Please cite this article as:
Henryk Piech, Michał Styś, Serialization of input and output data transfers in parallel structures, Scientific Research of the Institute of Mathematics and Computer Science, 2007, Volume 6, Issue 1, pages 241-252.
The website: http://www.amcm.pcz.pl/

SERIALIZATION OF INPUT AND OUTPUT DATA TRANSFERS IN PARALLEL STRUCTURES

Henryk Piech, Michał Styś

Institute of Mathematics and Computer Science, Czestochowa University of Technology, Poland
email: hpiech@adm.pcz.czest.pl

Abstract. We propose the new interpretation of feature named "serialization" as a characteristic of scheduling algorithms. In our interpretation serialization would be the property of algorithm referring to accumulating date in processes as well as to summary procedure of sending date to chosen place (processor or process) directly before their utilization. We want to practically prove that serialization can help us to enlargement the effectiveness of scheduling procedures.

Introduction

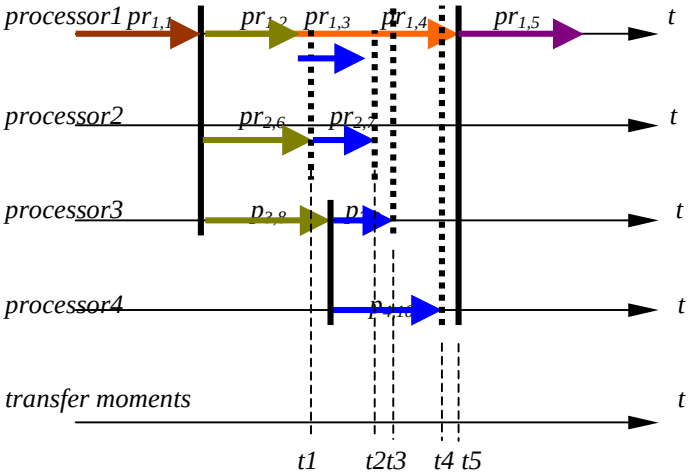
Communication in a distributed and parallel environment is seen as a necessity relating to an additional loss of time, but necessary to obtain the final effect consisting in the effective shortening of global processing time [1]. Communication is related to the establishment of connections and the quantity of information which is supposed to be transmitted in individual connection sessions [2]. An essential thing is to prepare and present data in such a way that it will be possible to send it conveniently (the best as a series) and at one time in the largest chunks (serializable) [3]. The more frequent and fragmented this process is, the more time it will take. It is even related to establishing the procedures of connections.

1. Indexes and examples of serialization in parallel processes

The possibility of establishing connections between many nodes (processors) is profitable and it can favourably influence the shortening of communication time. Certainly, the number of connections and information transfers depends on the realization algorithm of concrete task but also on a realization algorithm of transfer, i.e. the communication algorithm (Fig. 1). The fork and message transfer of threads to processors 2 and 3 follow the realization of process pr1,1. The results of the pr2,1 process will determine the data of the pr1,5 process. Thus, they could to be sent to processor 1 at moment t1 (Fig. 1b). However, it is not the optimal variant, because it leads to additional waste of time.

a)

Variant 1



$pr_{i,j}$ - *process with number "j" realized on i-th processor*
t_i – *potential moments of realization of information transfer to the 1-st processor*

b)

