2(s - (s - 2))]$ time steps. So the tridiagonalization takes $(2s^2 - 8)$ time steps.

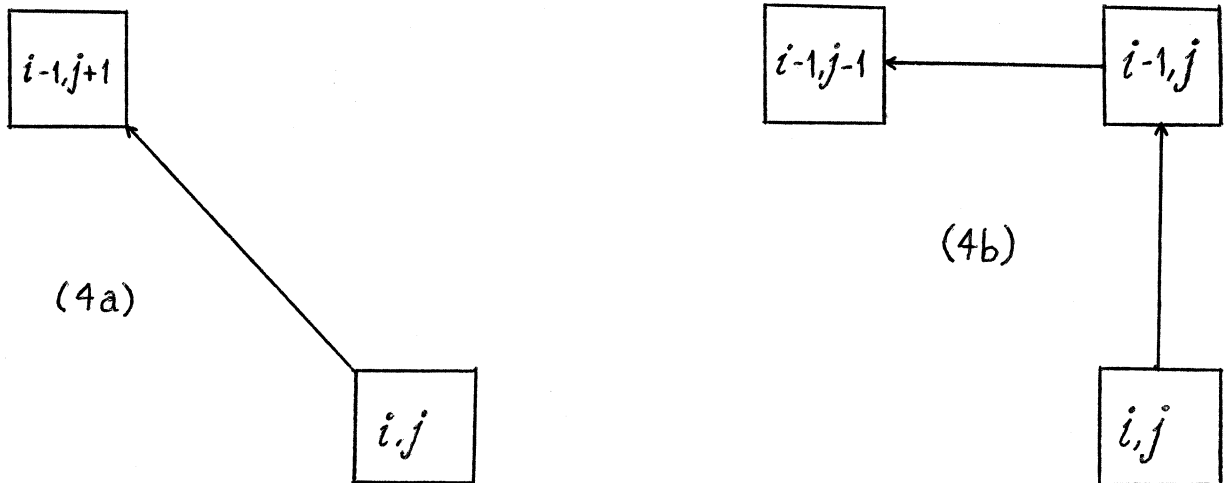


Figure 4: (4a) is replaced by (4b).

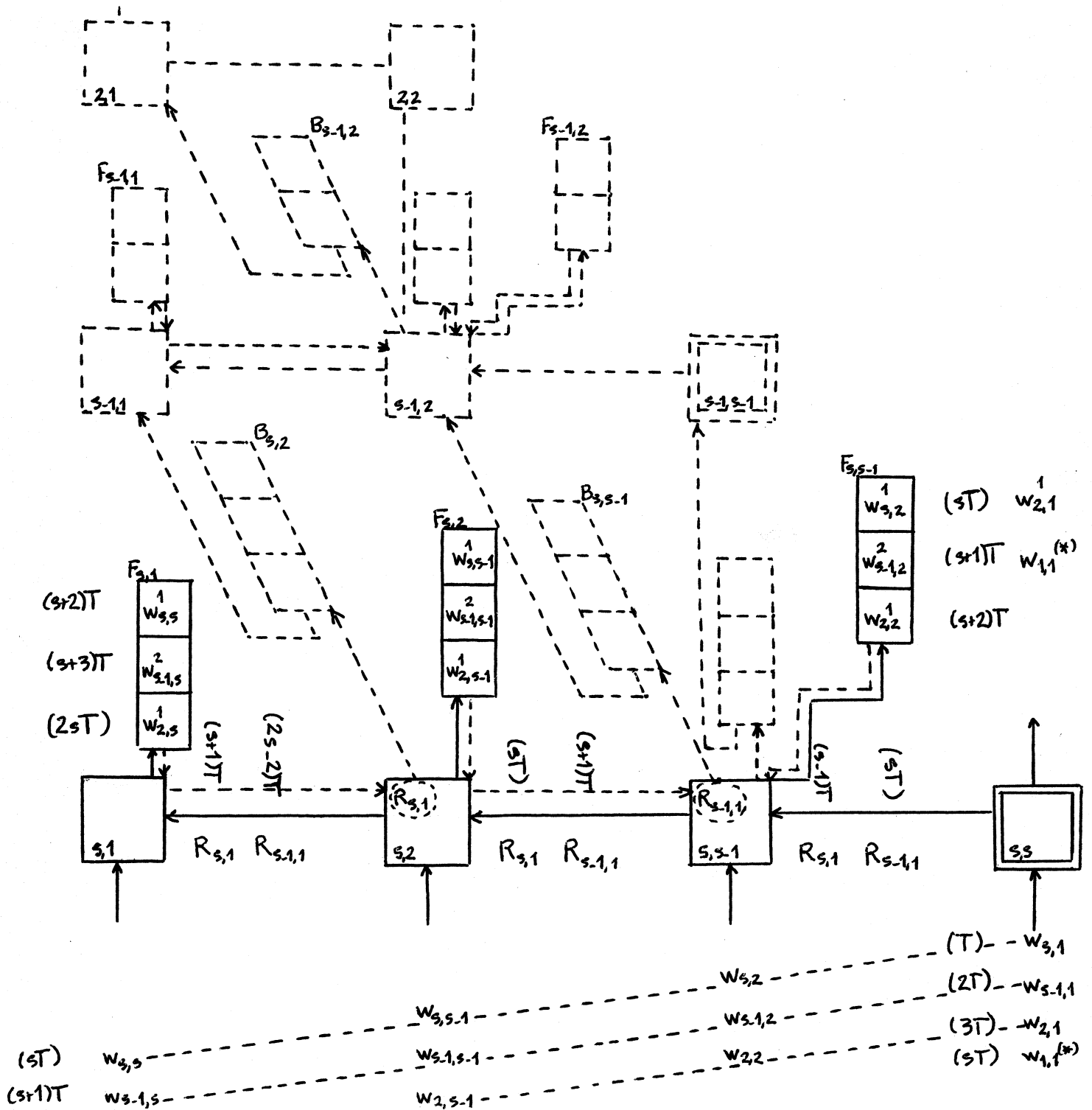


Figure 5: First forward pass: computation of the matrix product $Q_1^T W'$ ($s=4$). $Q_1^T = R_{s-1,1} R_{s,1}$.

The duration of the first forward pass is still $2s$ time steps. After applying the rotations $R_{s,1}$ and $R_{s-1,1}$ the columns of $Q_1^T W'$ are stored in the LIFOs $F_{s,i}$, $i = 1, \dots, s-1$. Only the used connections for the first forward pass are represented by continuous lines.

