Abstract:

In this paper we present several systolic algorithms for Factorial Data Analysis:matrix products of several types such as XX^T where X is a rectangular matrix of size $k \times n$,RX where R is upper triangular of size k,AB where A and B are square dense matrices of size k,Cholesky factorizations and triangular matrix inversions.

All these algorithms are built to run efficiently on the same asynchronous MIMD triangular systolic array with orthogonal connections:**SARDA**(Systolic Array for Data Analysis).

A Programmable Systolic Array for Factorial Data Analysis Part I:Matrix Computations

Tiba Porta

Research Report YALEU/DCS/RR-542
June 1987

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

1. Introduction

The computations needed for Factorial Data Analysis come down to a unique computation kernel [2,3]. For each of the factorial techniques there is a preliminary computation to do on the matrix of observations to obtain the $k \times n$ matrix of data X and possibly on the A, (respectively B),symmetric positive definite matrix of size n,(respectively k),which represents the chosen norm on R^n,(respectively R^k). Then we compute the Cholesky factorization of A and B when they are not diagonal:we define R_A and R_B upper triangular of sizes n and k such that $A = R_A^T R_A$ and $B = R_B^T R_B$ (where T stands for transposition).
After that we let $E = R_B X R_A^T$ and $W = EE^T$ and we define the eigenvalue decomposition of W:(λ_i, w_i) for $i = 1, \ldots, k$ such that $w_i^T w_i = 1$. Finally we compute $f_i = R_B^T w_i$ and $g_i = \lambda_i^{-1/2} A X^T f_i$.

We can therefore restrict ourselves to the following set of computations:

1. matrix product $\Lambda = XX^T$ where X is $k \times n$.
2. Cholesky factorization $\Lambda = R_\Lambda^T R_\Lambda$ of order k.
3. triangular matrix inversion R_Λ^{-1}.
4. matrix products RX where R is upper triangular of size k and X rectangular of size $k \times n$ and matrix products AB where A and B are both square of order k.
5. dominant eigenvalues computation of the symmetric positive definite matrix W of order k.

The purpose of that work was to devise efficient systolic algorithms for the computations listed above from 1 to 5 all running on the same programmable systolic array:the **Systolimag machine** built by Gerard Chevalier[11]. This machine composed of 9 processors allows to create or to simulate experimental,two-dimensional systolic arrays of sizes from 2 up to 12:**SARDA**(Systolic Array for Factorial Data Analysis).
Each processor has a microprocessor $MC6809$,an arithmetic coprocessor $AMD951$,a ROM(Read Only Memory) of 4Kbytes,a RAM(Random Access Memory) of up to 16 Kbytes and 4 communication channels to let the processor communicate with its 4 nearest neighbors from North,South,East, and West[11].
We wanted to use the Systolimag machine as an asynchronous MIMD(Multiple Instruction Multiple Data) machine.That is to say to use the fact that the processors are programmable and they don't have to do all the same thing and that their programs are ruled by data flow.

As our systolic array had to be of a fixed size s ($s \leq 12$) we realized the computations 1 to 5 with matrices of size $s \times s$ or $s \times n$ by block partitioning (see [10,3]). We wanted also to keep the same structure for all our algorithms in order not to lose time by changing the connections with the nearest neighbors between two algorithms.
We chose a triangular array with orthogonal connections (four nearest neighbors north-east-south-west) as our stucture to deal with Factorial Data Analysis.The first reason is because Data Analysis involve mainly symmetric or triangular matrices so we only need to use $s(s+1)/2$ processors (one per matrix element) instead of s^2.The second reason is that this structure is especially well suited for implementing 1 and 2 in cascade [10,12,13].The orthogonal connections are the simpliest to build because every processor has 4 "natural" (hardware) communication channels with its neighbors and they are mainly used for the computations 1 to 5.

In section 2 we present the systolic implementation of the matrix product $\Lambda = XX^T$ with X rectangular of size $s \times n$ followed in cascade by the Cholesky factorization of size s of Λ: $\Lambda = R^T R$.In section 3 we detaile the upper triangular matrix inversion with R stored in the array and the matrix products described as computations 4.

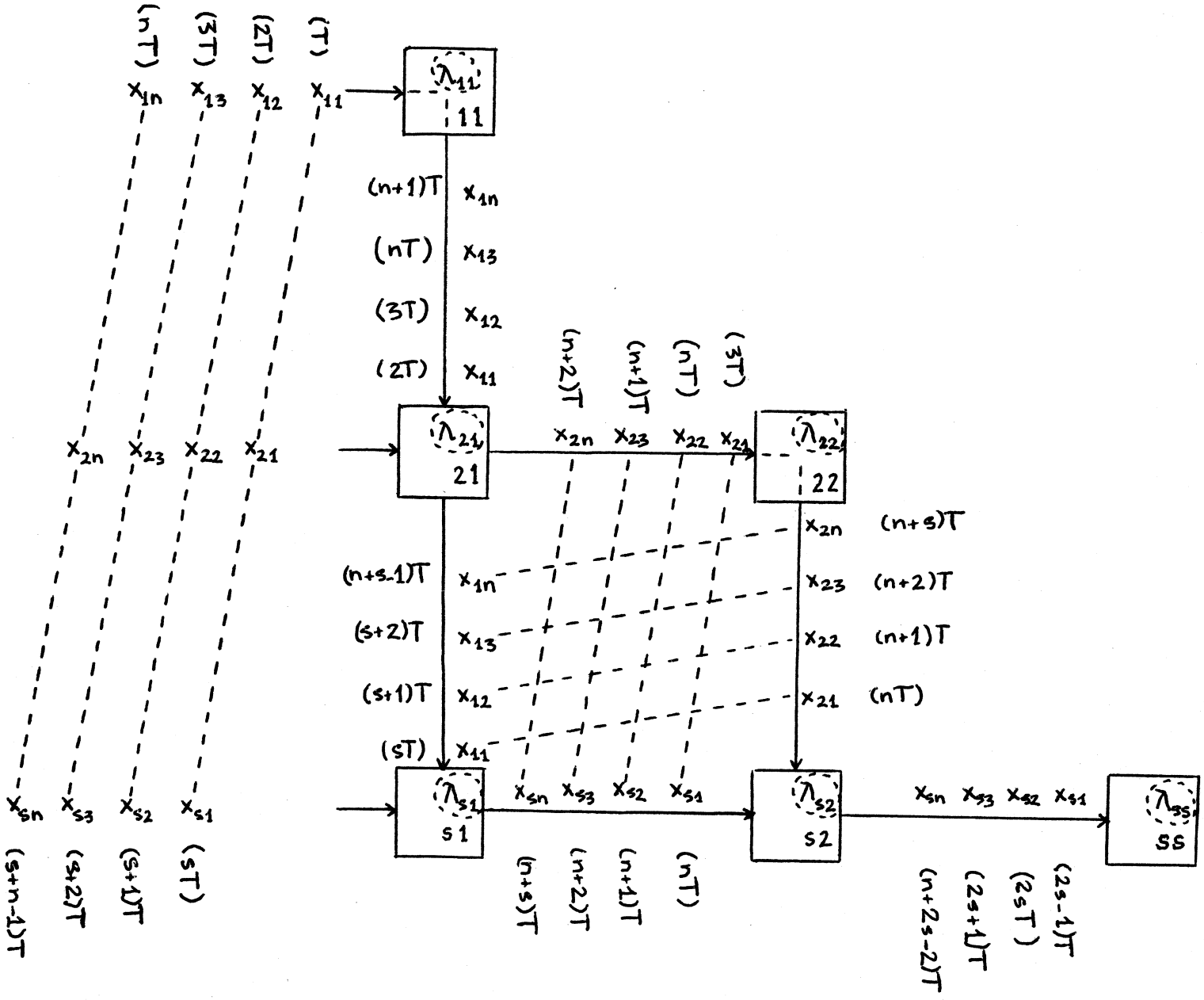

Figure 1: progression of the matrix X in the array for the computation of $\Lambda = XX^T$.