Variant 2

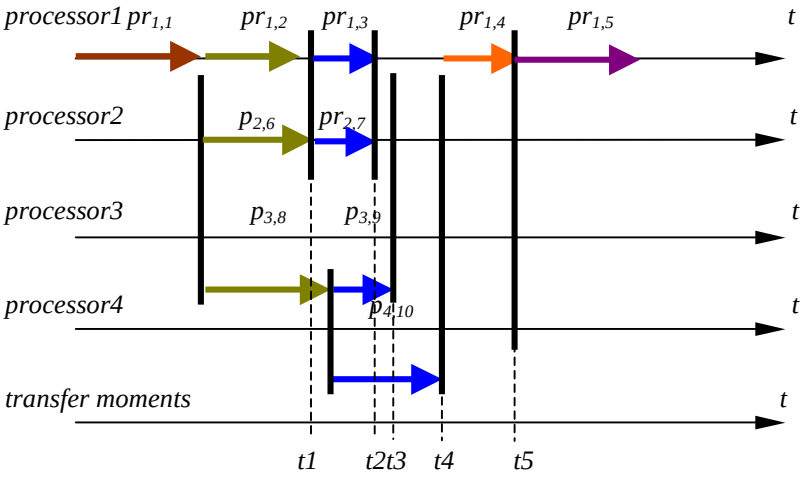


Fig. 1. Realization ways of information transfer between processors (variant 1, variant 2)

In variant 1 the moment when all data for the pr1,5 process will be determined is anticipated (the moment t5). Then, a connection with processor 1 is established and the transfer of data series from the other processors occurs. This feature will be called the series creation capability (serializable). It relates to the executed task

algorithm in combination with the communication algorithm. The series creation capability can be expressed by the number of data fed into a given process in relation to the number of independent processes which create these data:

$$SLA = 1/mp * \sum_{i=1}^{m} \{1/n(i) * \sum_{j=1}^{n(i)} \delta(i, j) D(i, j)\} \quad (1)$$

where:

SLA - the series creation capability: serialization factor,

D(i,j) - data number determined in uniform or autonomous units and transferred directly from the j-th processor to the i-th process,

mp - number of all processes,

n(i) - number of processors feeding information to i-th process,

$\delta(i,j)$ - transfer factor.

Thus, it is not significant whether the data come from a single or many processes, however, it is important that the data be directly fed before the initiation of the i-th process. If the data is transferred earlier than directly before the initiation of a given process then they are not taken into consideration in the formula (1). It is taken into consideration thanks to the transfer factor (i,j).

$$\delta(i,j) = \begin{cases} 1, \text{ if the data transfer from the j-th processor to the i-th process directly precedes the given process,} \\ 0, \text{ if the data transfer from the j-th processor to the i-th process is earlier or if the process is realized on the j-th processor.} \end{cases}$$

Example 1

The situation presented in Figure 1a and 1b is considered.

Variant 1

Table 1

Values of transfer factors $\delta(i,j)$

	1proc.	2proc.	3proc.	4proc.	5proc.	6proc.	7proc.	8proc.	9proc.	10proc.
processor1	0	0	0	0	0	1	0	1	0	0
processor2	0	0	0	0	1	0	0	0	0	0
processor3	0	0	0	0	1	0	0	0	0	1
processor4	0	0	0	0	1	0	0	0	0	0

Table 2

Number of processors feeding information to the given process n(i)

	1proc.	2proc.	3proc.	4proc.	5proc.	6proc.	7proc.	8proc.	9proc.	10proc.
n(i)	0	0	0	0	3	1	0	1	0	1

Table 3

Data number transferred to successive processes D(i,j)

	1proc.	2proc.	3proc.	4proc.	5proc.	6proc.	7proc.	8proc.	9proc.	10proc.
processor1	n+2	n	n	n	n	n	0	n	0	0
processor2	0	0	0	0	3	m	m	0	0	0
processor3	0	0	0	0	n	0	0	n+1	n+1	2
processor4	0	0	0	0	n	0	0	0	0	0

Variant 2

Table 4

Values of transfer factors δ(i,j)

	1proc.	2proc.	3proc.	4proc.	5proc.	6proc.	7proc.	8proc.	9proc.	10proc.
processor1	0	0	0	0	0	1	0	1	0	0
processor2	0	0	0	0	0	0	0	0	0	0
processor3	0	0	0	0	0	0	0	0	0	1
processor4	0	0	0	1	0	0	0	0	0	0

