Linear Algebra in One-Dimensional Systolic Arrays

Gregor Papa and Jurij Šilc
Computer Systems Department, "Jožef Stefan" Institute, Jamova 39, 1001 Ljubljana, Slovenia
email: gregor.papa@ijs.si, jurij.silc@ijs.si

Keywords: systolic array, QR, LU, decomposition, Gauss elimination, matrix multiplication

Edited by: Rudi Murn
Received: June 1, 1999 **Revised:** July 20, 1999 **Accepted:** April 5, 2000

Frequently used problems of linear algebra, such as the solution of linear systems, triangular decomposition and matrix multiplication, are computationally extensive. To increase the speed, those problems should be solved with systolic structures, where many processors are used concurrently to compute the result. Since two-dimensional array of processors is very consumptive, considering space and resources, it is better to use one-dimensional array of processors. This leads to the operation reallocation and causes unequal utilization of processors, but it is much easier to implement since there is only one linear array of processors.

1 Introduction

Many scientific problems can be solved by linear algebraic computations, but even some basic operations are computationally extensive. Computation time could be shortened by synchronous data processing, which is enabled through the systolic structure. Systolic solving is presented by the processor structure, where data is flowing through the net of specialized processors, which are locally connected and work synchronically. This approach has some disadvantages, while there is a lot of connections. It is difficult to monitor all processors and to read data from them. Besides, they are poorly utilized, since they mostly wait for their data to compute. It is possible to compose the structure with higher utilization, time suitability and lower complexity [3], which would remove the mentioned disadvantages. To realize that, we can merge some processors, i.e. one processor performs tasks of more processors, and we can put them into one straight array, to reduce the number of connections and to make easier access to the processors. This work presents the linearization of different matrix transformation algorithms, such as elimination, decompositions and multiplication, and also some comparisons of two-dimensional and linear arrays are given.

2 Linear system of equations

Systolic arrays can be used to solve the system of linear equations [2] in the form:

$$\mathbf{A} \cdot x = \mathbf{b}.$$

Suitable triangular systolic array for realization of Gauss elimination and various decompositions (QR and LU) [4, 9] is presented in Fig. 1. Shapes ○ and □ represent two types of processor (diagonal and inner), performing their own instructions; diagonal operations are executed in diagonal processors and inner operations are executed inside the structure. Inputs of the structure are matrix coefficients (a_{ij}) and at the end there are coefficients of the upper-triangular matrix inside the structure and the coefficients of the lower-triangular matrix on the outputs. Dotted square represents a delay τ. According to the matrix size $n \times n$ the number of required processors n^* is:

$$n^* = \frac{n(n+1)}{2}.$$

Where n diagonal processors and $(n^* - n)$ inner processors are required.

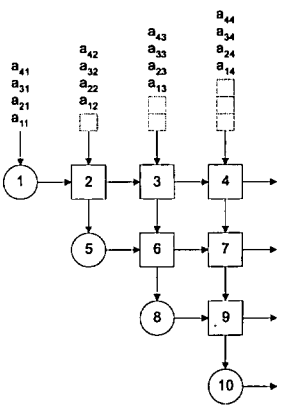
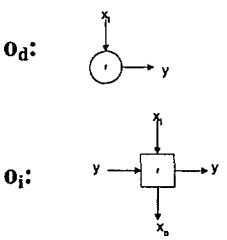


Figure 1: Triangular systolic array (n=4)

Two-dimensional array in Fig. 1 can be transformed into one-dimensional [11, 6] in several directions; horizontal linear array (Fig. 2), vertical linear array (Fig. 4), diagonal linear array (Fig. 6) and interweaved linear array (Fig. 8). Symbol ⊡ represents the processor that performs the

tasks of processors ○ and □. Next, the operations of diagonal and inner processors are presented. All mentioned operations [1] are executed in one systolic cycle (step), but of course, more cycles are needed to finish a transformation, i.e. those operation are repeated (operations present only the set of processor's instructions).

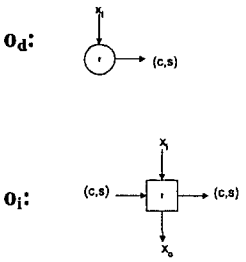
Gauss elimination [5] and LU decomposition [7]:



$$1.\ y=\frac{x_i}{r}$$
$$2.\ r=x_i$$
$$1.\ x_o=x_i-y\cdot r$$
$$2.\ r=x_i$$

In such structure there is a similarity of Gauss elimination and LU decomposition (results of LU decomposition are just transformed Gauss coefficients) [7].

