AENSI Journals

Australian Journal of Basic and Applied Sciences

ISSN:1991-8178

Journal home page: www.ajbasweb.com

An Approach to Achieving Increased Fault-Tolerance and Availability of Multiprocessor-Based Computer Systems

[1]Jamil Al Azzeh, [2]Dmitriy B. Borzov, [3]Igor V. Zotov and [4]Dmitriy E. Skopin and [5]Dr. Mazin Al Hadidi

[1]*Computer Engineering, Al Balqa' Applied University, Amman, 11134, Jordan.*
[2]*Computer Engineering, South West State University, Kursk, 305040, Russia.*
[3]*Computer Engineering, South West State University, Kursk, 305040, Russia.*
[4]*Computer Engineering, Al Balqa' Applied University, Amman, 11134, Jordan.*
[5]*Computer Engineering, Al Balqa' Applied University, Al-Salt 19117, Jordan.*

ARTICLE INFO

Article history:
Received 25 January 2014
Received in revised form
8 April 2014
Accepted 20 April 2014
Available online 25 May 2014

Keywords:
Multiprocessor, high-availability system, fault-tolerance, deserialization, parallel process, allocation, assignment, PLD, simulation.

ABSTRACT

In the present paper, mesh-connected multiprocessor-based computer systems are considered, a new approach that makes it possible to increase fault-tolerance and availability of a system in the presence of faulty units/links is discussed.

To Cite This Article: Jamil Al Azzeh, Dmitriy B. Borzov, Igor V. Zotov and Dmitriy E. Skopin and Dr. Mazin Al Hadidi., An Approach to Achieving Increased Fault-Tolerance and Availability of Multiprocessor-Based Computer Systems. ***Aust. J. Basic & Appl. Sci.,*** *8(6): 512-522, 2014*

INTRODUCTION

Many present-day computer systems (CS) are constructed as highly parallel multiprocessor-based architectures capable of concurrent execution of complex applications. Efficient utilization of such systems requires the use of specific compilers for control program deserialization, separation and interprocessor allocation (Chen, C.L. and G.M. Chiu, 2001; Ho, C.T. and L. Stockmeyer, 2002; Sokolinskaya, I.M., L.B. Sokolinsky, 2009)[1-3]. For many critical CS applications, it is necessary to provide fault-tolerance and increased availability such that the system can immediately resume its operation as a local fault has been detected and isolated. Fault-tolerant CS operation requires a reconfiguration procedure to be performed as fast as possible which presupposes re-execution of some parallel compiler operations. The interprocessor code allocation is one of the aforementioned operations (Maltsev, I., 2009; Sobolev, S.I., 2008). The allocation operation is invoked to reassign code sections onto healthy processor units and build-up a modified message routing graph whose arcs are mapped onto non-faulty links only. As a result, the logical configuration of the system is kept unchanged while its physical organization becomes irregular because of the presence of faulty nodes/links (Keller, A., A. Reinfeld, 2001; Buyya, R., 1999). Bearing in mind stringent time constraints, the allocation operation is implemented with certain hardware-support which is considered as a part of the parallel compiler.

2. Searching for parallelism in sequential control programs:

Many existing programs are still sequential which requires some deserialization to be performed to efficiently use the system.

A sequential control program is represented as a set of $|Q|_i$ lines of code, $|Vi|_k$ input variables, and $|Vo|_k$ output variables, where $i=\overline{1,N}$, $k=\overline{1,M}$, N is the number of lines in the program (Anderson, G.A., L.D. Jensen, 1975; Wittie, L.D., 1981; Gochman, S., *et al*., 2003), *M* stands for the total number of variables. Factors $|ki|_M$

Corresponding Author: Jamil Al Azzeh, Computer Engineering, Al Balqa' Applied University, Amman, 11134, Jordan.
E-mail: jamil33jo@fet.edu.jo.

and $|ko|_M$ are introduced for each line Q_i to indicate the presence of input variables Vi and output variables Vo, such that $ki_p = \{0,1\}$, $ko_p = \{0,1\}$, $p = \overline{1,M}$. Then for each line Q_i the following equation takes place

$$ko_f \cdot Vo_k = ki_1 \cdot Vi_1 \Xi ki_2 \cdot Vi_2 \Xi \ldots \Xi ki_M \cdot Vi_M$$

where Ξ is the symbol of an operation to be performed, and *f* stands for the number of one-valued factors ko for *ith* line of code.

The set $KI = \{ki_{p1}, ki_{p2}, ..., ki_{pN}\}$ of factors $|ki|_M$ consisting of *N* subsets $ki_p = \{ki_1, ki_2, ..., ki_M\}$ is defined by the binary input variable matrix $I = \|I\|_{q \times p}$, where $|I|_q = ki_p$, $q = \overline{1,N}$. The cell at a row i and at a column *k* of the matrix is supposed to contain 1 if *kth* variable is a member of *i*th statement to the right of the equality operator.