Table 5

Number of processors feeding information to the given process n(i)

	1proc.	2proc.	3proc.	4proc.	5proc.	6proc.	7proc.	8proc.	9proc.	10proc.
n(i)	0	0	0	1	0	1	0	1	0	1

Introducing the exemplifying data to the formula (1) we receive:

$$SLA_{variant1} = 1/mp* \sum_{i=1}^{m}\{1/n(i)* \sum_{j=1}^{n(i)} \delta(i, j)D(i, j)\} = 1/40*(6*n+4*m+11)$$

$$SLA_{variant2} = 1/mp* \sum_{i=1}^{m}\{1/n(i)* \sum_{j=1}^{n(i)} \delta(i, j)D(i, j)\} = 1/10(m+n+2) = 1/40*(4*n+4*m+8) \quad (2)$$

Comparing both serialization factors apart from the obvious inference:

$$SLA_{variant1} > SLA_{variant2}$$

the pace of the serialization factor changes can be determined by the increase in the value of m and n data numbers (Fig. 2).

2. Application of analysis of the series creation capability with reference to linear transportation algorithms

It is possible to divide the transportation algorithm into processes depending on the data sets used in these processes. Such a segregation of processes is effective and is conducive to the minimization of interprocessor communication. Thus, it is possible to separate, for example, the following processes [4]:

1. searching for the minimal unit values of carriage costs in demand columns (1proc.),
2. determination of a transport order (2proc.),
3. realization of "transports" (3proc.),
4. verification of a distribution end (4proc.),
5. division into "priority" and "non-priority" classes (5proc.),
6. determination of a correction quantity in the priority class (6proc.),
7. correction of unit carriage costs of the priority class (7proc.) and return to the first process.

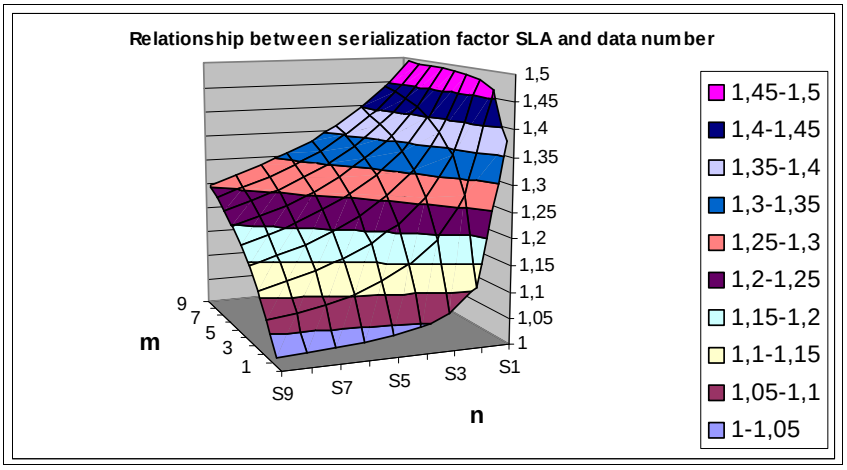


Fig. 2. Relationship between serialization factor and number of transferred data (expressed in their sizes)

The data for process 1 is the matrix size of the unit carriage costs and elements of this matrix (m x n) + 2, where: m is the number of suppliers and n is the number of receivers.The data for process 2 are n vectors with the number of elements equal to m. Forking this process to the n subprocesses (2.1-2.n), let us deliver the m+1 data to each of them. Process 3 is the use of the results from the subprocesses (2.1-2.n) and let us deliver nx(m+2)+1 data to it. The next process (process 4) requires information related to commodities not transferred, which is registered in the vector with length of n ((n+1) data). The 5-th process requires full information on carriages (it uses (m+1)x(n+1)+2 data). The 6-th process can also be forked into 6 subprocesses (6.1-6.n). Each of the processes use (m+1) data. In process 7 the unit carriage costs in the priority group are corrected. The average number of data can

be estimated as m/2*n+1. The graphical picture of the transportation algorithm can be presented as it is in Figure 3.