2. Systolic implementation of $\Lambda = XX^T$ and Cholesky factorization of Λ,$\Lambda = R^T R$:

2.1. matrix product $\Lambda = XX^T$ with X of size $s \times n$:

Set down $\Lambda = (\lambda_{i,j})_{1 \leq i,j \leq s}$.We have: $\lambda_{i,j} = \sum_{l=1}^{l=n} x_{i,l} x_{j,l}$

The processor ij computes $\lambda_{i,j} = \lambda_{j,i}$. Here are now the detail of the operations carried out by the array in the case $s = 3$ and $n = 4$.The figure 1 presents the progression of the matrix X in the array with the time steps which correspond to.At each clock cycle T,each processor receiving input data performs a multiplication on them, adds this result to its memory contents and broadcasts the data on its outputs.There are two types of cells,the diagonal cells which only receive one input data and the others which receive two.It corresponds to two data treatments which are described respectively by figures 2a and 2b.

We get $\lambda_{1,1} = \sum_{l=1,n} x_{1,l}^2$ at the instant nT.

We get $\lambda_{2,2} = \sum_{l=1,n} x_{2,l}^2$ at the instant $(n+2)T$.

...

We get $\lambda_{s,s} = \sum_{l=1,n} x_{s,l}^2$ at the instant $(n+2(s-1))T$.

So we need $(n+2s-2)$ time steps to compute XX^T.

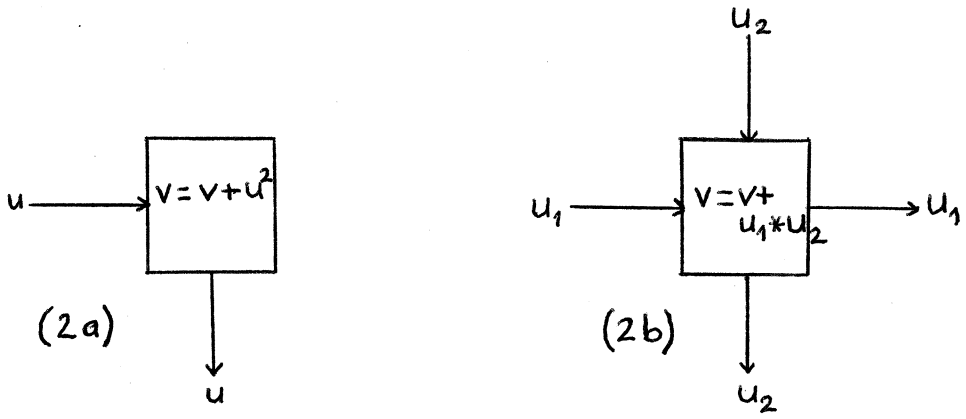

Figure 2: (2a) program of a diagonal cell.
(2b) program of a non diagonal cell.

2.2. Cholesky factorization of Λ,$\Lambda = R^T R$:

We can link the matrix product $\Lambda = XX^T$ together with the Cholesky factorization of Λ.As soon as the first $\lambda_{i,j}$ are computed the computation of R upper triangular such that $\Lambda = R^T R$ can begin.

Set down $R = (r_{i,j})_{1 \le i,j \le s}$.We have:

$r_{i,i} = [\lambda_{i,i} - \sum_{l=1,i} r_{l,i}^2]^{1/2}$ for $i = 1, \ldots,$.

$r_{i,j} = [\lambda_{i,j} - \sum_{l=1,i-1} r_{l,i} r_{l,j}]^{1/2} / r_{i,i}$ for $j = 1, \ldots, s$ and $i < j$.

There are Cholesky algorithms different from this one presented here.Those are adaptations of the LU factorization of a square matrix (chapter 4 of [7],chapter 2 of [4],[6]) to the symmetric case [1].The Brent-Luk model [1] is an hexagonal array of $s(s+1)/2$ processors and needs $4s$ times steps.The algorithm of Schreiber [12],[13] can be implemented on a triangular orthogonal array composed of $s(s+1)/2$ processors and needs only $3s$ time steps.The arithmetic operations are roughly the same.The winning of s time steps comes from the fact that in our case the matrix

R stays in the array for a susequent utilisation (computation of R^{-1},or matrix product of R by another matrix). In the Brent and Luk model [1] the computation of R is achieved only when the matrix R is out of the array which takes s additional times steps.

In our array the non diagonal processors ji compute $r_{i,j}$ for $j > i$,and the diagonal processors compute $r_{i,i}$.We can see on the figure 3 how we obtain the $r_{i,j}$ from the $\lambda_{i,j} = \lambda_{j,i}$ at each time step.As for the matrix product XX^T there are two types of cells,the diagonal cells and the others which perform different tasks showed in the figures 4a,4b,5a and 5b.These tasks can be decomposed in two steps:the first one is composed of $(j-1)$ time steps where j stands for the column index of the array and,the second one,of one time step. During the first step,every processor receiving input data performs a multiplication on them,substracts this result to its memory content and broadcasts the data on its outputs.During the second step,the diagonal processors compute the square root of their memory content and send it downwards and,the non diagonal processors,divide their memory content by the last data they receive and send this result to the right.