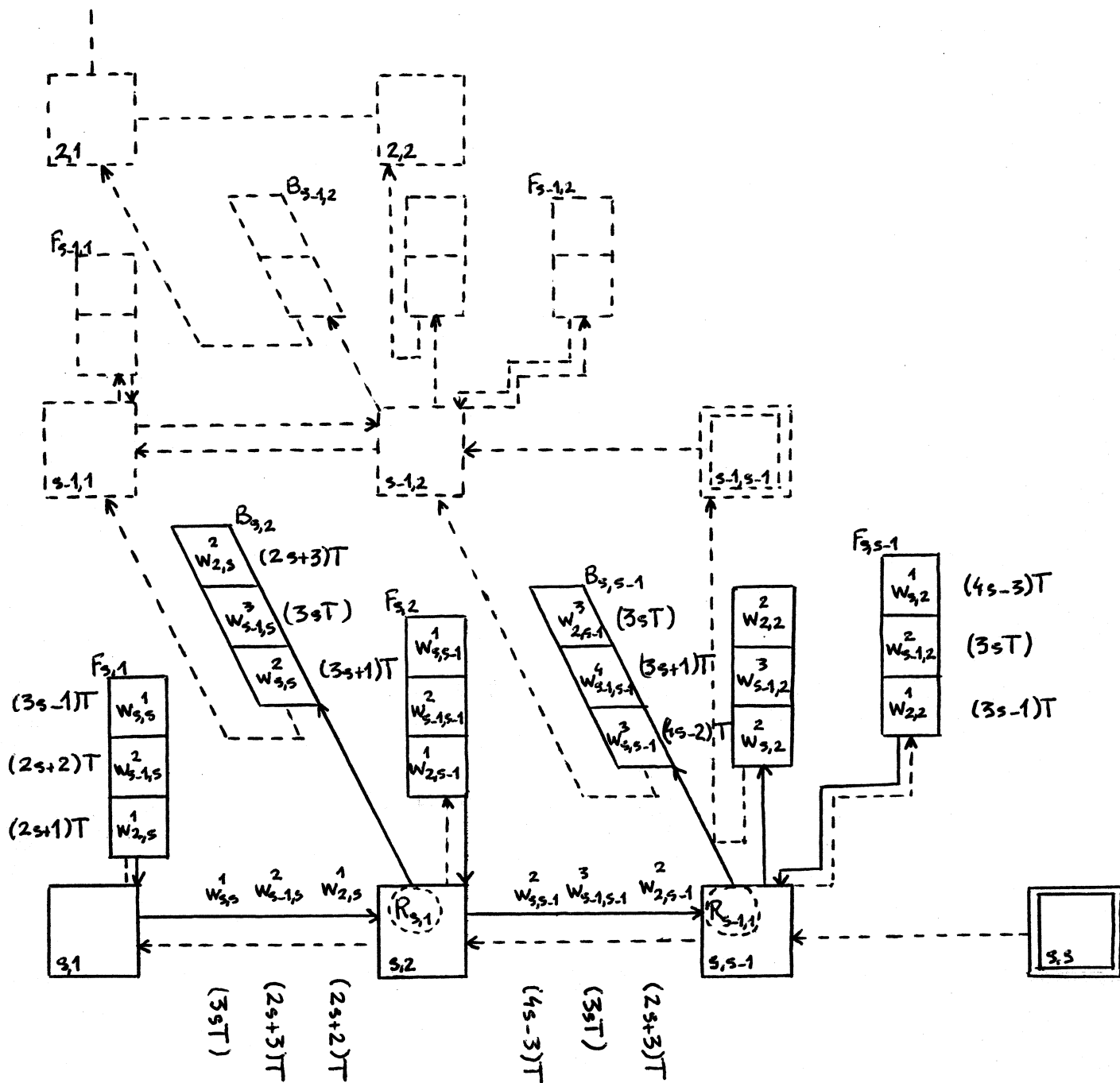


Figure 6: First backward pass: computation of the matrix product $Q_1^T(Q_1^T W')^T$.

The first backward pass begins at the instant $(2s + 1)T$ after the computation of the last element of $Q_1^T W'$ and takes $2(s - 1)$ time steps. The second forward pass can here only begin at the instant $(4s - 1)T$ after the computation of the last element of $Q_1^T W' Q_1$. Only the active parts during the first backward pass are represented by continuous line.