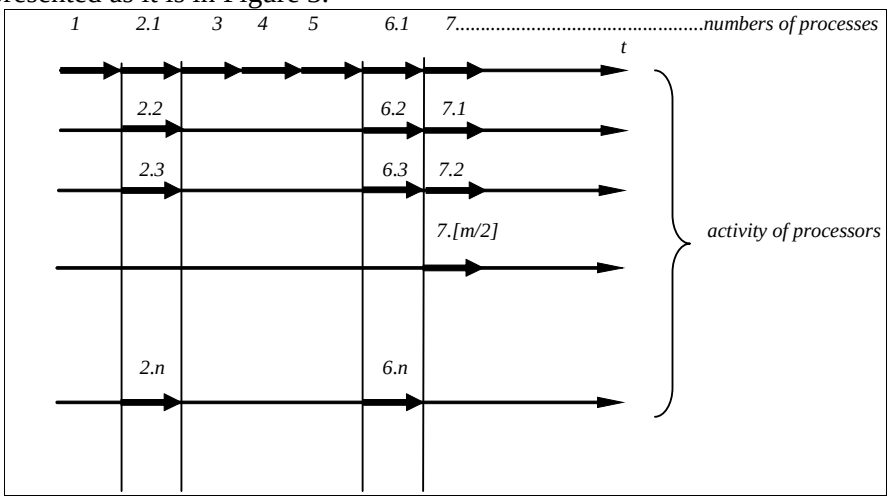


Fig. 3. Distribution sample of processes in the transportation task according to the above description

The transfer factors and the ranges of transferred data for the example from Figure 3 are described in Tables 6 and 7.

Table 6

Transfer factors δ(i,j)

	1proc.	2y_proc.	3proc.	4proc.	5proc.	6proc.	7proc.
processor1	0	1	0	0	0	1	0
processor2	0	0	1	0	0	0	1
.....	0	0	1	0	0	0	1
processor n	0	0	1	0	0	0	1

Table 7

Transferred data D(i,j)

	1proc.	2y_proc.	3proc.	4proc.	5proc.	6proc.	7proc.
processor1	mxn+2	m+1	m+n+2	n+1	(m+1)x(n+1)+2	m+1	m/2*n+1
processor2	0	0	m+1	0	0	0	m/2*n+1
.....	0	0	m+1	0	0	0	m/2*n+1
processor n	0	0	m+1	0	0	0	m/2*n+1

The serialization factor for the exemplifying variant of communication amounts to:

$$SLA_{example1} = 1/mp * \sum_{i=1}^{m} \{1/n(i) * \sum_{j=1}^{n(i)} \delta(i, j) D(i, j)\} = 2(m+n)/(5+2n) \quad (3)$$

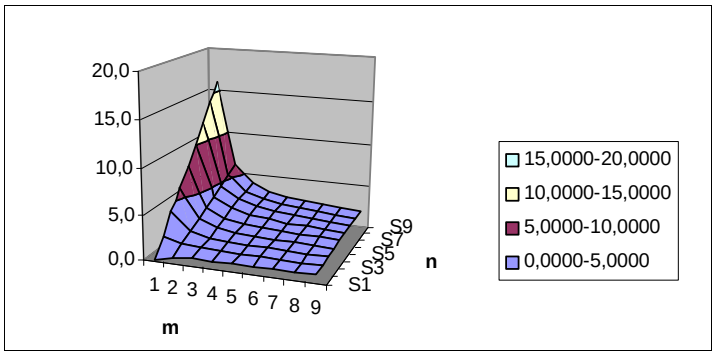


Fig. 4. Relationship between serialization factors and data numbers (expressed in their sizes)

The next example is the search algorithm of the shortest path between two points in an undirected graph. The course of this problem can be divided into the following processes:

1. searching for the node in the nearest distance (v) from the current node (starting point s) (1proc.),
2. verification of the end (the chosen node is the end-point v = t) (2proc.),
3. addition of the found edge to the set of selected edges (3proc.),
4. searching for the node in the nearest distance from the already selected set and return to process 2 (4proc.).

