Optimization of the Parallel Matrix Multiplication

GREGOR PAPA, JURIJ ŠILC
Computer Systems Department
"Jožef Stefan" Institute
Jamova 39, 1000 Ljubljana
SLOVENIA

Abstract: - Many frequently met problems of linear algebra, such as the solution of linear systems and matrix multiplication, are computationally expansive. Those problems need to be solved with parallel structures, such as systolic arrays, where many processors are used concurrently to compute the result. The parallelism decreases the computing time. In the case of matrix multiplication it is very appropriate to use two-dimensional array of processors, that reflects the real situation of computing the coefficients. But, since two-dimensional array of processors is very space- and resource-consumptive, it is better to use one-dimensional array of processors. Although this leads to the problem of operation reallocation and unequal utilization of the processors, it is easier to implement since there is only one straight line of processors. Beside the simplicity of implementation the utilization of processors is also higher. This paper presents the evaluation of different approaches to the linearization of the two-dimensional multiplication array.

Key-Words: - optimization, matrix, multiplication, parallel, systolic, linearization

1 Introduction

Many problems in science can be solved by linear algebraic computation, but even some basic problems are computationally expansive. To shorten the computing time, new structure has to be build, which enables synchronous data computing. According to the nature of the problem, where matrix transformation is the main goal, parallel systolic structure is very appropriate. With the group of processors it enables more data to be processed synchronously. Detailed structure and function of systolic structure will be presented in the next section.

Beside the advantages of this approach (shorter execution time) there are also some disadvantages. It is difficult to compose a net of processors, since there is a lot of connections and it is difficult to monitor all of them and finally to read data out of them at the end. They are also poorly utilized, since they mostly wait for their input data.

It is possible to compose the structure with higher utilization, economic justification, time suitability and lower complexity, which would remove disadvantages mentioned in the previous paragraph. Namely, some processors can be merged, so that one processor performs tasks of more processors, and also their access is easier since there is just one straight array of the processors [4].

This work presents different approaches to the matrix multiplication and the comparison of two-dimensional and linear processor arrays is given.

2 Systolic arrays

Systolic solving is presented by the processor structure, where data is flowing through the processors. Systolic array is a net of specialized processors, which are locally connected and work synchronized. Its use is appropriate when solving time-consuming problems. The characteristics of this array are:

- synchronization (computing and data-flow are performed rhythmically),
- modularity and regularity (consists of modular and connected elements, which can be illimitable added),
- area and time localization (between processors there are local connections with delays),
- speed-up factor (faster execution when performed over many processors).

To get a perfect systolic array a convenient approach is needed. One of them is graph-based design methodology [3, 1]. It is based on the graph representation and consists of three steps:

- dependency graph design; a suitable algorithm for the problem has to be identified and dependency graph is made. Since this graph affects the final design, some changes are possible to improve the result,

- signal flow graph design; according to the projection of the algorithm, there is more types of this kind of a graph,
- array processors design; signal flow graph is mapped into the processor array.

Dependency graph describes dependencies between data in different processors. If there is no data dependency between two operations they can be executed concurrently. Very important step in the design is scheduling, while it affects the processor utilization [6, 5].

Nodes of the graph represent arithmetic or logic operation without delay, and connections represent time delays and data dependencies. Mapping is performed in more steps. First, processors are determined to perform single operations with minimal communications, and then the execution order is made to minimize total execution time. Basic steps:

- determination of the basic operation modules; systolic array is more effective (faster), if the algorithm is broken into small pieces,
- use of time rules; the graph is time-limited,
- combination of delays and operation modules; basic systolic element is made of delay and operation module.

Signal flow graph is mapped into systolic array, where operation modules are mapped into processors and their connections are mapped into processor connections.