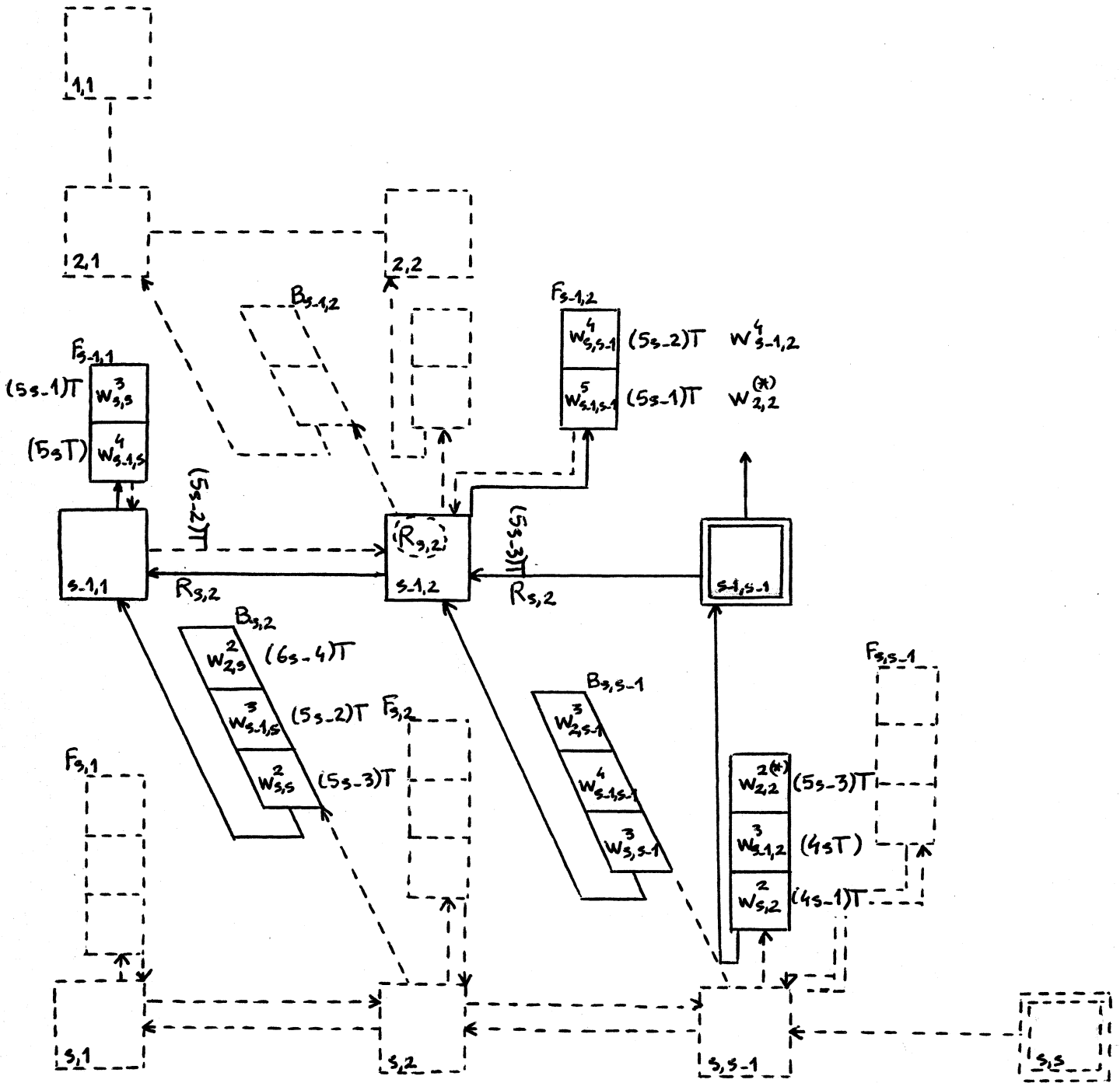


Figure 7: Second forward pass: computation of the matrix product $Q_2^T W^1$. $Q_2^T = R_{s,2}$.

The duration of the second forward pass is $2(s - 1)$ time steps.

3. Multisection method:

The bisection method can be applied to tridiagonal symmetric matrices. It allows to determine all the eigenvalues of the matrix in an interval fixed in advance.

As in Factorial Data Analysis the eigenvalues are all positive, we can determine a research interval for the eigenvalues from the beginning, for example $[0, \|W'\|_1]$ ($\|W\|_1 = \max_j \sum_i |w_{i,j}|$) We can often choose $[0, 1]$. This method relies essentially on the following theorem (see [6] pp120-123). Set down:

$$W_0 = \begin{pmatrix} b_1 & c_1 & & & & \\ c_1 & b_2 & c_2 & & & \\ & c_2 & & \ddots & & \\ & & & & \ddots & \\ & & & & & c_{s-1} \\ & & & & c_{s-1} & b_s \end{pmatrix}$$

We define the Sturm series of characteristic polynomials of the dominant submatrices of size j of W_0 : $p_0(\lambda) = 1, p_1(\lambda) = (b_1 - \lambda), \dots, p_j(\lambda) = (b_j - \lambda)p_{j-1}(\lambda) - c_{j-1}^2 p_{j-2}(\lambda)$. Set down $1 \leq j \leq s$ and $\mu \in R$, if we define:

$$\text{sgn} p_j(\mu) = \begin{cases} \text{sign}(p_j(\mu)), & \text{if } p_j(\mu) \neq 0 \\ \text{sign}(p_{j-1}(\mu)), & \text{if } p_j(\mu) = 0 \end{cases}$$

then the number $N(j; \mu)$ of sign changes of the set $E(j; \mu) = \{\text{sgn} p_0(\mu), \text{sgn} p_1(\mu), \dots, \text{sgn} p_j(\mu)\}$ is equal to the number of roots of the polynomial p_j which are strictly smaller than μ .

3.1. description of the method:

It is based on the ideas of Schreiber [21]. Suppose we are using $(s + 1)$ processors, that is to say for example, the last row of our triangular array plus one supplementary processor. The s first compute the series $(p_i(\mu))$ for $i = 1, \dots, s, \mu \in R$ and the last, stores the numbers $N(s; \mu)$ for different values of μ . The array works like this: we introduce a real μ into the first processor $s, 1$ which computes $p_1(\mu)$ and $N(1; \mu)$ and sends its results to the second processor $s, 2$ which, at its turn, computes $p_2(\mu)$ and $N(2; \mu)$ and so on until the sth processor s, s which computes $p_s(\mu)$ and $N(s; \mu)$. Every time we send a real μ in the row of $(s + 1)$ processors, we have to wait s time steps to get $N(s; \mu)$. The goal is then to use as best as possible every processor at every time step, and for that, to choose judiciously the values μ to introduce into the array in such a way that every processor is as active as possible during all the course of the method. In the other way we have to choose the multisection method as efficient as possible, that is to say, the one requiring as less arithmetic operations as possible. In order to do that let us compare different methods: assume we have $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_s$ contained in $[a_0, b_0]$ and that we search the i th eigenvalue λ_i .

1. bisection method: we set $c_0 = (a_0 + b_0)/2$ and we compute $N(s; c_0)$, if $N(s; c_0) \geq i$ then $\lambda_i \in [a_0, c_0]$, we set $a_1 = a_0$ and $b_1 = c_0$, else $\lambda_i \in [c_0, b_0]$ and we set $a_1 = c_0$ and $b_1 = b_0$. We set $c_1 = (a_1 + b_1)/2$ and we compute $N(s; c_1), \dots$. Thus we determine a series of intervals fitting into each other $[a_k, b_k], k \in N$ such that $\lambda_i \in [a_k, b_k]$ and $b_k - a_k = (b_0 - a_0)/2^k$.

2. multisection of order p : we set $c_j = a_0 + j(b_0 - a_0)/p$ and we compute $N(s; c_j)$ for $j = 0, \dots, p - 1$. If $N(s; c_j) - N(s; c_{j-1}) = 1$ then $\lambda_i \in [c_{j-1}, c_j]$, we set $a'_1 = c_{j-1}, b'_1 = c_j$ and $c_j = a'_1 + j(b'_1 - a'_1)/p$, and we compute $N(s; c_j)$ for $j = 0, \dots, p - 1, \dots$. Thus we determine a series of intervals fitting into each other $[a'_k, b'_k], k \in N$ such that $\lambda_i \in [a'_k, b'_k]$ and $b'_k - a'_k = (b_0 - a_0)/p^k$.