The graphical picture of the search algorithm of the shortest path can be shown in the following way (Fig. 5).

The transfer factors with taking into consideration the transfer iteration in the process coded “4” to process “2”, i.e. after the addition of the successive found edge, are presented in Table 8.

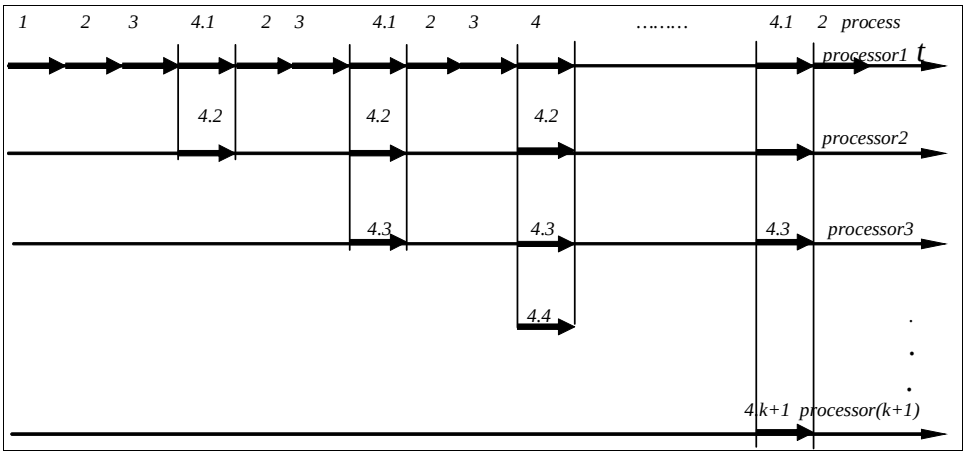


Fig. 5. Distribution of processes in the problem of searching for the shortest path between points s-t, where: k - the number of edges connecting vertices s and t

Table 8

Transfer factors δ(i,j)

	1proc.	2proc.	3proc.	4proc_y
processor1	0	0	0	0
processor2	0	1	0	1
……	0	1	0	1
processor k	0	0	0	1

After finding out the successive vertex, the next processor is activated to which information on its distance from the remaining vertices is transferred. Table 9 illustrates it. Feeding the neighbourhood matrix with the nxn size, where n is the number of vertices, and codes of vertices of the start and end of the shortest path to process 1 is required. Process 2 requires feeding the codes of the current and last vertex. The addition of the found edge to the current path structure takes place in process 3. The data number is equal one (it is only the code of vertex). The distances from the created structure are used in process 4, i.e. the n-elements vector as well as the vertex code and the graph size in the activated processor are available.

Table 9

Values of transferred data D(i,j)

	1proc.	2proc.	3proc.	4proc_y
processor1	nxn+4	2	1	n+2
processor2	0	2	0	n+2
……	0	2	0	n+2
processor k	0	0	0	n+2

The value of the serialization factor can be estimated as follows:

$$SLA_{example1} = 1/mp* \sum_{i=1}^{m} \{ 1/n(i)* \sum_{j=1}^{n(i)} \delta(i, j)D(i, j) \} = 2/1*(n+2)+2/2*(n+2)+ ...$$

$$+2/(k)* (n+2)/(2k*(1+2+...+k)) = (n+2)*(1+1/2+...+1/k)/ (k*(1+2+...+k)) \quad (4)$$