QR decomposition [5]:



$$\text{if } x_i=0,\ r=0 \text{ then } c=1,\ s=0$$
$$\text{else } t=\sqrt{r^2+x_i^2},\quad c=r/t,\quad s=x_i/t,\ r=t$$
$$x_o=-s\cdot r+c\cdot x_i$$
$$r=c\cdot r+s\cdot x_i$$

Input or output (c, s) of QR decomposition will be treated as y in the following sections.

Because of the transformations the instruction sets of the processors are changed as described in the following sections.

2.1 Horizontal array

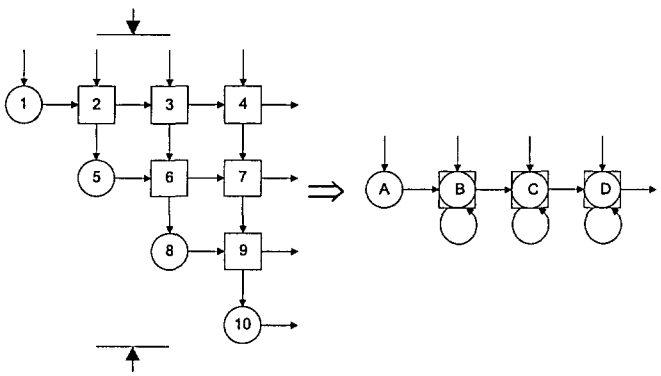


Figure 2: Transformation into horizontal array

As presented in Fig. 2, processor 1 is mapped into processor A; processors 2 and 5 into processor B, processors 3, 6 and 8 into processor C; processors 4, 7, 9 and 10 into processor D. So, processor A takes over the tasks of one processor and performs operation o_d, but processor D takes over the tasks of four processors and performs operations o_d and o_i. They work in different modes:

- mode 1: operation o_d with one input x_i,
- mode 2: operation o_i with two inputs (x_i,y),
- mode 3: operation o_i with one input y and one input from its output (x_o to x_i).

Each processor works in these modes:

- processor A always in mode 1,
- processor B in modes 2 and 1,
- processor C in modes 2, 3 and 1,
- processor D in modes 2, 3, 3 and 1,
- additional processors would work in modes 2, 3, ... 3 and 1.

Occupation of processors is presented in Table 1.

Table 1: Processor occupation in horizontal array

	A	B	C	D
1	1			
2		2		
3		1	2	
4			3	2
5	1		1	3
6		2		3
7		1	2	1
8			3	2
...				
17			1	3
18				3
19				1

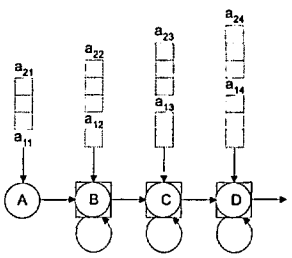


Figure 3: Data inputs in horizontal array

Input values a_{11}, a_{12}, a_{13} and a_{14} are delayed for one τ, and values a_{21}, a_{22}, a_{23} and a_{24} are delayed for $(n-1)\tau$ according to values a_{11}, a_{12}, a_{13} and a_{14}, where n is the number of processors, as presented in Fig. 3.

2.2 Vertical array

As it can be seen in Fig. 4, processors 1, 2, 3 and 4 are mapped into processor A; processors 5, 6 and 7 into processor B; processors 8 and 9 into processor C; processor 10 into processor D. Processor A is the most loaded, while processor D takes over the tasks of only one processor.

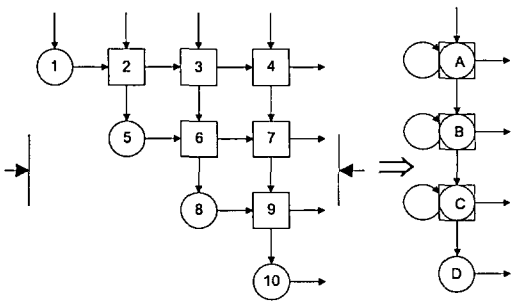


Figure 4: Transformation into vertical array

Processors A, B and C perform operations o_d and o_i, while processor D performs only operations o_d. They work in different modes:

- mode 1: operation o_d with one input x_i,
- mode 2: operation o_i with one input x_i and one input from its output (y to y).

Table 2: Processor occupation in vertical array

	A	B	C	D
1	1			
2	2			
3	2	1		
4	2	2		
5	1	2	1	
6	2		2	
7	2	1		1
8	2	2		
...				
17		2	1	
18			2	
19				1

Each processor works in these modes:

- processor A in mode 1, 2, 2 and 2,
- processor B in mode 1, 2 and 2,
- processor C in mode 1 and 2,
- processor D always in mode 1,
- additional processors would work in modes 1, 2, ...2 and 2.