Assume we want to get the eigenvalue λ_i with a precision $PREC$, if at the k th step we set $LB[i] = (a_k + b_k)/2$ we get λ_i with a precision of $(a_k - b_k)/2$. For the method 1 we will need to do k computations of $N(s; c_i)$ with $k = \lceil \ln((b_0 - a_0)/2PREC) / \ln 2 \rceil$ and for the method 2, $p \times k$ computations of $N(s; c_i)$ with $k = \lceil \ln((b_0 - a_0)/2PREC) / \ln p \rceil$. The efficiency of the multisection method of order p related to the bisection method is then: $E_p = O(\log_2 p/p)$ [13]. The optimal

number p^* will have then to be as small as possible and also optimize the output of every processor (idle times minimized).

Let us look now at the parallelized method principle. We are only interested in Factorial Data Analysis in the greatest eigenvalues, for example those greater than a bound X_0 fixed in advance. We then cut the interval $[X_0, \|W'\|_1]$ into $(s+2)$ pieces of equal length $p_0 = (\|W'\|_1 - X_0)/(s+2)$ and we introduce successively the reals $X_i^0 = X_0 + ip_0$ at the instants $(i+1)T$ for $i = 0, \dots, s+2$ into the array. It then computes $N(s; X_i^0)$ for $i = 0, \dots, s+2$ and $NB[i] = N(s; X_i^0) - N(s; X_{i-1}^0)$ for $i = 1, \dots, s+2$. At the instant sT we get $N(s; X_0^0)$, we know then that there are $k := s - N(s; X_0^0)$ eigenvalues contained in $[X_0, \|W'\|_1]$. When the last X_i^0 is introduced into the array the first difference $NB[1]$ is computed; we can then, from this instant, define new values X_j^1 to introduce into the array for the second sweep (if $NB[1] \geq 1$). We choose, as for the first sweep, and for all the next sweeps, a set of about $(s+2)$ values X_j^1 such that they are as less interruptions as possible of the activity of the processors between two sweeps. We will have at most k subintervals $[X_{i-1}^0, X_i^0]$ containing eigenvalues; we thus choose to cut each of them into l equal parts with $l := \lceil (s+2)/k \rceil$. These l parts will be of length $p_1 = (\|W'\|_1 - X_0)/(s+2)l$.

We introduce successively the $X_j^1 = X_{i-1}^0 + jp_1$ for i such that $NB[i] > 0$ and for $j = 1, \dots, l-1$, into the array which computes the $N(s; X_j^1)$ as well as $NB1 := N(s; X_j^1) - N(s; X_{j-1}^1)$. After this second sweep we determine subintervals of length $(\|W'\|_1 - X_0)/(s+2)l$ containing eigenvalues. And so on for the following sweeps: we set $p_i = p_{i-1}/l$ and we cut the subintervals containing the eigenvalues into l equal parts of length p_i . After K sweeps the subintervals $[X_{i,inf}^K, X_{i,sup}^K]$ containing the eigenvalues will be of length $(\|W'\|_1 - X_0)/(s+2)l^{K-1}$. By setting down $LB[i] = (X_{i,inf}^K + X_{i,sup}^K)/2$ we will get the eigenvalue λ_i with a precision of $(\|W'\|_1 - X_0)/2(s+2)l^{K-1}$.

3.2. Number of arithmetic operations:

The computation of $N(s; \mu)$ requires: s integer additions, $(2s-1)$ real soustractions, $(4s-5)$ real multiplications and s sign comparisons. The first sweep requires $(s+2)$ computations of $N(s; \mu)$ and the following sweeps, at most $l \times k$ computations of $N(s; \mu)$ where $l \times k \simeq s+2$. So K sweeps will require about $K(s+2)$ computations of $N(s; \mu)$. To get the eigenvalues of W' with a precision $PREC$ we will have to take $K = \lceil \ln((\|W'\|_1 - X_0)/2kPREC)/\ln(s+2)/k \rceil$, and so to execute $O(s(s+2)/\ln(s+2))$ arithmetic operations.

3.3. Duration of the algorithm:

The first sweep requires $2(s+2)$ time steps, $(s+2)$ time steps for the introduction of the X_i^0 into the array, and $(s+2)$ for the computation of the $N(s; X_i^0)$. The following sweeps require $k \times l$ time steps, that is to say about $(s+2)$ time steps for the computation of the $N(s; X_i^p)$. So our method requires $(K-1) \times k \times l + 2(s+2) \simeq (K+1)(s+2)$ time steps. Finally our multisection method can be executed in $O((s+2)/\ln(s+2))$ time steps with $O(s(s+2)/\ln(s+2))$ arithmetic operations.

The figure 8 shows us the array for our method, as well as the sending of the data X_i^0 for the first sweep. The processor s, j for $2 \leq j \leq s$ computes the j th characteristic polynomial $p_j(X_i^0)$ for $i = 0, \dots, s+2$ from the characteristic polynomials $p_{j-1}(X_i^0)$ and $p_{j-2}(X_i^0)$ and the elements c_{j-1} and b_j of the matrix W_0 (W_0 is the matrix W' tridiagonalized); c_{j-1} and b_j are memorized in the cell for other computations of characteristic polynomials $p_j(X_i^m)$ for $m = 1, 2, \dots$. The processor s, j also updates the variable $NNCS$ which computes the number of sign changes of the series $(p_j(X_i^0))$, $j = 0, \dots, s$. The last processor $s, s+1$ computes the $NB[i] = N(s; X_i^0) - N(s; X_{i-1}^0)$ for $i = 1, \dots, s+2$ and determines the intervals $[X_{i-1}^0, X_i^0]$ containing eigenvalues. It fixes then a new set of values $X_j^1 = X_{i-1}^0 + jp_1$ for $j = 1, \dots, l-1$ to test during the second sweep.