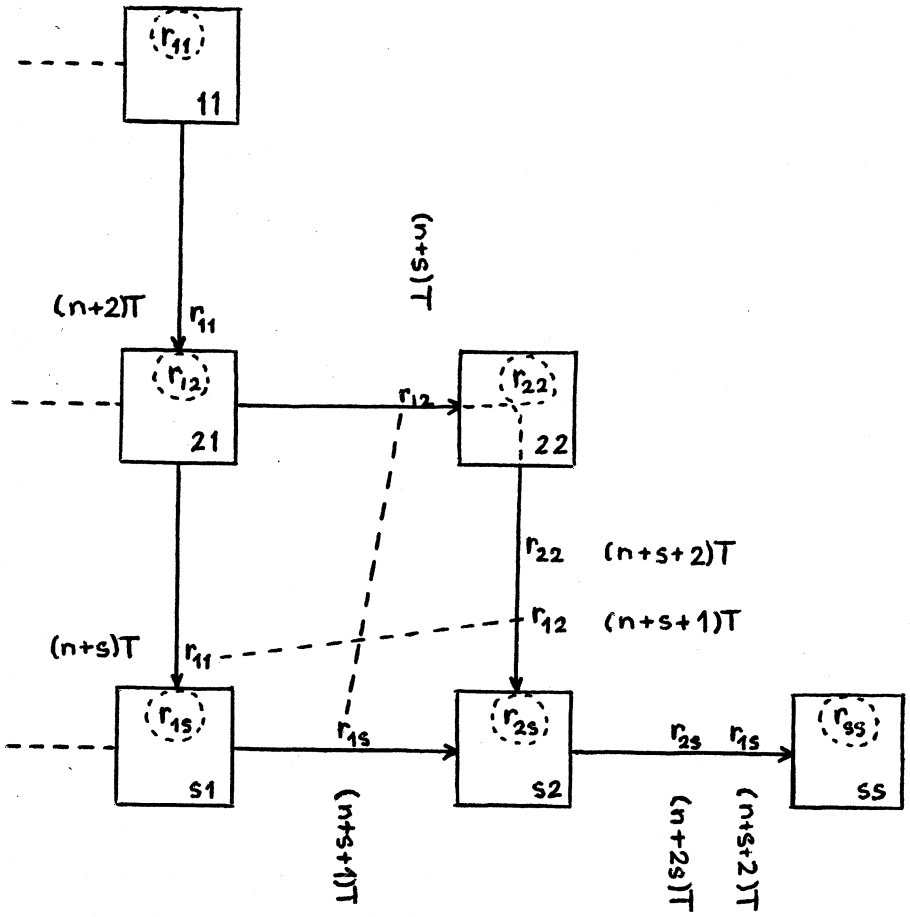

Figure 3: Cholesky factorization of Λ:computation of the upper triangular matrix R such that $\Lambda = R^T R$.

We get $r_{1,1} = [\lambda_{1,1}]^{1/2}$ at the instant $(n+1)T$.
We get $r_{2,2} = [\lambda_{2,2} - r_{1,2}^2]^{1/2}$ at the instant $(n+s+1)T = (n+4)T$.

...

We get $r_{s,s} = [\lambda_{s,s} - \sum_{l=1,s-1} r_{l,s}^2]^{1/2}$ at the instant $(n+2s-1)T = (n+1+3(s-1))T$.
So we need $(3s-2)$ time steps to compute R from Λ.

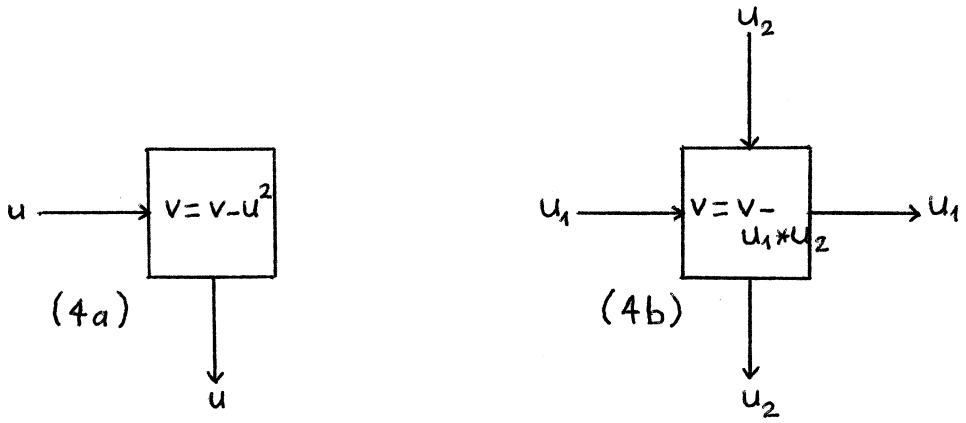

Figure 4: (4a) first step of the program of a diagonal cell.
(4b) first step of the program of a non diagonal cell.

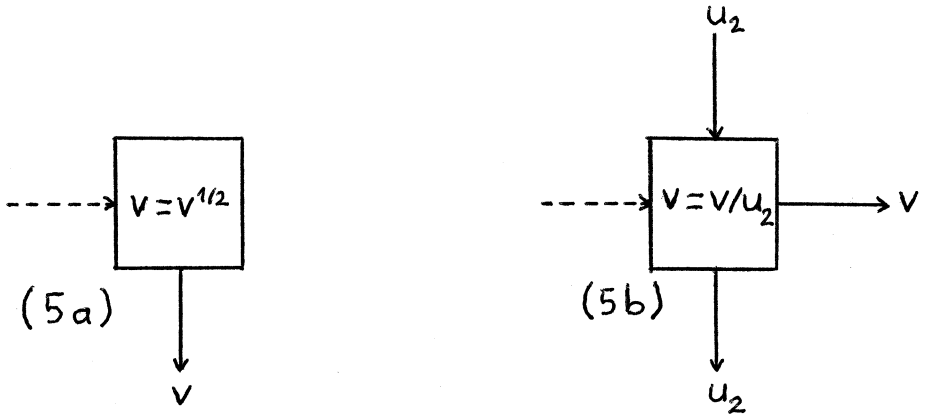

Figure 5: (5a) second step of the program of a diagonal cell.
(5b) second step of the program of a non diagonal cell.

3. Other matrix computations:

We defined the triangular structure of the array from the two functions to perform in cascade:computation of $\Lambda = XX^T$ and then the Cholesky factorization of Λ.It remains to show that the triangular array allows us to execute efficiently the other functions:matrix inversion and matrix products.

3.1. triangular matrix inversion:

Sometimes,in Factorial Data Analysis,we have A^{-1} and B^{-1} which are matrices of empirical variance of the form XX^T,instead of A and B, and for the computation of $E = R_B X R_A^T$,we need the Cholesky factorizations of A and B.To get these we proceed in two steps:first,we define the Cholesky factorizations of A^{-1} and B^{-1}:$A^{-1} = R_{A^{-1}}^T R_{A^{-1}}$ and $B^{-1} = R_{B^{-1}}^T R_{B^{-1}}$ and second,we inverse the upper triangular matrices $R_{A^{-1}}$ and $R_{B^{-1}}$ .Then we have: $E = (R_{B^{-1}})^{-1} X (R_{A^{-1}})^{-T}$.

The step of matrix inversion is always preceded by a Cholesky factorization. That is why we can always compute R^{-1} from R,R being already stored in the array.We present here a method for our triangular array of $s(s+1)/2$ processors which can be executed in $(2s-1)$ time steps from R stored in the array.

The Li and Wah algorithm [8] can be done on a triangular array of $s(s+1)/2$ processors and in $(2s-1)$ time steps but the interconnections are this time hexagonal (every processor has 6 neighbors).Furthermore the matrix R and the matrix W of the intermediate results which will be R^{-1} at the exit of the array,are both introduced in the array.

For our method and that of Li and Wah [8] the arithmetic operations are roughly the same,only the diagonal processors tasks and the data flow movements differ.These algorithms are optimal in that sense that they minimize the number of processors and time steps.(see [8])

Set down $R = (r_{i,j})_{1 \le i \le j \le s}$ and $R^{-1} = (r'_{i,j})_{1 \le i \le j \le s}$.We have:

$r'_{i,i} = r_{i,i}^{-1}$ for $i = 1, \ldots, s$ and $r'_{i,j} = -[\sum_{l=j,i+1} r_{i,l} r'_{l,j}] \cdot r'_{i,i}$ for $j = 1, \ldots, s$ and $i < j$.