2.1 The use of systolic arrays

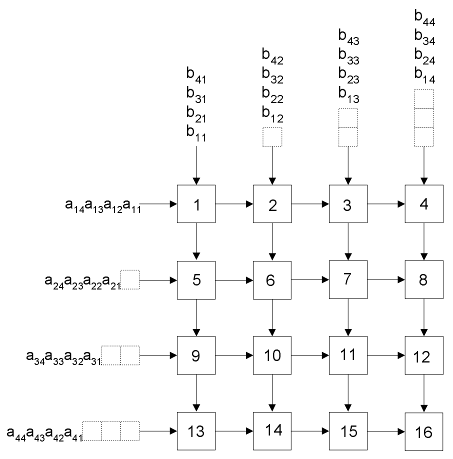
Systolic arrays can be used when performing matrix multiplication [2] of the form

$$C = A\,B + C_0\,.$$

Array of processors for multiplication of two square matrices is presented in Fig. 1 [7]. Inputs of the structure are matrices' coefficients (a_{ij} in b_{ij}) and at the end there are coefficients c_{ij} inside the structure. According to the matrix size $n \times n$ the number of required processors n^* is:

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

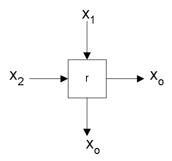
Shape □ represents a processor and □ represents a delay τ.



b_{41} b_{31} b_{21} b_{11}
b_{42} b_{32} b_{22} b_{12}
b_{43} b_{33} b_{23} b_{13}
b_{44} b_{34} b_{24} b_{14}
$a_{14}a_{13}a_{12}a_{11}$
$a_{24}a_{23}a_{22}a_{21}$
$a_{34}a_{33}a_{32}a_{31}$
$a_{44}a_{43}a_{42}a_{41}$
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Fig. 1: Square systolic array (n=4)

All processors in square array in Fig. 1 perform the same operations [7]:



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

Next chapters present some transformations of two-dimensional arrays into linear arrays. There is also the comparison of data flow, utilization of processors and complexity of needed operations.

3 Matrix multiplication

3.1 Horizontal array

Horizontal array is made 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. 2. Processors perform the same operations, as before the transformation, beside there is an additional input from one of its outputs.

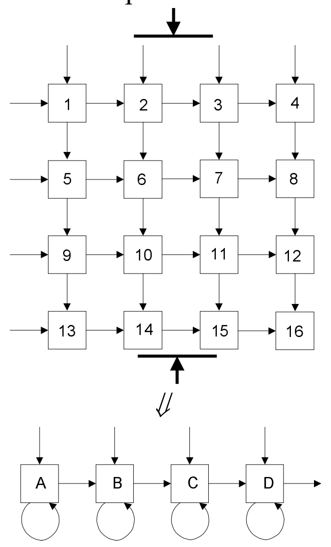


Fig. 2: Transformation into the horizontal array

Occupation of the processors is presented in Table 1, where shadowed field represents the time frame when the processor is occupied.

	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

Table 1: Occupation of processors in horizontal array

Due to the processor merging the data inputs are changed as presented in Fig. 3.

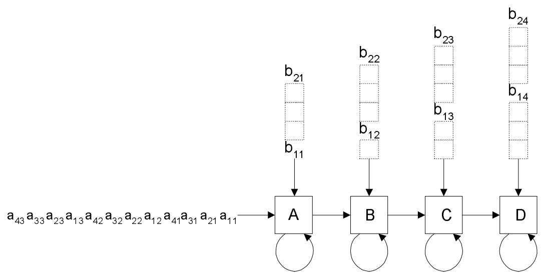

$a_{43}a_{33}a_{23}a_{13}a_{42}a_{32}a_{22}a_{12}a_{41}a_{31}a_{21}a_{11}$
b_{21}
b_{11}
b_{22}
b_{12}
b_{23}
b_{13}
b_{24}
b_{14}
A
B
C
D

Fig. 3: Data inputs in horizontal array

3.2 Vertical array

Vertical array is made when merging processors of the first row into processor A, processors of the second row into processor B, etc, as presented in Fig. 4. Processors perform the same operations as when transformed into horizontal array.