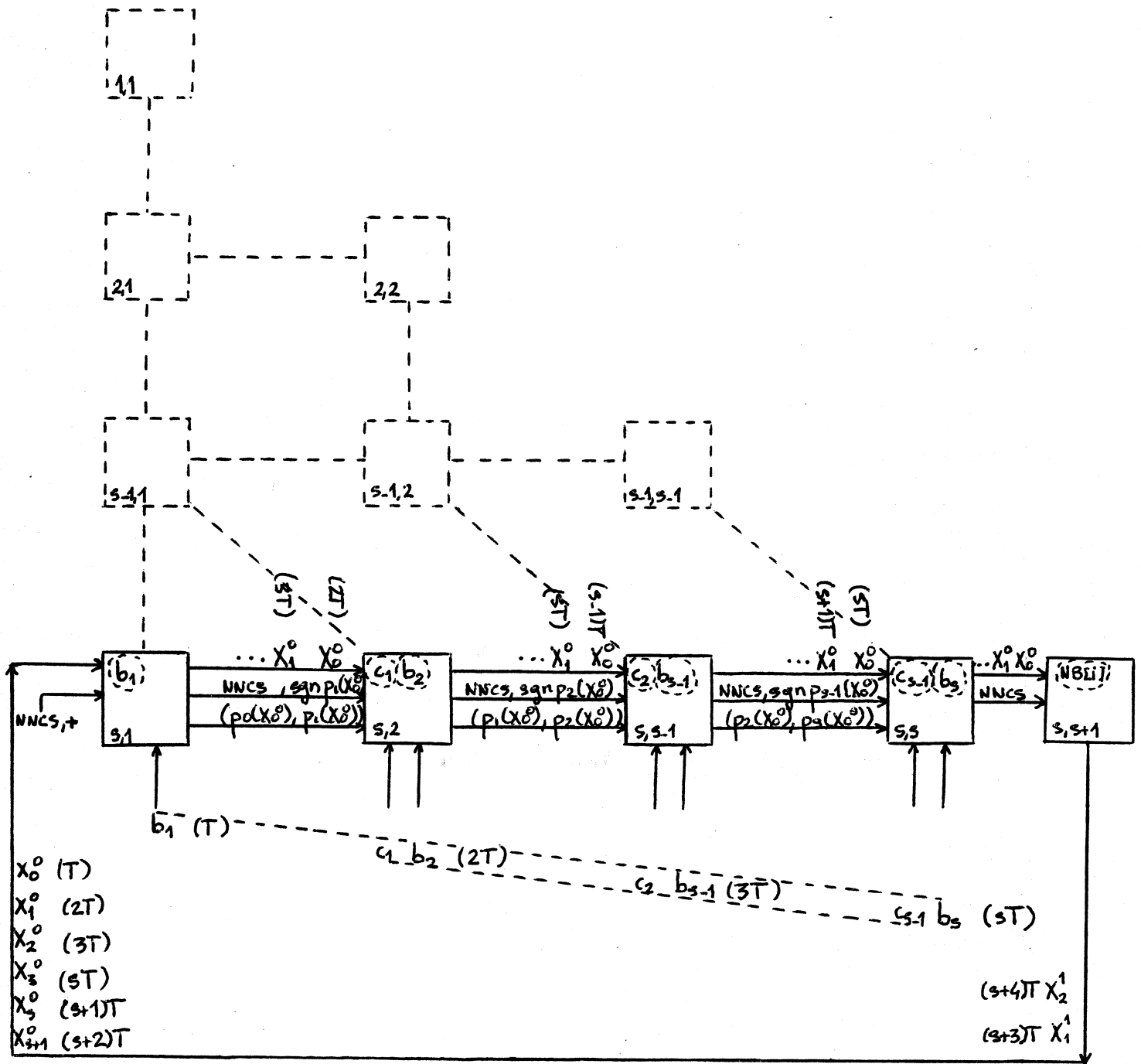


Figure 8: Systolic array for the multisection method of order $(s+2)$ ($s=4$).

The first values of the set $X_j^1, j=1, \dots, l-1$ are defined from the instant $(s+3)T$ only if $NB[i] > 0$, else we have to wait the instant when we find the first i such that $NB[i] > 0$.

4. Inverse iteration method:

After the tridiagonalization of W' into W_0 with the product of Givens rotations P : $W_0 = PW'P^T$, and after the computation of the eigenvalues of W' with the multisection method of order $(s+2)$ on W_0 , it remains to determine the eigenvectors of W' .

We use the inverse iteration method [4] which requires to know quite accurate approximations of the eigenvalues to get a fast convergence. Assume σ is an approximation of the eigenvalue λ , then to compute the eigenvector x associated with λ , we build the series $(q_k)_{k \in \mathbb{N}}$ of unitary vectors converging on x and defined by:

- $q_0 = u / \|u\|_2$.
- resolution of $(W' - \sigma I)z_k = q_{k-1}$, $k = 1, 2, \dots$
- $q_k = z_k / \|z_k\|_2$.

It is the power method applied to $(W' - \sigma I)^{-1}$, the convergence factor is then: $|\lambda - \sigma| / \max_{\lambda \neq \lambda_i} (|\lambda_i - \sigma|)$. We apply here this method to the tridiagonalized matrix W_0 and we use as eigenvalues approximations, the values computed by the multisection method. The tridiagonal system $(W_0 - \sigma I)z_k = q_{k-1}$ can be solved by LU factorization or by QR factorization. As σ is very close to λ the matrix $(W_0 - \sigma I)$ is almost singular, and we can not use the LU factorization without partial pivoting for stability and pivoting does not parallelize easily.

So first we determine a product of $(s-1)$ Givens rotations Q^T such that $Q^T(W_0 - \sigma I) = R$ is upper triangular and we compute $u_{k-1} = Q^T q_{k-1}$ and then, we solve the triangular system $Rz_k = u_{k-1}$. The QR factorization is not done column by column from the triangular array of $s(s+1)/2$ processors of Gentleman and Kung [7], but diagonal by diagonal from the array of Heller and Ipsen [9] for band matrices.

In this array composed of $w \times q$ processors where w is the band width of the matrix and q , the number of subdiagonals. The matrix is introduced diagonal by diagonal and the QR factorization is done in $2(s+(q-1))$ time steps (instead of $3s$ for the array of Gentleman and Kung). Every row of the array is composed of one cell which creates the rotations and that the diagonal of elements to be annihilated crosses and of $(w-1)$ cells which apply and broadcast the rotations to the other diagonals. The computation of the right hand member $u_{k-1} = Q^T q_{k-1}$ can be done on the same array if we add q supplementary cells.