Reference to the number of processors does not fully reflect the ability to create a series. Another way would be to relate to the number of transfers. This number will be associated with the connection sessions but not with the number of processes. In that case, the serialization factor could be determined by the following expression:

$$SLI = 1/lp \sum_{i=1}^{lp} Dp(i) \quad (5)$$

where:

lp - number of connection sessions,

Dp(i) - data number transferred during *i*-th session.

$$SLI_{example1}=1/lp \sum_{i=1}^{lp} Dp(i) =1/4*(n-1)(m/2*n+3*m+4)$$

$$SLI_{example2}=1/lp \sum_{i=1}^{lp} Dp(i) =1/4*(n+4)*(1+k)$$

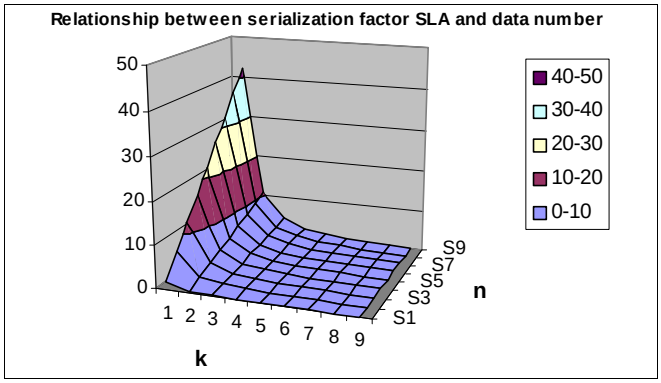


Fig. 6. Relationship between serialization factors and input data parameters

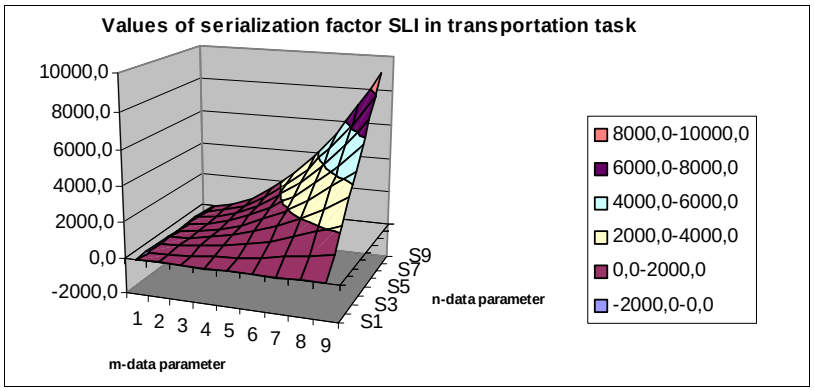


Fig. 7. Example 1 - relationship between serialization factor and data parameters

It is possible to select the level (alpha) of symmetrical distribution of the function form taking both serialization elements (SLA i SLI) into consideration:

$$SL=alpha*SLA+(1-alpha)*SLI \quad (6)$$

Example 1

alpha = 0.4
alpha = 0.7
alpha = 0.9 $\mathbf{alpha_{sym}} = 0.66$

Example 2

alpha = 0.001
alpha = 0,0015
alpha = 0,002 $\mathbf{alpha_{sym}} = 0.0016$

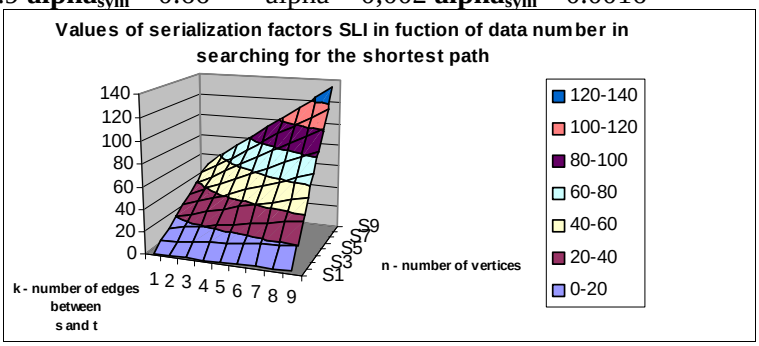


Fig. 8. Example 2 - relationship between serialization factor and data parameters