The figure 6 presents the movements of the $r'_{i,j}$ and the time steps they are obtained from the $r_{i,j}$ in the array for $s = 4$.The diagonal processors ii having their memory content equal to $r_{i,i}$,inverse it and send this value downwards and to the left at the right time steps.(see figure 7a).The non diagonal processors ji compute the $r'_{i,j}$ from the $r_{i,j}$ contained in their memory.They execute tasks which can be splitted up into three steps,the first and the third,of one time step and the second, of $(j-i-1)$ time steps corresponding to $(j-i-1)$ successive couple of data inputs where i and j stand respectively for the row and column indices of the array.These three steps are described by the figures 7b,8a and 8b. During the first step every processor receiving an input data sends first its memory content downwards,then multiplies the data with it,changes the sign and sends the data on its output.During the second step the processors perform a multiplication on their input data,substract this result to their memory content and send the data on their outputs.Finally,during the third step,the processors multiply the last input data they receive with their memory content,send the data on their output and then this result to the left.

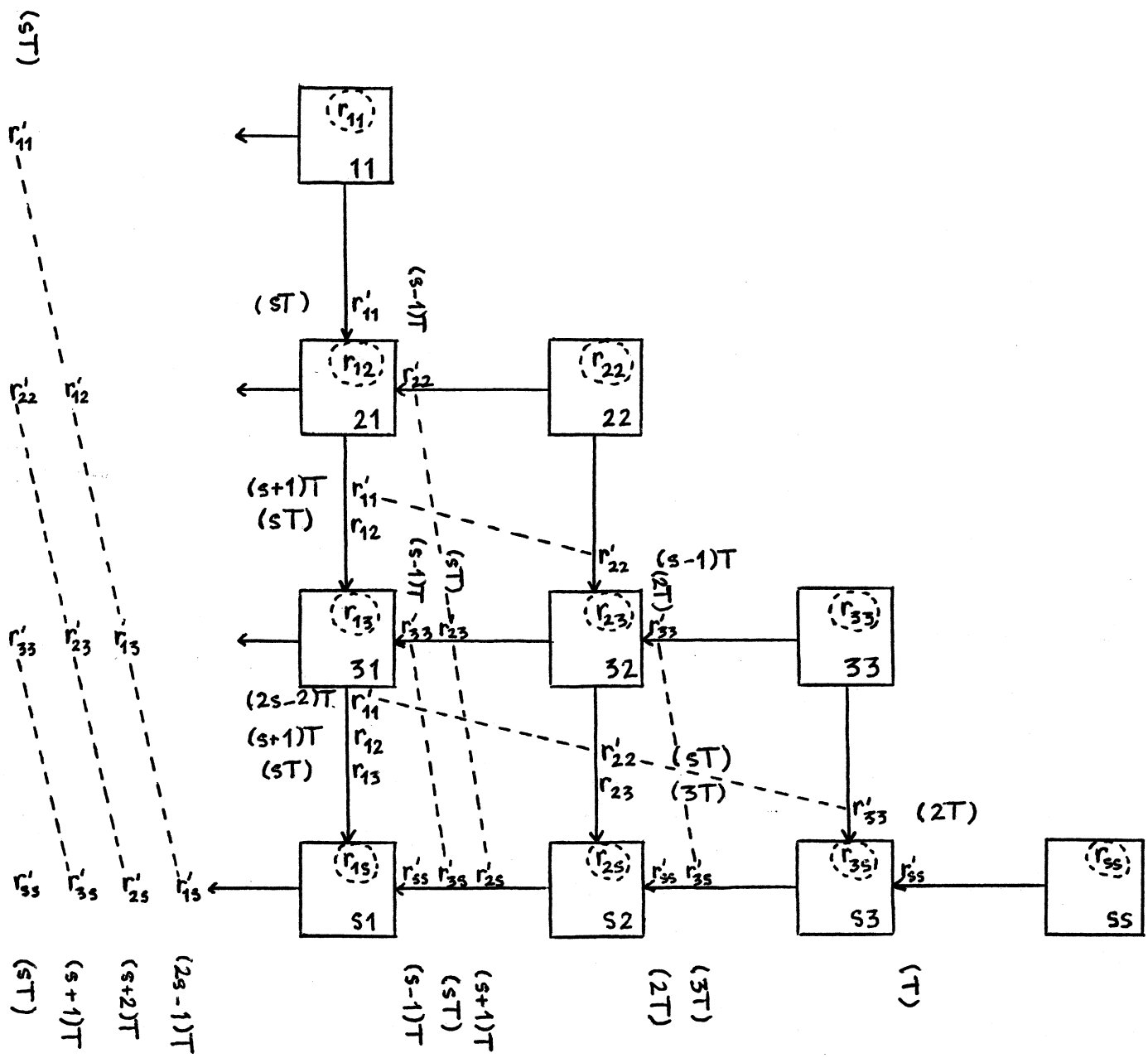

Figure 6: inversion of the upper triangular matrix R.

We get $r'_{s,s} = 1/r_{s,s}$ at the instant T.

We get $r'_{s-1,s-1} = 1/r_{s-1,s-1}$ at the instant $2T$.

We get $r'_{s-1,s} = -r_{s-1,s} \cdot r'_{s,s} \cdot r'_{s-1,s-1}$ at the instant $3T = (1 + 2 \times 1)T$.

...

We get $r'_{1,s} = -[\sum_{l=s,2} r_{1,l} r'_{l,s}] \cdot r'_{1,1}$ at the instant $(1 + 2(s-1))T$.

So we need $(2s-1)$ time steps to compute R^{-1} from R.

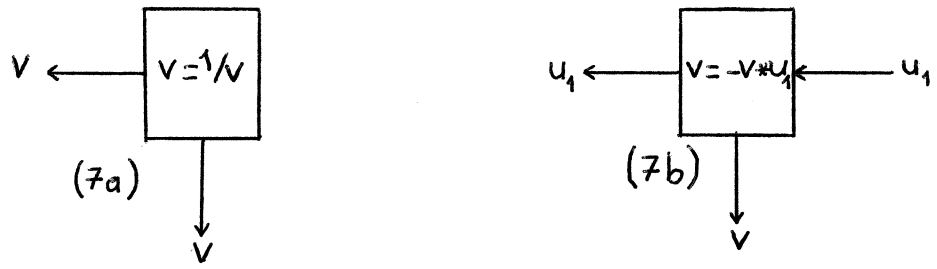

Figure 7: (7a) program of a diagonal cell.
(7b) first step of the program of a non diagonal cell.

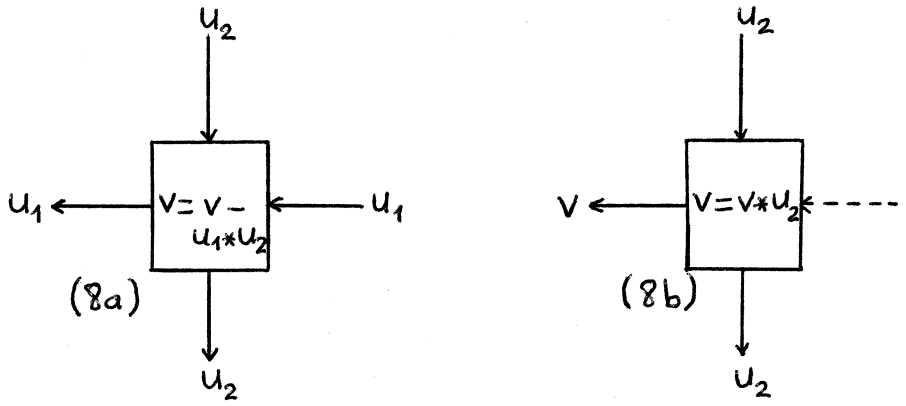

Figure 8: (8a) second step of the program of a non diagonal cell. (8b) third step of the program of a non diagonal cell.