As $(W_0 - \sigma I)$ is tridiagonal, $w = 3$ and $q = 1$. So we will only need 4 processors, 3 for the computation of $Q^T(W_0 - \sigma I) = R$ (1) and one for the computation of $Q^T q_{k-1} = u_{k-1}$ (2). R will have then 2 upper diagonals and the band triangular system $Rz_k = u_{k-1}$ (3) will be solved by 3 processors linearly connected as described by Kung and Leiserson [12]. For the operations (1) and (2) we can use for example the fourth row of our triangular array composed of 4 processors, and for the operation (3), the third row. The figure 9 represents the QR factorization of $(W_0 - \sigma I)$. $(W_0 - \sigma I) = (c_{i-1}, (b_i - \sigma), c_i)_{1 \leq i \leq s}$ as well as $q_{k-1} = (q_i)_{1 \leq i \leq s}$ are introduced into the fourth row of the array and the matrix $R = (r_{i,i}, r_{i,i+1}, r_{i,i+2})_{1 \leq i \leq s-2}$ as well as the right hand side $u_{k-1} = (u_i)_{1 \leq i \leq s}$ are stored into internal stacks (LIFO) of the four processors. The operation (3) described in the figure 10 is executed from the data from the LIFOs. As we get the last value u_s at the instant $(2s+2)T$, the resolution of the triangular system $Rz_k = u_{k-1}$ can begin at the instant $(2s+3)T$ and we get z_k after $(2s+1)$ time steps.

The speed of convergence of the series $(q_k)_{k \in \mathbb{N}}$ to the eigenvector depends on the accuracy of the eigenvalue. In fact if σ is such that $(W' - \sigma I + E)$ is singular with $\|E\|_2 = \epsilon$ then it exists at least one vector q such that the solution z of $(W' - \sigma I)z = q$ determines an eigenvector z of W' exact at the precision ϵ (that is to say such that z is an eigenvector of $W' + F$ with $\|F\|_2 = O(\epsilon)$; see p 173 of [4]). So we can get from the approximation σ of λ , with one step of the inverse iteration method, an approximation of the eigenvector of the same order of accuracy in $(4s+3)$ time steps.

We can reduce this execution time to an average of $2s$ time steps per eigenvector by adding a LIFO to each of the four cells of the fourth row. Another resolution of tridiagonal system $(W_0 - \sigma'I)z'_1 = q_0$ can then begin at the instant $2sT$, the new matrix $R' = Q'^T(W_0 - \sigma'I)$ and the new right hand member $u_0 = Q'^T q_0$ being stored in the new LIFOs.

If q_1 is an eigenvector of W_0 , then $P^T q_1$ is an eigenvector of W' . It remains then to compute this matrix-vector product. We can get the matrix P by introducing the identity matrix I_s just after the matrix W' during its tridiagonalization (from the instant $(s+1)T$: see figures 5-6-7) and by only applying the forward passes to I_s (resending of the rotations memorized in the diagonal processors along the rows from the right to the left). The product $P^T q_1$ can be done in $2s$ time steps with s processors linearly connected. We have only to introduce the matrix P row by row and the vector q coordinate by coordinate (the i th row and the i th coordinate entering the i th processor).

5. Conclusion:

The study of systolic algorithms for matrix computations on our triangular systolic array SARDA has showed us the possibility to use a systolic array for a great number of systolic algorithms without having to modify the interconnection structure between the different operations. In fact the processors have only to have a memory large enough to allow, during the programming of the nodes, the creation of a great number of processes and internal or external FIFOs and LIFOs.

On the other hand the asynchronous programming of the algorithms has allowed us to simplify the supervision operations of the host system and to save time. These operations are now reduced to sending the data and receiving the results. Nevertheless we have to replace the global synchronization of the system by local controls in each cell such as tests or loops counters (see [15]).

However this study has above all pointed out the necessity of adding to the systolic array another computational system to process the data efficiently, that is to say, receive the results and store the intermediate computations such that the systolic array be as active as possible. In particular the system should be able to store automatically the matrices by rows, columns or blocks and to transpose the matrices asynchronously. In that way the sending of data and receiving of results could be done asynchronously as the operations inside the array.

On page 18 a summary table describes the algorithms we have studied for the symmetric eigenvalue problem. First, to determine the eigenvalues (Jacobi or tridiagonalization + multisection), second, to determine the eigenvectors (inverse iteration method).