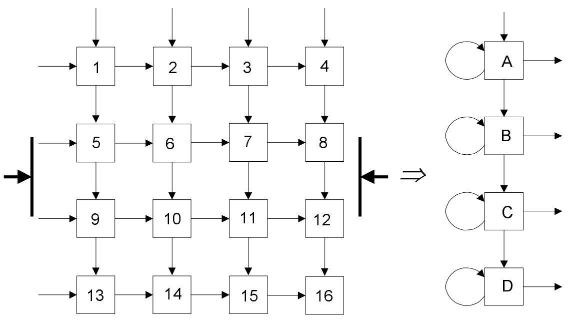


Fig. 4: Transformation into vertical array

Occupation of processors is presented in Table 2 and data inputs are changed as presented in Fig. 5.

	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

Table 2: Occupation of processors in vertical array

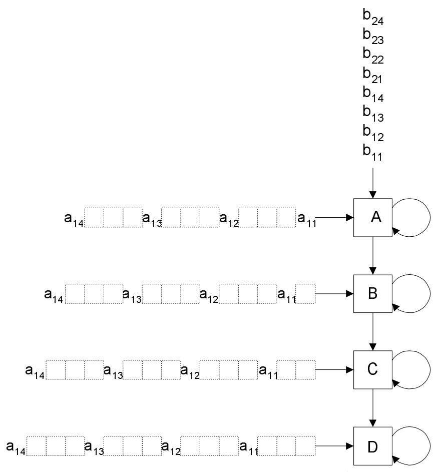


Fig. 5: Data inputs in vertical array

Actually there is no 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, but we can choose which coefficients are delayed when entering the array.

3.3 Diagonal array

Because of the array structure (square), diagonal transformation is a bit more complicated. According to the merging process, there can be different linear solutions.

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

In the first one, as presented in Fig. 6, 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 an even (*n*=4*) number of processor in linear processor array.

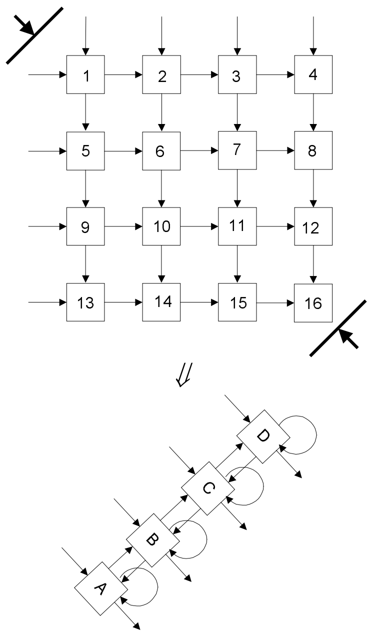


Fig. 6: Transformation into diagonal array

Table 3 represents the occupation of the processors, while data inputs are changed as presented in Fig. 7.

	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	

Table 3: Processor occupation (n=4, n=4)*

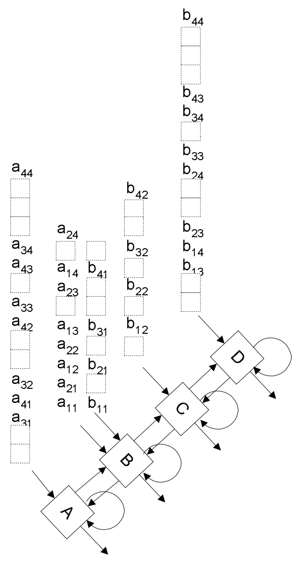


Fig. 7: Data inputs in diagonal array n=4, (n=4)*

In the second case, there is an odd (*n*=5*) number of processors in the linear array. According to Fig. 6, 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.

	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		

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

Processor occupation is shown in Table 4, while Fig. 8 presents the data inputs.