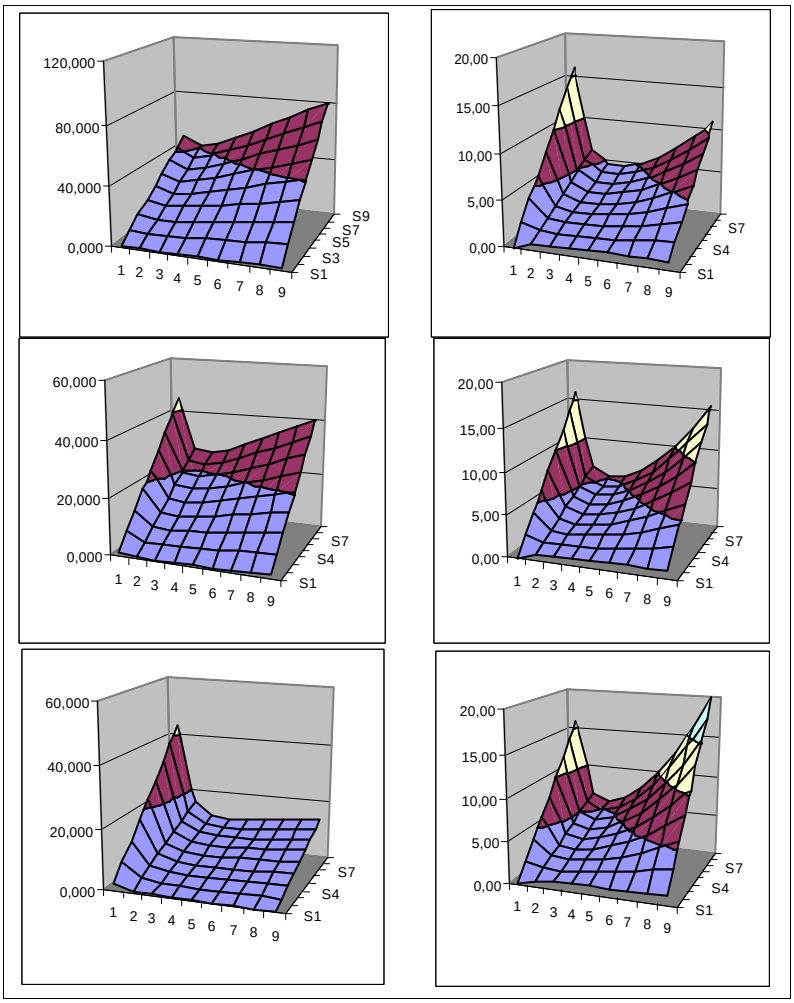


Fig. 9. Operations heading for equalization (middle figures) of influences of SLA and SLI elements (from formula (6))

The symmetrization ($alpha_{sym}$ → Figure 10) of the levels of SLA and SLI components leads to the equalization of property influences which the components represent. If the alpha level exceeds 0.5 then the influence of the first component, i.e. SLA from the formula (6) was increased. The first component characterizes the algorithm capability to the accumulation of series in separate nodes preceding a direct use of data. If the alpha level is smaller than 0.5 then the influence of the second component (SLI) was increased, i.e. the component which reflects the algorithm capability to collect and transfer of data from various processors directly before the use of data. An increase in the participation of one of the components to reach the balance (e.g. the increase in alpha value) is a consequence of the operation heading for a complement of structural lacks of serializable properties of concrete algorithm in a given range.

alpha>0.5 $\Rightarrow$ SLA_was_too_small (participation of SLA was increased)
alpha=0 $\Rightarrow$ SLA_and_SLI_have_equal_influences (7)
alpha<0.5 $\Rightarrow$ SLI_was_too_small (participation of SLI was increased)