Occupation of processors and their work modes are presented in Table 2. Values a_{11}, a_{12}, a_{13} and a_{14} follow each other without delay, values a_{21}, a_{22}, a_{23} and a_{24} are immediate successors of values a_{11}, a_{12}, a_{13} and a_{14}, as presented in Fig. 5.

When transformed into horizontal or vertical array, the processors' occupation and their instruction set are equal. The only difference can be noticed in data inputs.

2.3 Diagonal array

Fig. 6 presents the diagonal contraction, where processors 1, 5, 8 and 10 are mapped into processor A; processors 2, 6 and 9 into processor B; processors 3 and 7 into processor C; processor 4 into processor D. Even here the most loaded is processor A and at least processor D, but all processors

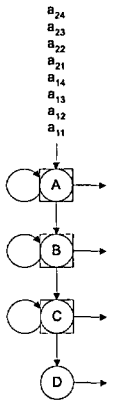


Figure 5: Data inputs in vertical array

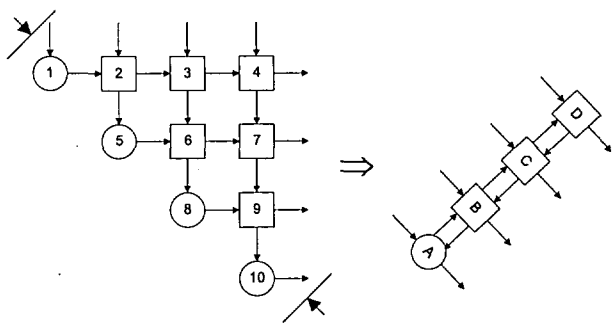


Figure 6: Transformation into diagonal array

execute only one type of operations (processor A performs only diagonal operations, the others only inner operations).

Processor occupation and their operations are presented in Table 3.

Table 3: Processor occupation in diagonal array

	A	B	C	D
1	o_d			
2	o_d	o_i		
3	o_d	o_i	o_i	
4	o_d	o_i	o_i	o_i
5	o_d	o_i	o_i	o_i
6	o_d	o_i	o_i	
7	o_d	o_i		
8	o_d			
...				
14	o_d	o_i	o_i	
15	o_d	o_i		
16	o_d			

Values a_{11}, a_{12}, a_{13} and a_{14} are one τ delayed and are followed by values a_{21}, a_{22}, a_{23} and a_{24}. Values a_{31}, a_{32}, a_{33} and a_{34}, are delayed $2(n-1)\tau$, where n is the number of processors, as presented in Fig. 7.

Contraction of the array in the direction of the other diagonal is not reasonable, while there would be too many delays and inputs/outputs on each processor.

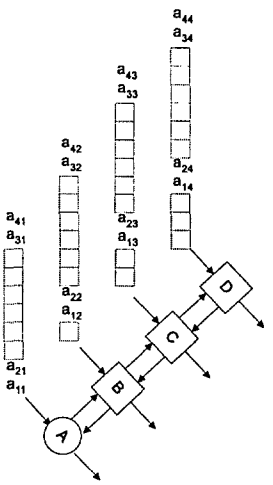


Figure 7: Data inputs in diagonal array

2.4 Processor mirroring

To decrease the number of processors and to enhance the performance of transformations, mirroring can be used. The processor can be mirrored into another processor, so that its tasks are executed while another processor would be idle otherwise. The example of processor mirroring in horizontal linear array is presented in Table 4. Processor A is mapped into processor B, and merged processor A+B executes tasks of both processors. Similarly other mirrorings can be used.

Table 4: Processor mirroring a)original array, b)array with mapped processor

a)	A	B	C	D	b)	A+B	C	D
1	1				1	1		
2		2			2	2		
3		1	2		3	1	2	
4			3	2	4		3	2
5	1		1	3	5	1	1	3
6		2		3	6	2		3
7		1	2	1	7	1	2	1
8			3	2	8		3	2
...					...			

2.5 Interweaved array

When there is an odd number of processors in the first line of the triangular array, the interweaved method can be used, as presented in Fig. 8 [11], where the isomorphic embedding of the graph is employed. Processors in Fig. 8a are mapped into processor array in Fig. 8b: processors 1, 6, 10, 13 and 15 are mapped into processor A; processors 2, 5, 7, 11 and 14 into processor B; processors 3, 4, 8, 9 and 12 into processor C. All processors (A, B, C) are evenly loaded, while each of them takes over the tasks of five processors.