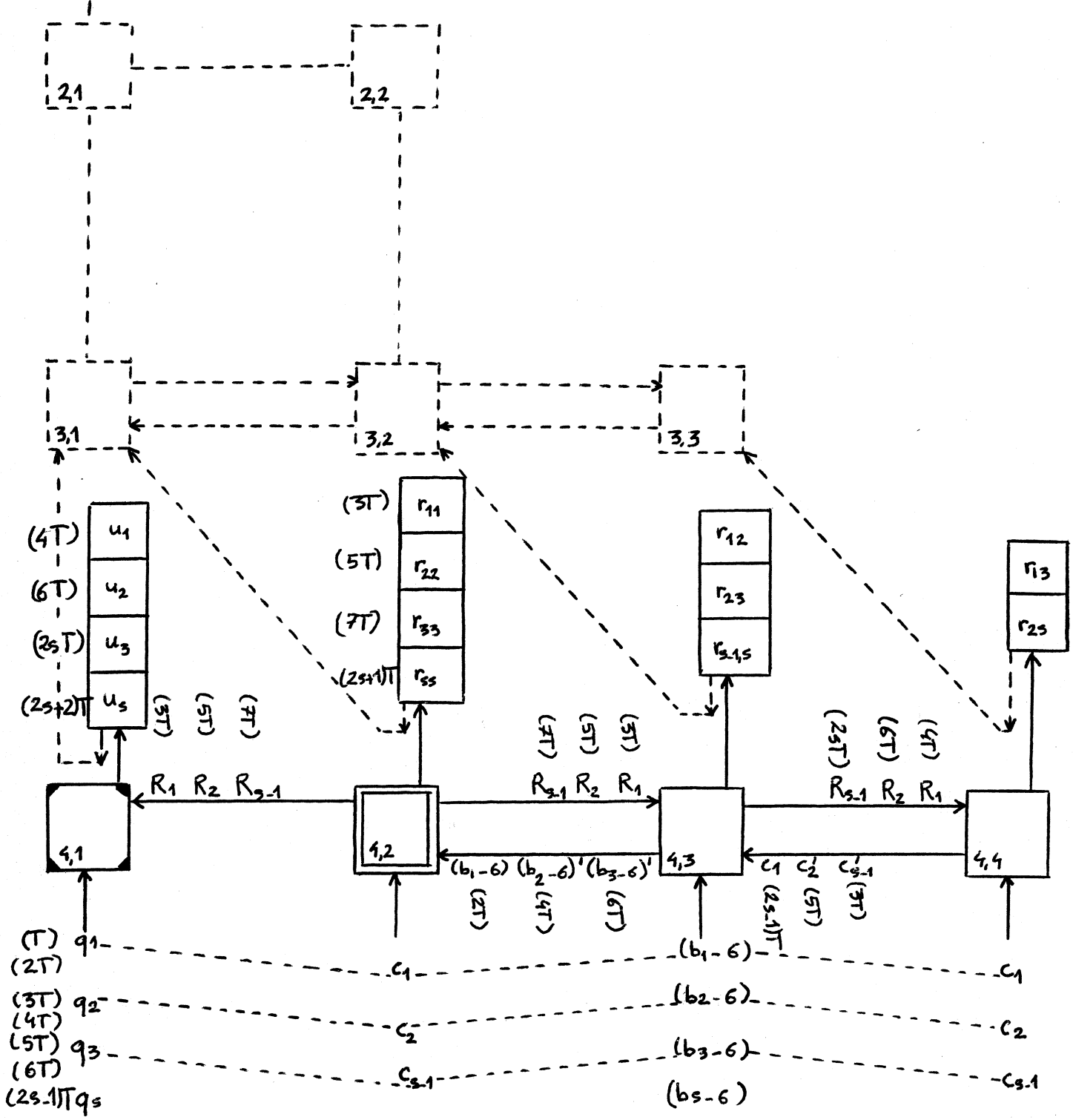


Figure 9: QR factorization of $(W_0 - \sigma I)$: computation of Q such that $Q^T(W_0 - \sigma I) = R$ and computation of $u_{k-1} = Q^T q_{k-1}$ ($s = 4$).

These computations require $(2s + 2)$ time steps.

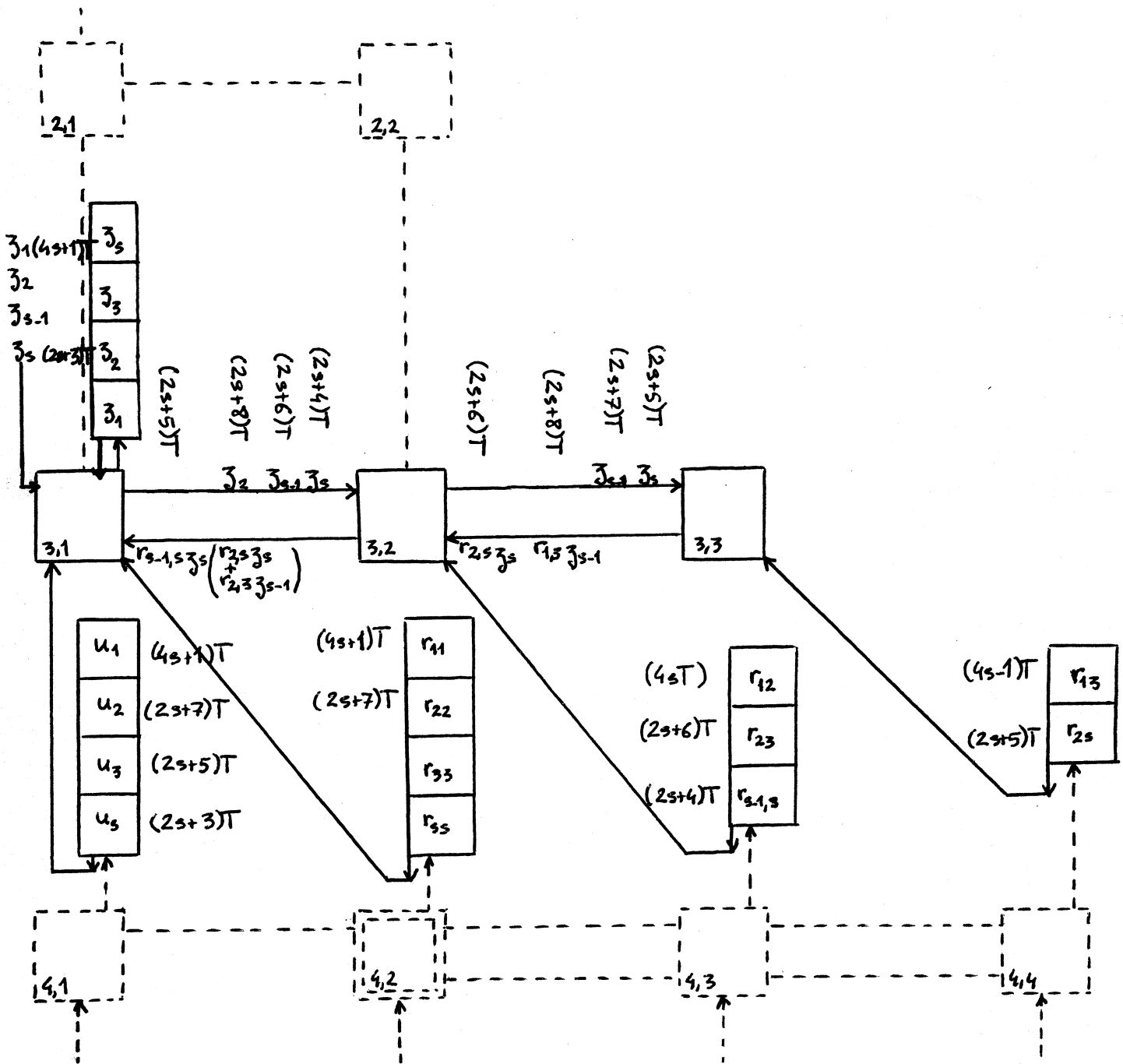


Figure 10: Resolution of the band triangular system $Rz_k = u_{k-1}$ ($s = 4$).