3.2. matrix products:

There are a lot of systolic algorithms for matrix products.They depend on the chosen structure of the array,(hexagonal or orthogonal connections), and on the type of cells,(with local memory or not). Suppose we want to compute the product $C = AB$ with A and B square matrices of size s;the rows of A and the columns of B will have to cross in the array to get the matrix C.

When the cells have no local memory the intermediate results $\sum_{l=1,p} a_{i,l}b_{l,j}, 1 \leq p \leq s$ have to go through the array at the same time as the matrices A and B ([9],[6],[8]).

In the model of Melkemi and Tchuente [9] the matrix A (respectively B) is introduced column by column (respectively row by row) and moves horizontally along the rows of the array from the left to the right (respectively from the right to the left). C moves vertically from the top to the bottom.The connections between cells are orthogonal and the array can be square (composed of s^2 cells) or rectangular (composed of $s \times m$ cells with $m \geq s$).The product is executed after $(3s-2)$ time steps.If the array is of size $s(2s-1)$ we only have to introduce the matrix A (respectively B) column by column (respectively row by row).But if the array is of size $s \times m$ with $s \leq m < 2s-1$,we have to repeat some of the elements of the columns of A (respectively of the rows of B) because some of the cells have to perform the tasks of several cells. By using this principle we can execute a matrix product of two square matrices of size s on an array of s^2 cells in $(3s-2)$ time steps. This algorithm is optimal (it minimizes the number of time steps and processors).Nevertheless it has two drawbacks:

1. it is necessary to apply a non trivial treatment to the matrix A and B (dupplicate some columns or rows) before introducing them into the array.
2. when we want to compute several matrix products in cascade,as it is our case when we have to deal with block partioning,the duration of introducing the matrices A and B into the array is practically doubled which delays for about s time steps for the next product.

The Kung and Leiserson model [6],[7] is used for band matrix products. If A and B are of band widths w_1 and w_2 the product $C = AB$ is performed on an array with hexagonal connections,a parallelogram shape and composed of $w_1 \times w_2$ cells.A and$B$ are introduced diagonal by diagonal along the diagonals of the array from the top and the result C moves vertically from the bottom to the top.The product is obtained after $3s + min(w_1, w_2)$ time steps.However this array cannot be used for dense matrices because the number of cells of the array becomes too high ($(2s-1)^2$ cells would be necessary).

The Li and Wah algorithm [8] deals with dense matrices on an array with hexagonal connections.It is of the same type of array than that of Kung and Leiserson but it allows,thanks to a different data introduction,to reduce the number of time steps and cells.The array has this time an hexagonal shape and is composed of $3s(s-1)$ cells.The matrices B and C move along the diagonals of the array,B from the top,C from the bottom, and A moves vertically from the bottom to the top.The matrices A and B are introduced counterdiagonals by counterdiagonals,the $(2s-1)$ counterdiagonals along the diagonals of the array and we get C diagonal by diagonal.The matrix product is computed after $2s$ time steps.The algorithm is simple to do and very fast but the cost of processors is still high.

When the cells have a local memory the arrays are the most often square, with orthogonal connections and composed of s^2 cells.The matrices A and B move along perpendicular directions and the intermediate results $\sum_{l=1,p} a_{i,l}b_{l,j}, 1 \leq p \leq s$ are stored in the cell of the ith row and jth column which computes $c_{i,j}$ ([5]).In that case the matrix product takes $(3s-2)$ time steps and $(s-1)$ additional time steps are necessary to get the matrix C out of the array.

Melkemi and Tchuente [9] use again the same technique of dupplication of some elements of A and B in order to activate faster the cells of the array and thus save $(s-1)$ time steps.This algorithm is optimal if we introduce A and B along two sides of the array but it has the same drawbacks than those quoted earlier.

3.2.1. matrix product RX with:

- R upper triangular of size s
- X rectangular of size $s \times n$

We cannot use here the techniques of matrix products described above because our array is triangular instead of being square or rectangular,but the basic operations are roughly the same.Instead of moving R and X simultaneously in the array along different directions we introduce the matrix R a little time before the matrix X in order to store it in the array.Every $r_{i,j}$ is then memorized in the cell ji.When X is introduced,its elements are multiplied by the memory contents of the cells forming the product $C = RX$ which moves from the top to the bottom.

Set down $R = (r_{i,j})_{1 \leq i \leq j \leq s}$,$X = (x_{i,j})_{\substack{1 \leq i \leq s \\ 1 \leq j \leq n}}$ and $C = (c_{i,j})_{\substack{1 \leq i \leq s \\ 1 \leq j \leq n}}$.

The figure 9 presents the movings of the matrices R and X and the dates, the $c_{i,j}$ are computed in the array for $s = 3$ and $n = 4$.There are once more two types of cells which tasks are described by the figures 10a and 10b. Every diagonal cell ii memorizes $r_{i,i}$,then multiplies its memory content by the input data it receives and sends this result downwards.Every non diagonal cell ij memorizes $r_{j,i}$,multiplies its memory content by the input data coming from the left and broadcasts it to the right,adds this product with the input data coming from the top and sends this result downwards. Our matrix product is executed in $(2s + n - 1)$ time steps which is of the same order of time as the product algorithms described above but with practically twice less cells.This product takes place in the computation of $E = R_B X R_A^T$ where R_A and R_B are upper triangular of sizes n and k respectively and X is $k \times n$ (see introduction).When $k > s$ we apply block partitioning to the matrices R and X (see [10]).

The product $C = RX$ is followed by the product $E = C(R')^T$ which can be performed in the same way but,instead of introducing C row by row as X, we introduce C column by column.This second step begins at the instant $(n+1)T$ just after the entry of $x_{1,n}$ in the array and is executed in $(2n+s-1)$ time steps.As soon as we get the first elements $e_{i,j}$ of E, the computation of $W = EE^T$ (see introduction) can begin.This last step begins at the instant $(2n+2)T$ and as it can be performed in $(n + 2s - 2)$ time steps (see 2.1.) we get W in the array after $(3n + 2s)$ time steps and,out the array, after $(4n + 2s - 1)$ time steps.