The method is similar to processor mirroring, but it occupies processors almost completely and evenly. Instead

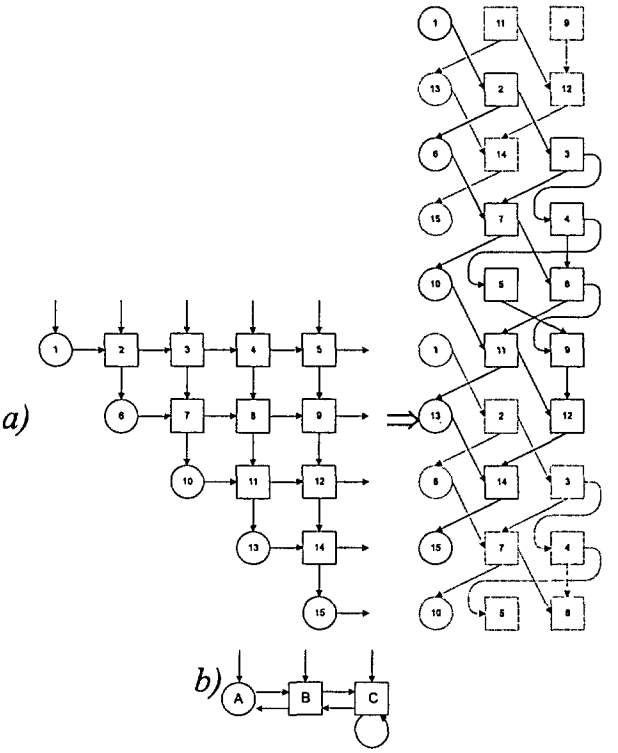


Figure 8: Transformation into interweaved array

of $n = 5$ processors only $n^* = \frac{(n+1)}{2} = 3$ are needed, which are fully utilized. Processor A performs operations o_d, while B and C perform operations o_i. Processors occupation and their operations are presented in Table 5.

Table 5: Occupation of interweaved array

	A	B	C
1	o_d		
2	o_d	o_i	
3	o_d	o_i	o_i
4	o_d	o_i	o_i
5	o_d	o_i	o_i
6	o_d	o_i	o_i
7	o_d	o_i	o_i
8	o_d	o_i	o_i
9	o_d	o_i	o_i
10	o_d	o_i	o_i
...			
27	o_d	o_i	o_i
28	o_d	o_i	
29	o_d		

Values a_{11}, a_{12}, a_{13}, a_{14} and a_{15} are delayed one τ, values a_{21}, a_{22}, a_{23}, a_{24} and a_{25} are delayed $(n^* + 1)\tau$, according to values a_{11}, a_{12}, a_{13}, a_{14} and a_{15}, as presented in Fig. 9.

3 Matrix multiplication

Systolic arrays can be also used when performing matrix multiplication [2] of the form

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} + \mathbf{C_0}.$$

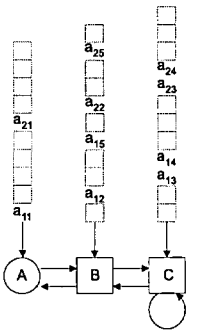


Figure 9: Data inputs in interweaved array

Square array of processors for multiplication of two square matrices is presented in Fig. 9 [8]. Inputs of the structure are coefficients (a_{ij} in b_{ij}) of the matrices and at the end of the process there are coefficients c_{ij} inside the structure. According to the matrix size $n \times n$ the number of required processors n^* is:

$$n^* = n^2.$$

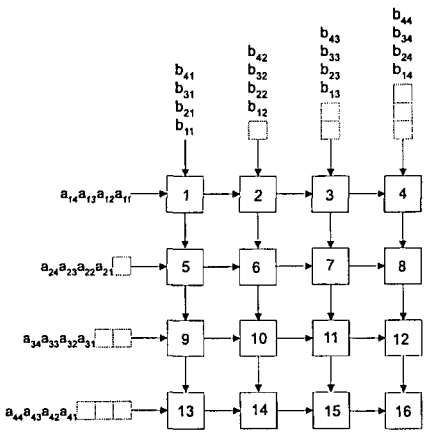
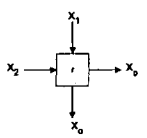


Figure 10: Square systolic array (n=4)

All processors in the square array in Fig. 10 perform the same operations [8]:



1. $x_o = x_1 \cdot x_2 + r$
2. $r = x_o$

3.1 Horizontal array

Horizontal array is obtained when all processors of the first column are merged into processor A, processors of the second column into processor B, etc, as presented in Fig. 11. Processors perform the same operations, as before the transformation, beside that, there is an additional input from one of its outputs.