The processor 3,1 computes the norm of the vector $z_k = (z_i)_{1 \leq i \leq s}$ as further the coordinates z_i enter its internal stack. We get z_1 at the instant $(4s + 2)T$ and $q_k = z_k / \|z_k\|_2$ at the instant $(4s + 3)T$.


algorithm	shape and connections of the array	number of time steps	authors and special features
Jacobi method on a symmetric matrix of size $s \times s$	square array of $s^2/4$ processors with octogonal connections	$3s \log s$ time steps	Brent-Luk, Schreiber [1],[20]: utilization of cyclic permutations
tridiagonalization of a symmetric matrix of size $s \times s$	triangular array of $s(s+1)/2$ processors with orthogonal connections + a systolic shifter [fig 1-3]	$(s^2 + 3s - 7)$ time steps	Schreiber [17],[18]: is done column by column and row by row
	triangular array of $s(s+1)/2$ processors with horizontal and diagonal connections and internal LIFOs [fig 4-6]	$(2s^2 - 8)$ time steps	Porta* [15]: is done column by column and row by row
LR-Cholesky method on a tridiagonal matrix of size $s \times s$	triangular array of $s(s+1)/2$ processors with orthogonal connections	very slow convergence. $2s$ time steps per iteration	
multisection method of order $(s+2)$ on a tridiagonal matrix of size $s \times s$	$(s+2)$ processors linearly connected [fig 8]	$O((s+2)/\log(s+2))$ time steps	Schreiber,Porta* [19],[13]
inverse iteration method on a tridiagonal matrix of size $s \times s$	4 processors linearly connected with internal LIFOs + 3 processors linearly connected and diagonally connected [fig 9-10]	average of $4s$ time steps per eigenvector (and per iteration)	Heller-Ipsen,Kung-Leiserson,Porta* [9],[12],[15]
singular value decomposition (SVD) of the Cholesky factor of W' of size $s \times s$	$s/2$ processors linearly connected with 2 FIFOs per processor	$3s$ time steps + $O(s(s-1)\log s)$ time steps	Schreiber,Brent-Luk [19],[1] Cholesky factorization + Hestenes method
	square array of $s^2/4$ processors with octogonal connections	$3s$ time steps + $O(s \log s)$ time steps	Brent-Luk-Van Loan, Luk [3],[14]:Cholesky factorization + SVD-Jacobi method

Table 1: Eigenlements computations

References

- [1] R.P.Brent, F.T.Luk (1982) *A systolic architecture for almost linear time solution of the symmetric eigenvalue problem*. Cornell Dpt of Computer Science-Technical Report 82-525.
- [2] R.P.Brent, F.T.Luk (1985) *The Solution of Singular Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays*. SIAM J.Sci.Stat. Comput. Vol 6 No 1 pp 69-84.
- [3] R.P.Brent, F.T.Luk, C.Van Loan (1985) *Computation of the Singular Value Decomposition Using Mesh-Connected Processors*. Journal of VLSI and Computer Systems, Vol 1 No 3 pp 242-270.
- [4] F.Chatelin (1986) *Valeurs propres de matrices Etude IBM F-096 Centre Scientifique de Paris*. to appear at Masson, Paris.
- [5] F.Chatelin, T.Porta (1987) *Numerical Analysis for Factorial Data Analysis. Part II. Etude IBM F-110 Centre Scientifique de Paris*.
- [6] P.G.Ciarlet (1982) *Introduction à l'analyse numérique matricielle et à l'optimisation*. Masson, Paris.
- [7] W.M.Gentleman, H.T.Kung (1981) *Matrix triangularization by systolic arrays*. Proc. SPIE Vol 298 Real Time Processing IV pp 19-26.
- [8] D.E.Heller, I.C.F.Ipsen (1982) *Systolic Networks for Orthogonal Equivalence Transformations and their Applications*. Conference on Advanced Research in VLSI M.I.T.
- [9] D.E.Heller, I.C.F.Ipsen (1983) *Systolic networks for orthogonal decompositions*. SIAM J.Sci. Stat. Comput. Vol 4 No 2 pp 261-269.
- [10] I.C.F.Ipsen, Y.Saad (1985) *The Impact of Parallel Architectures on the Solution of Eigenvalue Problem*. Research Report YALEU/DCS/R-R 444.
- [11] L.Johnsson (1982) *A Computational Array for the QR method*. Conference on Advanced Research in VLSI M.I.T.
- [12] H.T.Kung, C.E.Leiserson (1981) *Highly concurrent systems in Introduction to VLSI Systems*. C.A.Mead-L.A Conway Eds. pp 271-289.
- [13] S.S.Lo, B.Philippe, A.Sameh (1985) *A parallel algorithm for the real symmetric tridiagonal eigenvalue problem*. CSRD Technical Report Univ. Ill., Urbana Champaign.
- [14] F.T.Luk (1986) *Architectures for Computing Eigenvalues and SVDs*. Proc. SPIE Vol 614 Highly Parallel Signal Processing Architectures.
- [15] T.Porta, F.Chatelin (1986) *Un réseau systolique programmable pour l'Analyse Factorielle des Données*. Etude IBM F-103 Centre Scientifique de Paris.
- [16] T.Porta (1987) *A Programmable Systolic Array for Factorial Data Analysis. Part I: Matrix Computations*. Research Report YALEU/DCS/R-542.
- [17] R.Schreiber (1982 a) *Systolic Array for Eigenvalues*. Proceedings of the Inter American Workshop in Numerical Methods. Springer Verlag.
- [18] R.Schreiber (1982 b) *Systolic Arrays for Eigenvalue Computation*. Real Time Signal Processing V Proc. SPIE Vol 341 Society of Photooptical Instrumentation Engineers, Bellingham, Washington pp 27-34.

[19]R.Schreiber (1983) *On the systolic arrays of Brent,Luk and Van Loan for the symmetric eigenvalue and singular value problem.* Dpt of Numerical Analysis and Computing Science.Royal Institute of Technology,Stockholm.Report TRITA-NA-8311.

[20]R.Schreiber(1984)*Solving Eigenvalue and Singular Value Problems on an Undersized Systolic Array.*Computer Science Dpt.Stanford University Stanford CA 94305 Submitted to SIAM J.Sci.Stat.Comput.

[21]R.Schreiber(1984)*Computing Generalized Inverses and Eigenvalues of Symmetric Matrices Using Systolic Arrays.*in Computing Methods in Applied Sciences and Engineering,VI Glowinski and Lions Eds.Elsevier Science Publishers BV (North Holland) pp 285-294.

[22]J.H.Wilkinson (1965) *The Algebraic Eigenvalue Problem.* Clarendon.Oxford.