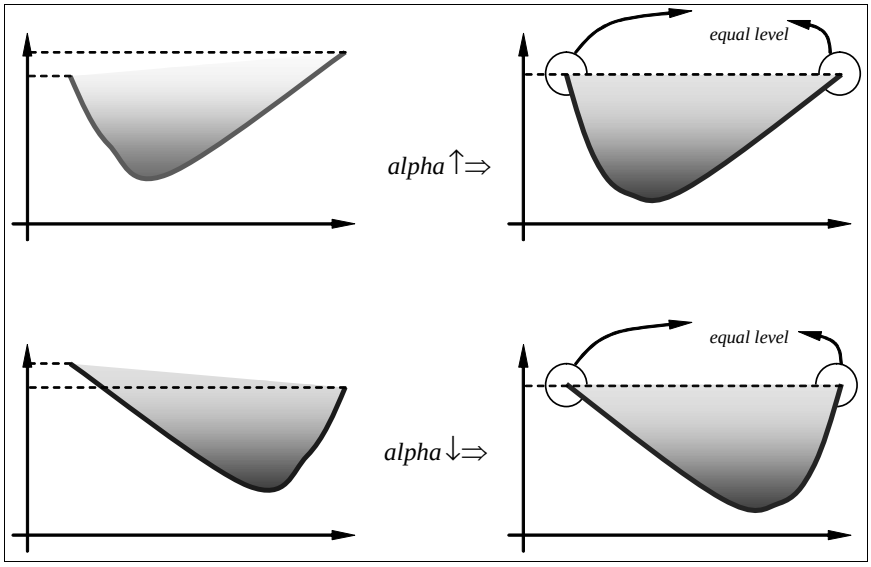


Fig. 10. Symmetrization of serialization components

Conclusions

1. In the existing multiprocessor hardware solutions and communication systems operating them, serialization is a feature consisting in collecting data directly before the realization of the process which uses them. It is often associated with the capability and procedure to create presence backup containing these data [5]. In our interpretation, the serialization would be an algorithm feature relating to the data accumulation in processes as well as to the cumulative procedure of data transfer to the selected place (processor and process) directly before their use.
2. As the algorithm research shows, as a rule, the influence of both SLA and SLI components is essentially diverse (more than 40%). It results from the fact that a greater diffusion of subprocesses is the reason for the intensification of diffusion procedures and data collection. Simultaneously, it is conducive to a decrease in the degree of data accumulation procedures used.
3. Serialization characterizes the susceptibility of an algorithm to parallelization or diffusion of its realization. Processing environment parameters indicate whether it is more effective - to diffuse or to accumulate data (i.e. whether

the realization of concrete algorithm in a given processing environment is effective).

4. Symmetrization of the serialization components (alpha level in Figure 9) allows one to deterministically specify the participation of the accumulation of SLA and the diffusion of SLI components in the given algorithm realization. If the diffusion influence is greater than the accumulation influence then alpha > 0.5, otherwise alpha < 0.5.

References

[1] Flynn M.J., Some computer organizations and their effectiveness, IEEE Trans. on Computers 1992, C21.

[2] Raghavan P., A statistical adversary for online algorithms, Discrete Mathematics and Theoretical Computer Science, Springer-Verlag 1991.

[3] Sait S.M., Youssef H., VLSI Design Automation: Theory and Practice, McGrow-Hill Book Co., Europe 1995.

[4] Georges-Schleuter M., Explicit parallelism of genetic algorithms through population structures, Problem Solving from Nature, Springer-Verlag, New York 1991.

[5] Valduriez P., Parallel Processing and Data Management, Chapman & Hall 1992.

[6] Aarts E.H., Bont F.M., Habers E.H.,Van Laarhoven P.J., Parallel implementation of the statistical cooling algorithm, Integration, the VLSI Journal, 1986.