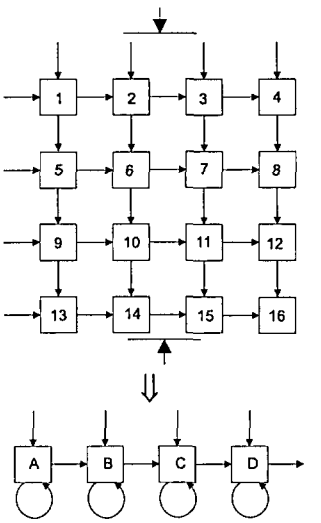


Figure 11: Transformation into horizontal array

Occupation of the processors is presented in Table 6, where numbers represent the processor of the adequate (square) array that would be used in that moment.

Table 6: Processor occupation in horizontal array

	A	B	C	D
1	1			
2	5	2		
3	9	6	3	
4	13	10	7	4
5	1	14	11	8
6	5	2	15	12
7	9	6	3	16
...				
17		14	11	8
18			15	12
19				16

Due to the processor merging the data inputs are changed as presented in Fig. 12.

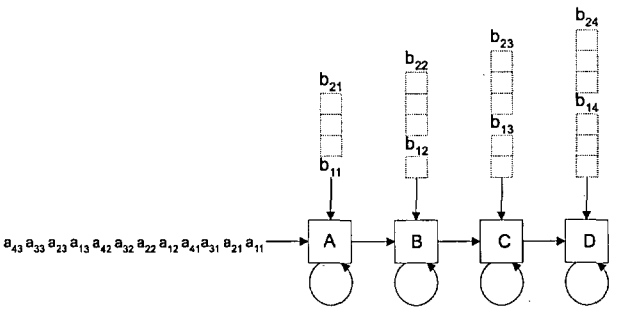


Figure 12: Data inputs in horizontal array

3.2 Vertical array

Vertical array is made when we merge the processors of the first row into processor A, processors of the second

row into processor B, etc, as presented in Fig. 13. Processors perform the same operations as when they were transformed into horizontal array.

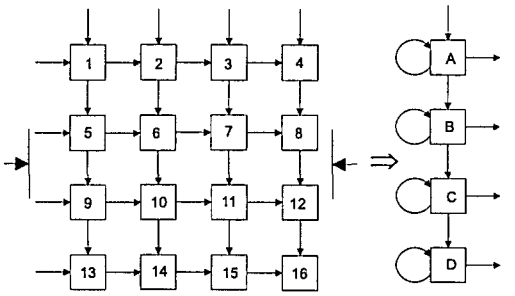


Figure 13: Transformation into vertical array

Occupation of processors is presented in Table 7 and changed data inputs are presented in Fig. 14.

Table 7: Processor occupation in vertical array

	A	B	C	D
1	1			
2	2	5		
3	3	6	9	
4	4	7	10	13
5	1	8	11	14
6	2	5	12	15
7	3	6	9	16
...				
17		8	11	14
18			12	15
19				16

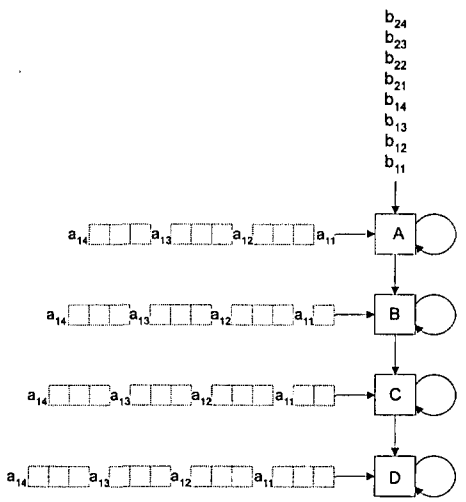


Figure 14: Data inputs in vertical array

Actually there is no significant difference between horizontal and vertical transformation, since all processors in two-dimensional array perform the same operations. Thus, it is insignificant what the contraction direction is, however we can choose which coefficients are delayed when entering the array.

3.3 Diagonal array

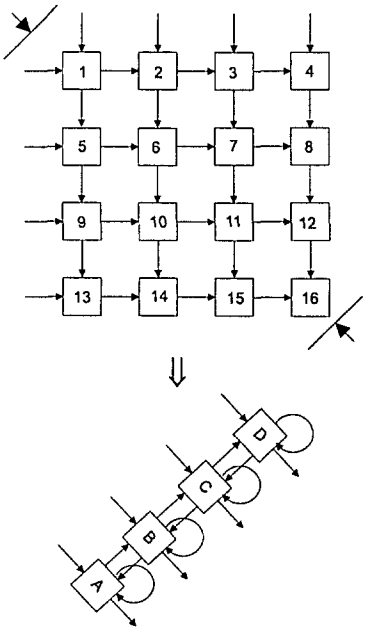


Figure 15: Diagonal transformation (n=4, n^*=4)

Due to the square array structure, diagonal transformation is a bit more complicated. According to the merging process, there can be different linear solutions, but only some typical will be presented in this paper.