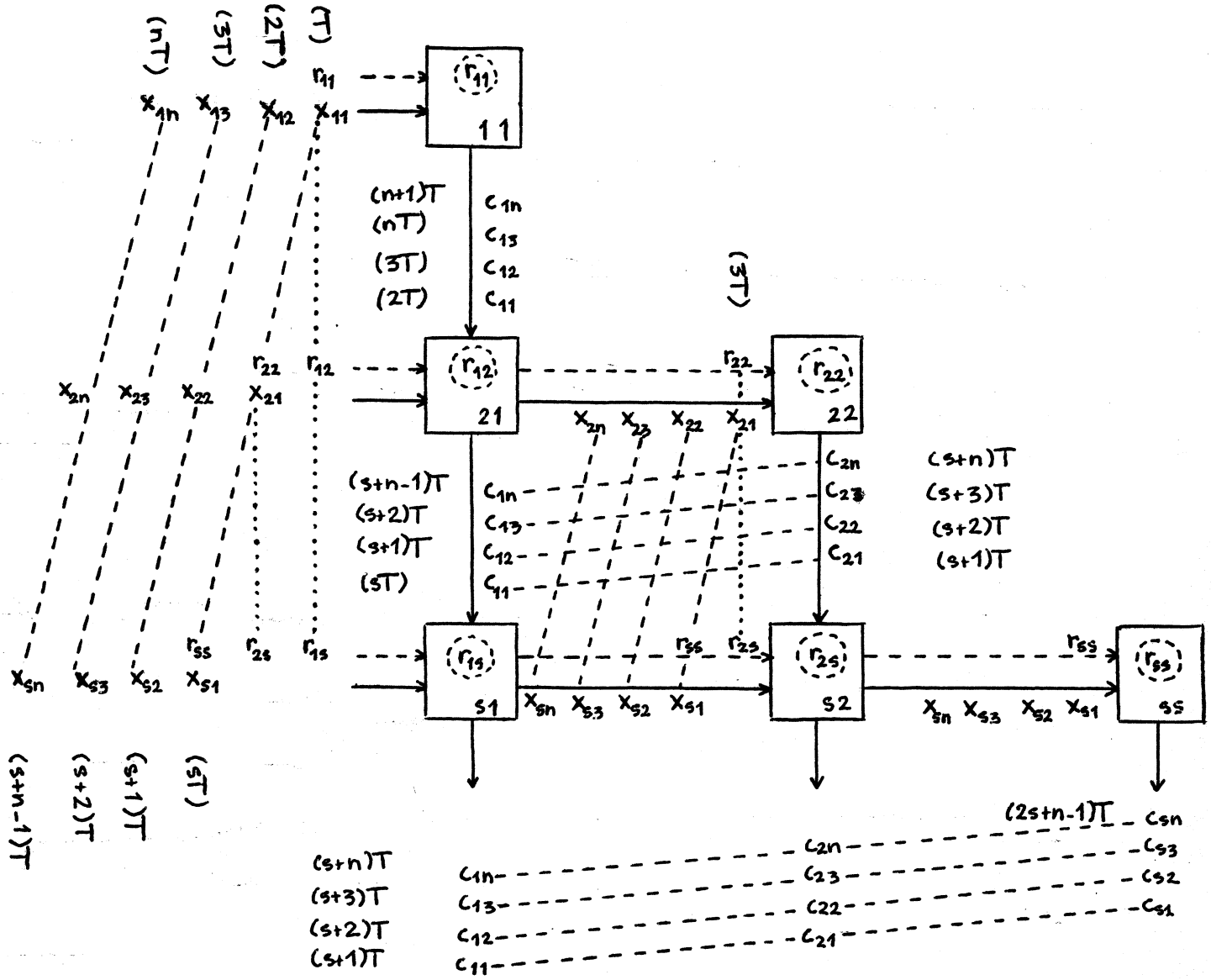

Figure 9: moving of the matrices R,X, and C in the array for the product $C = RX$.

We get $c_{1,1} = \sum_{l=1,s} r_{1,l} \cdot x_{l,1}$ at the instant $(s+1)T$.

We get $c_{1,2} = \sum_{l=1,s} r_{1,l} \cdot x_{l,2}$ at the instant $(s+2)T$.

...

We get $c_{1,n} = \sum_{l=1,s} r_{1,l} \cdot x_{l,n}$ at the instant $(s+n)T$.

...

We get $c_{s,n} = r_{s,s} \cdot x_{s,n}$ at the instant $(2s+n-1)T$.

So we get $C = RX$ after $(2s+n-1)$ time steps.

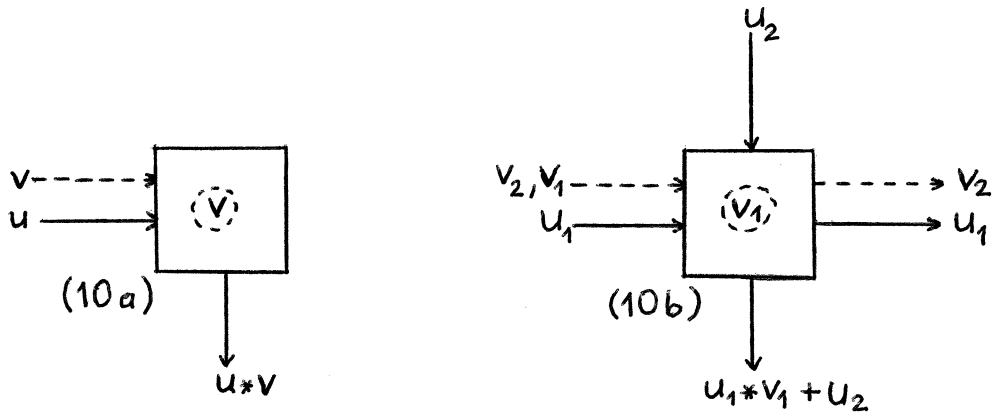

Figure 10: (10a) program of a diagonal cell.The cell memorizes the value v it receives.
(10b) program of a non diagonal cell.The cell memorizes the first value v it receives and then broadcasts the following values to the right.

3.2.2. matrix product $C = AB$ with:

- A and B square matrices of size s.

Our array being composed of processors with local memories,we can use the techniques presented in [5] which consist of sending A horizontally by rows, B vertically by columns and of storing the intermediate results of the product $\sum_{l=1,s} a_{i,l}b_{l,j}$ in the processor ij.(see figure 11) But in that way we only get the elements of the lower triangular part of C. In order to obtain the upper triangular part of C we introduce the matrix B (respectively A) by columns (respectively rows) again into the array but that time horizontally (respectively vertically).The matrices A and B are introduced again into the array in the following way:

Every $a_{j,l}, l = 1, \ldots, s$ of the jth row of A is memorized in the jth diagonal processor during the computation of $c_{j,j} = \sum_{l=1,s} a_{j,l}b_{l,j}$ and is sent again to the bottom of the array, one time step after $b_{s,j}$;the $b_{l,i}, l = 1, \ldots, s$ of the ith column of B enter the array one time step after $a_{i,s}$.(see figure 12)

Every diagonal processor ii computes $c_{i,i}$ and every non diagonal processor ij,$c_{i,j}$ and then $c_{j,i}$ with $i > j$,without waiting delay between these two operations.The movings of the matrices A and B in the array and the instants when the $c_{i,j}$ are computed are detailed in the figures 11 and 12 and the tasks of the diagonal and non diagonal cells,in the figures 13a,13b,14a and 14b.This product is executed in $(4s - 3)$ time steps which represents a saving in comparison with the standard matrix product which is executed in $(3s - 2)$ time steps on an array of s^2 processors.