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

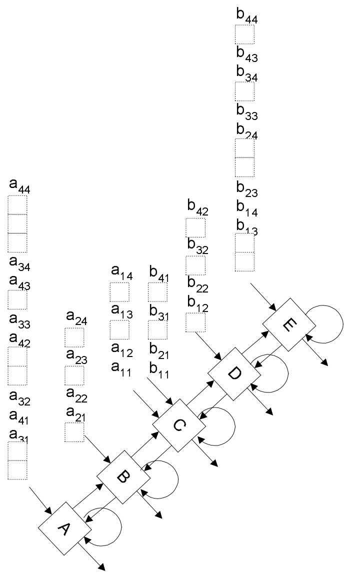


Fig. 8: Data inputs in diagonal array (n=4, n=5)*

	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		

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

Transformation is obtained by merging processors 11, 16, 21, 22 and 23 into processor A, processors 6, 12, 17, 18 and 24 into processor 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 has to be set according to the new processor utilization, as presented in Fig. 9.

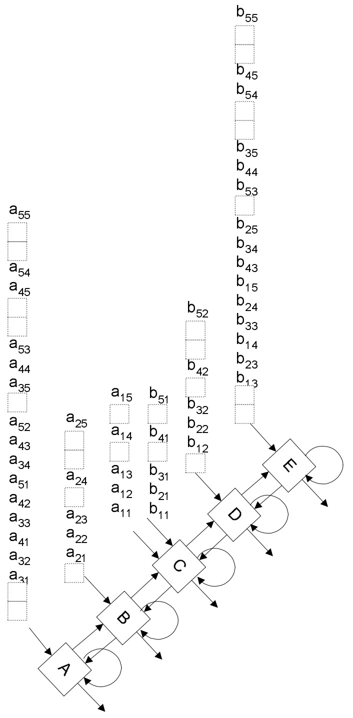


Fig. 9: Data inputs in diagonal array (n=5, n=5)*

4 Conclusion

According to the results, there are important gains when transforming two-dimensional array in different directions. The difference is in execution time and processor utilization.

Table 6 represents the characteristics of *n=4* and *n=5* arrays. Different transformations are considered (horizontal, vertical, and diagonal). 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.

The number of steps, to execute the algorithm, increases with the transformation, but the number of processors decreases significantly.

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 according to the simplicity of the implementation horizontal and vertical linear arrays are better than any diagonal array (Figs. 3, 5, 7, 8, 9). Data inputs are simpler and the utilization of the processors is much higher in horizontal or vertical array.

	number of steps	number of processors	processors utilization
square (*n=4*)	10	16	40.0%
- horizontal	19	4	84.2%
- vertical	19	4	84.2%
- diagonal (*n*=4*)	21	4	76.2%
- diagonal (*n*=5*)	16	5	80.0%
square (n=5)	13	25	38.5%
- horizontal	29	5	86.2%
- vertical	29	5	86.2%
- diagonal (*n*=5*)	29	5	86.2%

Table 6: Efficiency of the arrays

With the transformation the number of control steps is increased, but the number of processors is significantly decreased and their utilization is increased. Thus, when the number of processors and other components is constrained, linearization of the processor array is very appropriate.

References:

[1] P.Blaznik, *Parallel Algorithms on Systolic Array (in Slovene)*, M.Sc. Thesis, Faculty of Electrotechnical and Computer Science, University of Ljubljana, Slovenia, 1991.

[2] P.Blaznik, J.Tasič, D.J.Evans, *Parallel Solving the Updated Linear Systems of Equations*, Proc. 2nd Electrotechnical and Computer Science Conference ERK'93, Volume B, 1993, pp. 115-118.

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

[4] 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.

[5] J.J.Navarro, J.M.Llaberia, M.Valero, *Partitioning: An Essential Step in Mapping Algorithms Into Systolic Array processors*, Computer, Vol. 20, No. 7, July 1987, pp. 77-90.

[6] G.Papa, J.Šilc, F.Bratkovič, *Scheduling Algorithms in High-Level Synthesis – overview and evaluation*, Electrotechnical Review, Vol. 65, No. 4, 1998, pp. 153-165.

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

[8] R.Wyrzykowski, Y.Kanevski, S.Ovramenko, *Dependence Graph Transformations in the Design of Processors Arrays for Matrix Multiplications*, Microprocessing and Microprogramming, No. 35, 1992, pp. 539-544.