If there is an even number of processors (e.g., n=4) in a two-dimensional array, we can choose between two possibilities.

In the first one, as presented in Fig. 15, the processor array is transformed as follows: processors 9, 13 and 14 are merged into processor A, processors 1, 5, 10, 11 and 15 are merged into processor B, processors 2, 6, 7, 12 and 16 are merged into processor C and processors 3, 4 and 8 are merged into processor D. So there is even number of processor (n^*=4) in linear processor array.

Table 8 represents the occupation of the processors, while data inputs are set as presented in Fig. 16.

In the second case, there is an odd number of processors (e.g., n^*=5) in the linear array. According to Fig. 15, processors are merged as follows: processors 9, 13 and 14 are merged into processor A, processors 5,10 and 15 are merged into processor B, processors 1, 6, 11 and 16 are merged into processor C, processors 2, 7 and 12 are merged into processor D and processors 3, 4 and 8 are merged into processor E.

Processor occupation is shown in Table 9, while Fig. 17 presents the data inputs.

But when there is an odd number of processors (n=5) in the two-dimensional array, the linear array consists of odd number of processors(n^*=5) . The situation is presented in Table 10.

Here processors 11, 16, 21, 22 and 23 are merged into processor A, processors 6, 12, 17, 18 and 24 into processor

Table 8: Processor occupation (n=4, n^*=4)

	A	B	C	D
1		1		
2		5	2	
3	9	1	6	3
4	13	5	2	4
5	9	1	6	3
6		10	2	
7	14	5	7	
8	13	1	6	8
9	9	10	2	4
10	14	5	7	3
11	13	11	6	8
12	9	10	12	4
13		15	7	3
14		11	16	
15	14	10	12	8
16	13	15	7	4
17	14	11	16	8
18		15	12	
19		11	16	
20		15	12	
21			16	

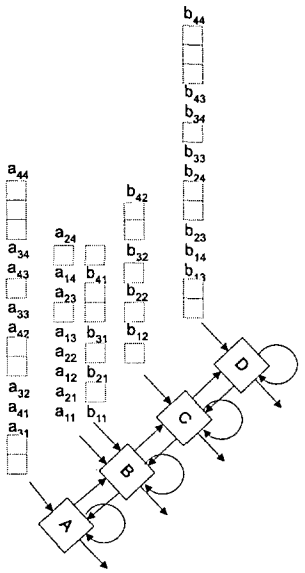


Figure 16: Data inputs (n=4, n^*=4)

Table 9: Processor occupation (n=4, n^*=5)

	A	B	C	D	E
1			1		
2		5	1	2	
3	9	5	6	2	3
4	13	10	1	7	4
5	9	5	6	2	3
6	14	10	1	7	8
7	13	5	11	2	4
8	9		6		3
9	14		11		8
10	13	15	6	12	4
11	9	10	16	7	3
12	14	15	11	12	8
13	13	10	16	7	4
14	14	15	11	12	8
15		15	16	12	
16			16		

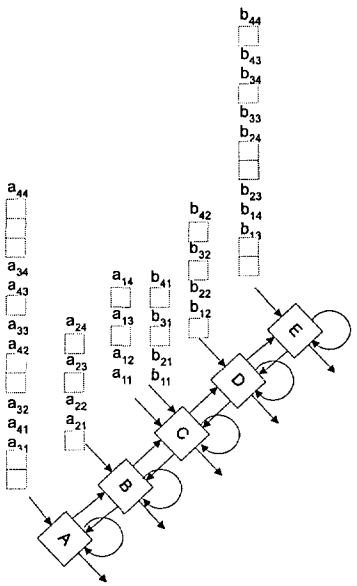


Figure 17: Data inputs (n=4, n^*=5)

B, processors 1, 7, 13, 19 and 25 into processor C, processors 2, 8, 9, 14 and 20 into processor D and processors 3, 4, 5, 10 and 15 into processor E.

Data inputs have to be set according to the new processor utilization, as presented in Fig. 18.

4 Conclusions

According to the results, there are important differences when transforming original triangular array in different directions and with different mirrorings. The difference is in execution time, processor utilization and complexity of processor's operations. Table 11 represents characteristics of $n = 4$ and $n = 5$ arrays. Different transformations are considered (horizontal, vertical, diagonal, interweaved) and different mirrorings (processor A mirrored into processor B, processors A and B mirrored into processor C, ...). Number of steps is the number of systolic cycles needed to perform the algorithm. Number of processors is the number of needed processors, and utilization is their use according to the number of steps (min and max utilization represent smallest and largest utilization of a single processor).