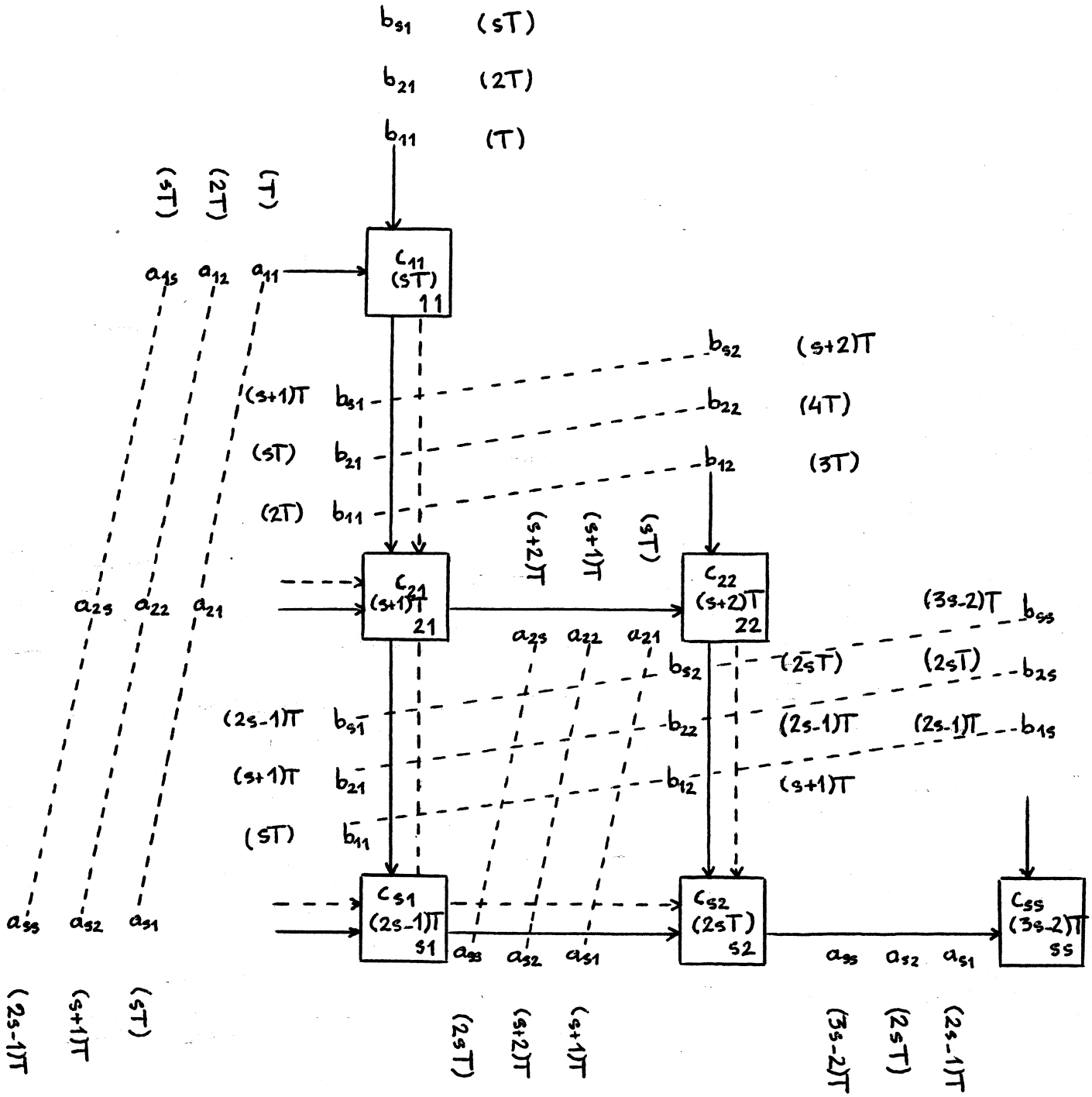

Figure 11: Moving of the matrices A and B for the matrix product $C = AB$.

First phase:computation of the lower triangular part of C.

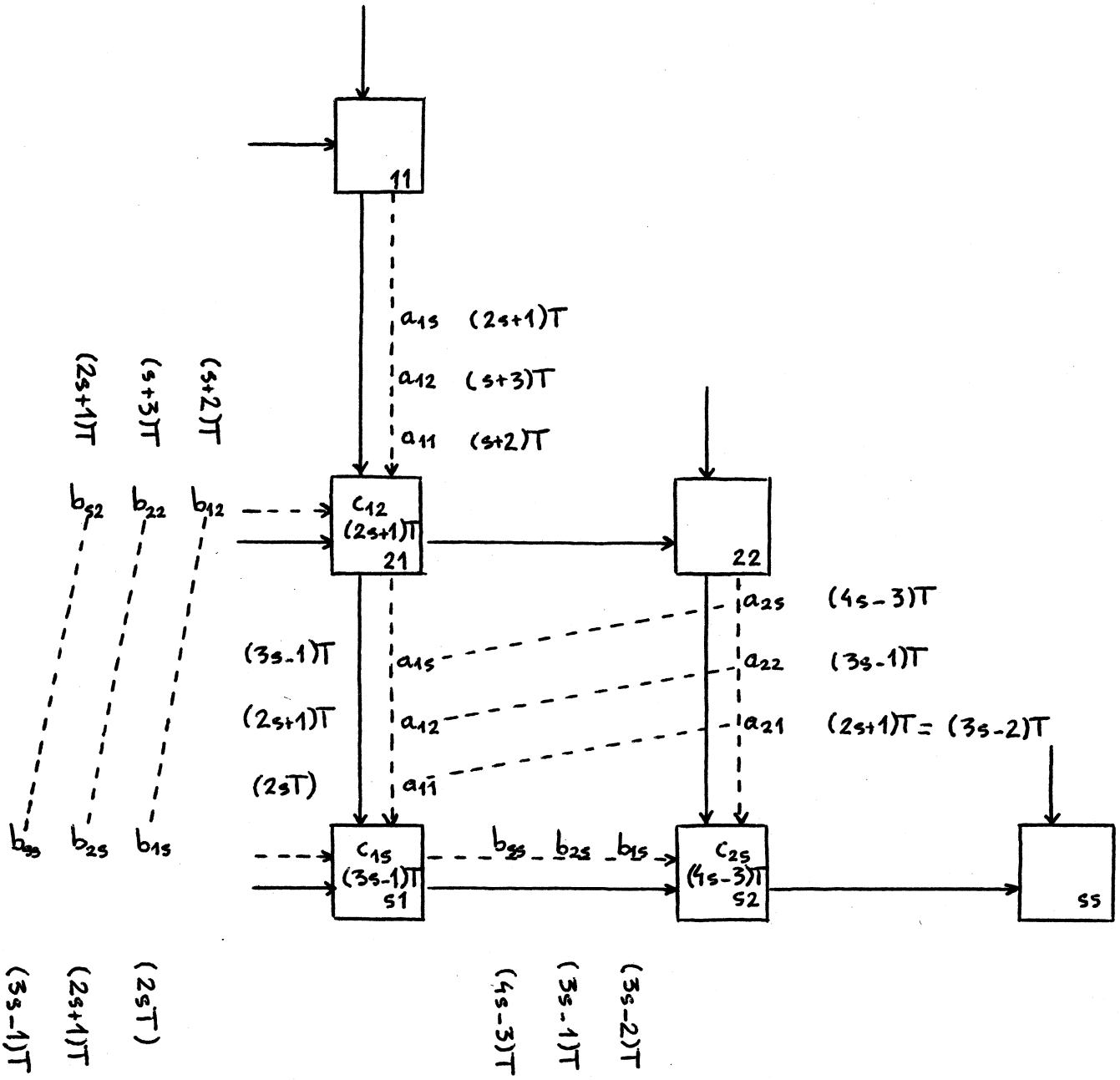

Figure 12: Moving of the matrices A and B for the matrix product $C = AB$.