To illustrate the above definitions the following linear fragment of a sequential program is considered:

```
b=a+e;
if (b>d){e=a+c;}
else {f=d+e; a=c+g;}
g=e-a;
e=a+b;
```

Table 1 contains the input variable matrix *I* corresponding to the code fragment constructed according to the above definition.

Table 1: Input variable matrix *I*.

Variable id Statement count	a	b	c	d	e	f	g
1	1	0	0	0	1	0	0
2	0	1	0	1	0	0	0
3	1	0	1	0	0	0	0
4	0	0	0	1	0	1	0
5	0	0	1	0	0	0	1
6	1	0	0	0	1	0	0
7	1	1	0	0	0	0	0

The set of statements of the program is given as $R = \{R_1, R_2, ..., R_N\}$, where *N* – the number of statements. We introduce the precedence matrix $Ms = \|Ms\|_{q \times q}$ to indicate the precedence relation between statements:

$$Ms_{ik} = \begin{cases} 1, \ if \ R_i \varphi R_k \\ 0, \ \text{otherwise}, \end{cases}$$

where $k = \overline{1,N}$ and symbol φ means that *k*th statement immediately follows *i*th statement.

Table 2 presents matrix *Ms* corresponding to the above code fragment. The cell at a row *i* and at a column *k* of the matrix is supposed to contain 1 if *k*th statement immediately follows *i*th statement.

Table 2: Precedence matrix *Ms*.

Statement count	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0
3	0	1	0	0	0	0	0
4	0	1	0	0	0	0	0
5	0	0	0	1	0	0	0
6	0	0	1	0	1	0	0
7	0	0	0	0	0	1	0

Based on the precedence matrix the statement reachability matrix *Md* can be defined:

$$Md_{ik} = \begin{cases} 1, \ if \ R_i \mapsto R_k \\ 0, \ \text{otherwise}, \end{cases}$$

where symbol $\mapsto$ means that statement R_k can be reached from statement R_i. If $R_i \mapsto R_k$, then there exists a subset of statements $R' \subseteq R$ such that

$$R_i \ \varphi \ R_1', R_1' \ \varphi \ R_2', ...,$$
$$R_{N'-1}' \ \varphi \ R_{N'}', R_{N'}' \ \varphi \ R_k.$$

Table 3 shows matrix *Md* corresponding to the above code fragment. The cell at a row *i* and at a column *k* of the matrix is supposed to contain 1 if *k*th statement can be reached from *i*th statement.

Table 3: Reachability matrix *Md.*

Statement count	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	0	0	0	1	0
3	0	1	0	0	0	0
4	0	0	1	0	0	0
5	0	0	0	1	0	0
6	0	1	0	0	0	0

Matrices *I*, *O*, and *Md* are considered as the basic data entities to perform statement parallelization. Matrix *Ms* is used for the analysis of the inner structure of the program.

In linear code sections of a program control transfer is done sequentially starting from the upper line. The precedence matrix of a linear section looks typically as that shown in Table 4.

Table 4: Precedence matrix of a linear code section.

Statement count	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0
3	0	1	0	0	0	0	0
4	0	0	1	0	0	0	0
5	0	0	0	1	0	0	0
6	0	0	0	0	1	0	0
7	0	0	0	0	0	1	0

If we represent the columns as set $K=\{k1,k2,...,kM\}$, and the rows we do so as set $I=\{i1,i2,...,iN\}$, then in the precedence matrix for a linear code section $Md_{ik}=1,\ if\ i+1=k$, otherwise $Md_{ik}=0$. There are three typical constructs in sequential programs which we consider below – the conditional branching, the while loop, and the do-while loop. The precedence matrix corresponding to a conditional branching is presented in Table 5.

Table 5: Precedence matrix for a conditional branching.

Statement count	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	0	0	0	0	0
3	0	1	0	0	0	0
4	0	0	1	0	0	0
5	0	1	0	0	0	0
6	0	0	0	1	1	0

A distinguishing feature of such a construct is that statement R_a transfers control to statement R_b, where $b-a>1$, which explains the existence of 1s below diagonal $i+1=k$. A while loop can be presented by the precedence matrix which can be seen in Table 6.

Table 6: Precedence matrix for a while loop.

Statement count	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	0	0	0	1	0
3	0	1	0	0	0	0
4	0	0	1	0	0	0
5	0	0	0	1	0	0
6	0	1	0	0	0	0

In Table 7 the precedence matrix for a do-while loop can be seen.

Table 7: Precedence matrix for a do-while loop.

Statement count	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0
3	0	1	0	0	0	1	0
4	0	0	1	0	0	0	0
5	0	0	0	1	0	0	0
6	0	0	0	0	1	0	0
7	0	0	0	0	0	1	0

As we can see from table 5 and table 6, the presence of 1s on the main diagonal in the precedence matrix above means the existence of a loop. This is the basis for the following procedure of loop search and classification.