As it can be seen in Table 11 and Fig. 19, mirroring improves the differences between the smallest and largest processor utilization in the array.

Table and figure show these conclusions:

- The number of steps, to execute the algorithm, increases with the transformation, but the number of processors decreases significantly, while their utilization is increased.

- When transforming triangular arrays with even number of processors in the first row of the array, the best transformation is diagonal one with mirroring. Diagonal transformation is the best even if there is no mirroring.

Table 10: Processor occupation (n=5, n^*=5)

	A	B	C	D	E
1			1		
2		6	1	2	
3	11	6	1	2	3
4	11	6	7	2	3
5	16	12	1	8	4
6	11	6	7	2	3
7	16	12	1	8	4
8	21	17	7	9	5
9	11	6	13	2	3
10	16	12	7	8	4
11	21	17	13	9	5
12	22	18		14	10
13	11				3
14	16	12	7	8	4
15	21	17	13	9	5
16	22	18	19	14	10
17	23				15
18	16	12		8	4
19	21	17	13	9	5
20	22	18	19	14	10
21	23	24	13	20	15
22	21	17	19	9	5
23	22	18	25	14	10
24	23	24	19	20	15
25	22	18	25	14	10
26	23	24	19	20	15
27	23	24	25	20	15
28		24	25	20	
29			25		

- When transforming triangular arrays with odd number of processors in the first row of the array, the best transformation is interweaved, while it offers largest utilization and needs only a few processors.
- When transforming square arrays any transformation is better than initial array. Since all processors perform the same operations it is irrelevant in which direction we contract the array, but horizontal or vertical arrays are much simpler to implement than diagonal.
- Processor utilization can be even higher if there are consecutive multiplication computations used one after another.
- In all untransformed arrays the number of steps is defined as $3n-2$ and the number of processors is $\frac{n(n+1)}{2}$ in triangular and n^2 in square arrays, where $n \times n$ is the size of matrix.
- In transformed arrays the number of steps is defined as n^2+n-1 and the number of processors is n.

In some common problems there are very big matrices, e.g., 250 × 250, which lead to the large number of processors required. Therefore in those cases it is appropriate to use also some other techniques with even fewer number of processors [5, 10].

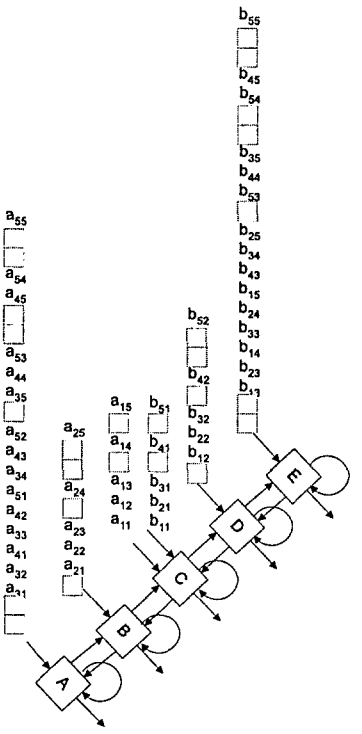


Figure 18: Data inputs (n=5, n^*=5)

References

[1] H.Barada, A.El-Amawy, Systolic architecture for matrix triangularisation with partial pivoting, *IEEE Proc., Vol. 135, Pt. E*, No. 4, July 1988, pp. 208-213.

[2] P.Blaznik, J.Tasič, D.J.Evans, Parallel Solving the Updated Linear Systems of Equations, *F.Solina and B.Zajc (ed.), Proceedings of the second Electrotechnical and Computer Science ERK'93*, Volume B, 1993, pp. 115-118.

[3] J.Kaniewski, O.Maslennikow, R.Wyrzykowski, VLSI implementation of linear algebraic operations based on the orthogonal Fadeev algorithm, *Parallel Computing: State-of-the-Art and Perspectives*, Elsevier, 1996, pp. 641-644.

[4] S.Y.Kung, *VLSI Array Processors*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[5] J.G.Nash, S.Hansen, Modified Faddeeva Algorithm for Concurrent Execution of Linear Algebraic Operations, *Proc. IEEE Transactions on Computers*, Vol. 37, No. 2, February 1988, pp. 129-136.

[6] J.G.Nash, C.Petrozolin, VLSI Implementation of a Linear Systolic Array, *Proc. 1985 Int. Conf. Acoust., Speech, Signal Processing*, Tampa, FL, pp. 1392-1395.

[7] N.Petkov, *Systolic Parallel Processing*, North-Holland, Amsterdam, 1993.

[8] P.Quinton, Y.Robert, *Systolic Algorithms & Architectures*, Prentice-Hall, UK, 1989.