Second phase:computation of the upper triangular part of C.

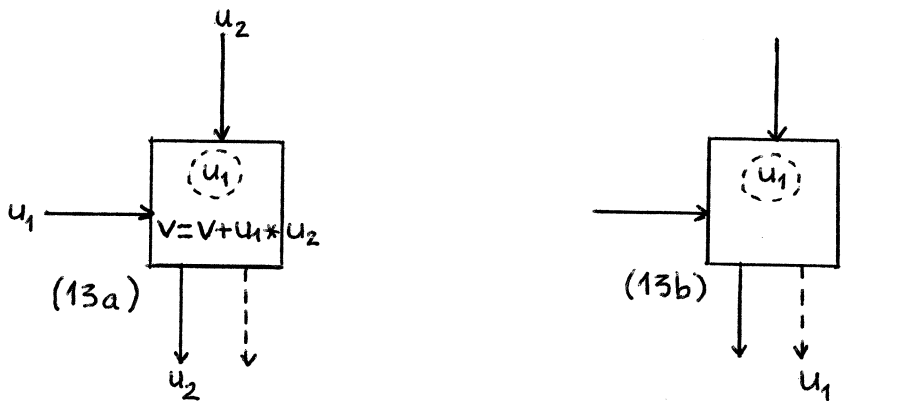

Figure 13: (13a) first phase of the program of a diagonal cell:the cell memorizes every u_1 it receives in an internal FIFO(First In First Out).
(13b) second phase of the program of a diagonal cell:the cell broadcasts every u_1 memorized in its internal FIFO.

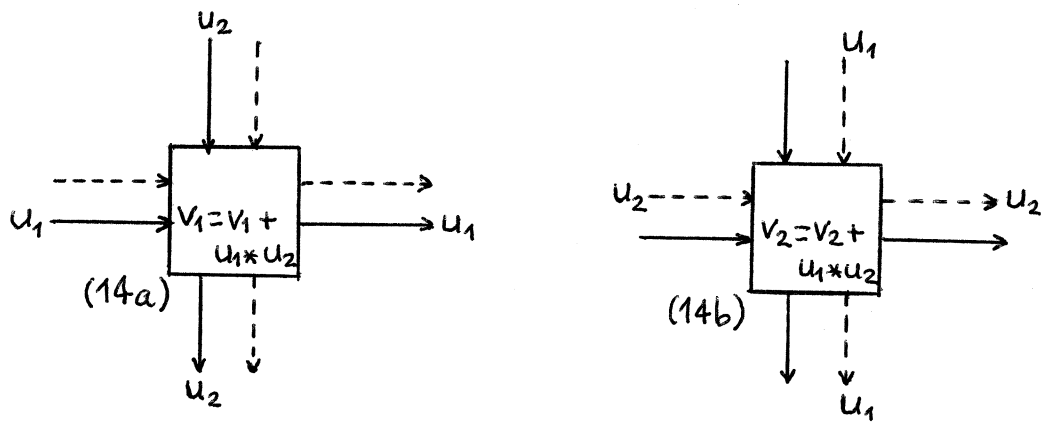

Figure 14: (14a) first phase of the program of a non diagonal cell.
(14b) second phase of the program of a non diagonal cell.

The reader will find page 16 a summary table of the studied algorithms as well as comparisons with other algorithms.We put a star $*$ on those we chose.All these algorithms are linear in time.

algorithm	shape and connections of the array	number of time steps	authors and special features
matrix product XX^T with X of size $s \times n$	triangular array of $s(s+1)/2$ processors with orthogonal connections [fig 1-2]	$(n+2s-2)$ time steps	Schreiber* [12],[13]
Cholesky factorization of XX^T,square matrix of size s	triangular array of $s(s+1)/2$ processors with orthogonal connections [fig 3-5]	$(3s-2)$ time steps	Schreiber* [12],[13]:in cascade with the matrix product XX^T
	triangular array of $s(s+1)/2$ processors with hexagonal connections	$4s$ time steps	Brent-Luk [1]
inversion of R upper triangular matrix of size s	triangular array of $s(s+1)/2$ processors with hexagonal connections	$(2s-1)$ time steps	Li-Wah [8]
	triangular array of $s(s+1)/2$ processors with orthogonal connections [fig 6-8]	$(2s-1)$ time steps	Porta* [10]:is computed from R stored in the array
matrix product RX with R upper triangular of size s and X rectangular of size $s \times n$	triangular array of $s(s+1)/2$ processors with orthogonal connections [fig 9-10]	$(2s+n-1)$ time steps	Porta* [10]:R is stored in the array before X is introduced
matrix product of two square dense matrices of size s	square array of s^2 processors with orthogonal connections	$(3s-2)$ time steps	Melkemi-Tchuente [9]
	hexagonal array of $3s(s-1)$ processors with hexagonal connections	$2s$ time steps	Li-Wah [8]
	triangular array of $s(s+1)/2$ processors with orthogonal connections [fig 11-14]	$(4s-3)$ time steps	Porta* [10]:computation of the upper triangular part and then of the lower triangular part.

Table 1: Matrix computations

References

[1]R.P.Brent,F.T.Luk (1982) *Computing the Cholesky factorization using a systolic architecture.*The Australian University,Dpt of Computer Science,Technical Report T-R CS 82-08.

[2]F.Chatelin,D.Belaid (1986) *Numerical Analysis for Factorial Data Analysis.Part I.*Etude IBM F-107.Centre Scientifique de Paris.

[3]F.Chatelin,T.Porta (1987) *Numerical Analysis for Factorial Analysis.Part II.*Etude IBM F-110.Centre Scientifique de Paris.

[4]J.M.Delosme (1982) *Algorithms for finite shift-rank processes.* Ph.D.Technical Report M 735-22.Stanford University.

[5]Guibas et al (1979) *Direct VLSI implementation of combinatorial algorithms.*Proc.Calt.Conf. on VLSI pp 509-521.

[6]H.T.Kung,C.E.Leiserson (1980) *Highly Concurrent systems* in *Introduction to VLSI Systems* C.A. Mead-L.A.Conway Eds. pp 271-289.

[7]C.E.Leiserson (1981) *Area Efficient VLSI Computation.*Ph.D. Carnegie Mellon University.

[8]G.J.Li,B.W.Wah (1985) *The Design of Optimal Systolic Arrays.* IEEE Transactions on Computers Vol C-34 No 1 pp 66-77.

[9]L.Melkemi,M.Tchuente (1986) *Complexity of Matrix Products on a Class of Orthogonally Connected Systolic Arrays.*to appear in IEEE Transactions on Computers.TIM3-IMAG Grenoble.

[10]T.Porta,F.Chatelin (1986) *Un reseau systolique programmable pour l'Analyse Factorielle des Donnees.*Etude IBM F-103.Centre Scientifique de Paris.

[11]G.Saucier,G.Chevalier (1986) *A 2-D Processor Array for Wafer Scale Integration.*Rapport Interne LCS-IMAG Grenoble.

[12]R.Schreiber (1985) *Private Communication.*Centre Scientifique IBM, Paris.

[13]R.Schreiber (1987) *Cholesky Factorization by Systolic Array.*DCS RPI Troy,NY 12180-3590 T-R No 87-14.