[9] R.Wyrzykowski, Processor arrays for matrix triangularisation with partial pivoting, *IEEE Proc.-E*, Vol. 139, No. 2, March 1992, pp. 165-169.

Table 11: Array efficiency

	number of		processor utilization (%)		
			overall	single	
	steps	procs.		min	max
n=4					
triangular	10	10	40.0	40.0	40.0
- horizontal	19	4	52.6	21.1	84.2
A into B	19	3	70.2	63.2	84.2
A into D, B into C	27	2	74.1	74.1	74.1
- vertical	19	4	52.6	21.1	84.2
D into C	19	3	70.2	63.2	84.2
D into A, C into B	27	2	74.1	74.1	74.1
- diagonal	16	4	62.5	25.0	100.0
D into A, C into B	24	2	83.3	83.3	83.3
square	10	16	40.0	40.0	40.0
- horizontal	19	4	84.2	84.2	84.2
- vertical	19	4	84.2	84.2	84.2
- diagonal (n*=4)	21	4	76.2	57.1	95.2
- diagonal (n*=5)	16	5	80.0	75.0	100.0
n=5					
triangular	13	15	38.5	38.5	38.5
- horizontal	29	5	51.7	17.2	86.2
A into B	29	4	64.7	51.7	86.2
A and B into C	34	3	73.5	58.8	88.2
- vertical	29	5	51.7	17.2	86.2
E into D	29	4	64.7	51.7	86.2
E and D into C	34	3	73.5	58.8	88.2
- diagonal	29	5	51.7	17.2	86.2
D into C, E into B	41	3	60.9	60.9	60.9
- interweaved	29	3	86.2	86.2	86.2
square	13	25	38.5	38.5	38.5
- horizontal	29	5	86.2	86.2	86.2
- vertical	29	5	86.2	86.2	86.2
- diagonal (n*=5)	29	5	86.2	86.2	86.2

[10] R.Wyrzykowski, Y.Kanevski, S.Ovramenko, Dependence graph transformations in the design of processor arrays for matrix multiplications, *Microproces. & Microprogram.*, Vol. 135, 1992, pp. 534-539.

[11] R.Wyrzykowski, J.S.Kanevski, H.Piech, One-dimensional processor arrays for linear algebraic problems, *Proc. Comput. Digit. Tech.*, Vol. 142, No. 1, January 1995, pp. 1-4.

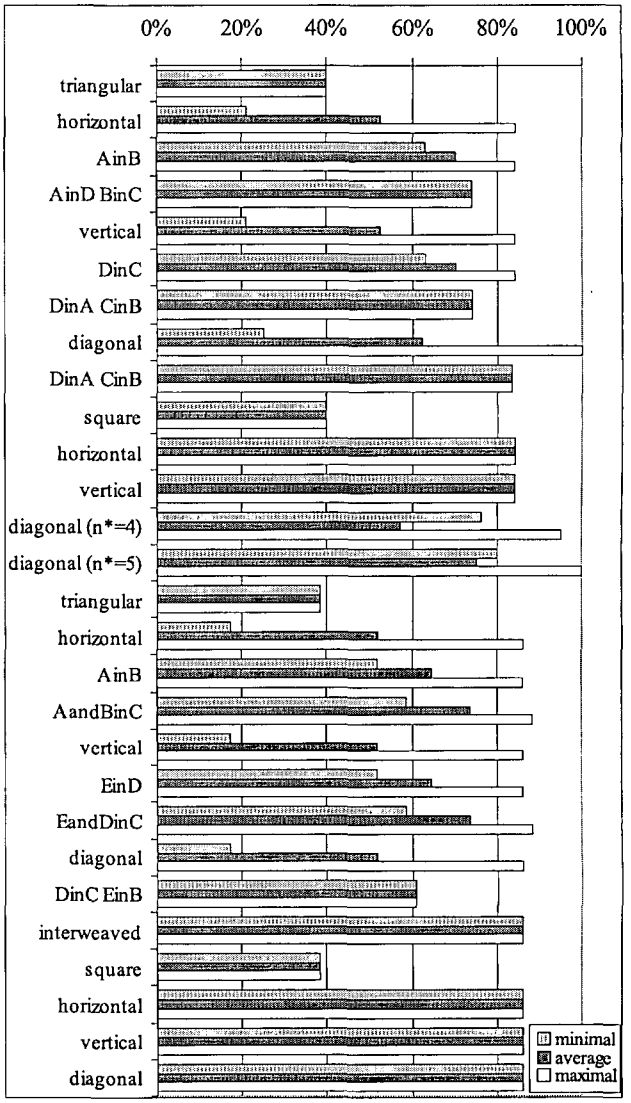


Figure 19: Arrays efficiency