1. Input $MS = \|\text{ms}\|_{N\times N}$;
2. Set $i = 1$;
3. Set $k = i$;
4. If $ms_{ik} = 1$, $z = i+1$; $s = k-1$; then go to step 5, otherwise go to step 6;
5. If $ms_{k+1,k} = 1$, then a do-while loop is found, otherwise a while loop is found. The statements $R_z, R_{z+1}, ..., R_s$ make the body of the loop.
6. Let $k = k+1$;
7. If $k > N$, then go to step 8, otherwise go back to step 4;
8. Let $i = i+1$;
9. If $i = N$, then terminate, otherwise go back to step 3.

The main problem when checking out the possibility of simultaneous statement execution is the presence of data dependencies between them. We propose that the set of statements constituting a linear code section *i* can be divided into 4 categories: read only (W_i); 2) write only (X_i); 3) first read then write (Y_i); 4) first write then read (Z_i). Variables' memory cells which are read out in a statement R_i should not be erased by the data of a statement R_k meaning that

$$\left(W_i \cup Y_i \cup Z_i\right) \cap \cap \left(X_k \cup Y_k \cup Z_k\right), = \varnothing, \quad (1)$$

If $|I|_k$ is a row of the input variable matrix and $|O|_k$ is the corresponding row of the output variable matrix, then the aforementioned requirements to the data independence between statements R_i and R_k can be stated in the following form:

$$I_j \cap O_i = \varnothing, I_i \cap O_j = \varnothing, \quad O_i \cap O_j = \varnothing. \quad (2)$$

Taking into consideration equation (2), the data independence condition for statements R_i and R_k can be checked out according to the following formula:

$$F(i,k) = \left(I_i \wedge O_k\right) \vee \left(I_k \wedge O_i\right) \vee \vee \left(O_i \wedge O_k\right). \quad (3)$$

If the resulting vector *F* is zero, then the statements can be executed in parallel as different variables are processed

3. Allocating code sections in a multiprocessor-based computer system:

The program to be allocated is considered as a set of interacting code sections (subroutines) which in turn is represented by a task interaction graph $G =< X, E >$, where *X* is the set of the vertices of graph *G*, each vertex $x_i \in X$ corresponds to a code section, and the set of arcs *E* models the links between code sections. Each arc $e_{ij} \in E$ is marked with the amount of data m_{ij} (in bytes) transferred during the exchange between the corresponding sections. Thus, graph *G* can be represented by the data exchange matrix (DEM) $M = \|m_{ij}\|$.

We shall represent the multiprocessor-based system by a topological model in the form of graph $H =< P, V >$, where

$P = \begin{Bmatrix} p_{1,1} & p_{1,2} & \dots & p_{1,n} \\ p_{2,1} & p_{2,2} & \dots & p_{2,n} \\ \dots & & & \\ p_{n,1} & p_{n,2} & \dots & p_{n,n} \end{Bmatrix}$ is the set of processor identifiers organized as matrix $|P|_{n\times n}$, whose capacity $|P| = N = n^2$ equals the number of processor units; *V* is the set of interunit links introduced as an $n^2 \times n^2$ adjacency matrix $\|W\|_{N\times N}$.

Taking into account the above definitions, we can represent the allocation of a program in the target system as

$$\beta_s = \{x_{k,q}\} \to \{p_{k,q}\}, \tag{4}$$

where $s = \overline{1, N!}$, $k = \overline{1,n}$, $q = \overline{1,n}$. Here *s* is the allocation count which corresponds to an *sth* variant of code interunit assignment. Note that the cardinality of set $\psi = \{\beta_S\}$ is the same as the number of various section rearrangements: $|\psi| = N!$.

We introduce the data exchange matrix $M = \| m_{ij} \|_{N\times N}$ to characterize graph G, where $N = n^2 = |X|$. Also we introduce the minimum distance matrix $D = \| d_{ij} \|_{N\times N}$ $N = n^2 = |H|$ based on the adjacency matrix $\|W\|_{N\times N}$ of graph *H*.

Let Ψ be the set of various assignments of form (4). In this case, the program allocation problem can be formulated as the search for an assignment $\beta^* \in \psi$ such that

$$T_{\beta^*} = \min_{\Psi}\{\max_{\beta_s \in \Psi}\{T_{\beta_s}(p_{a,b}, p_{x,y})\}\}, \tag{5}$$

where $T_{\beta_s}(p_{a,b}, p_{x,y})$ denotes the communication delay caused by the data transfer between processors $p_{a,b}$ and $p_{x,y}$ corresponding to assignment β_s, and it is calculated according to the following formula

$$T_{\beta_s}(p_{a,b}, p_{x,y}) = d_{ij} \cdot m_{ij}, \tag{6}$$

where $i = (a-1)\cdot n + b$, $j = (x-1)\cdot n + y$.

Because the cardinality of the set $\psi = \{\beta_s\}$ consisting of all possible assignments of form (4) is equal to the number of various allocations of program sections in the system, the search for the best assignment β^* according criterion (5) is an NP-hard problem. That is why we try to control the degree of communication delay reduction (6) to decide if it is necessary to continue the search process. The decision is made based on the comparison of the min-max value of the communication delay (5) to a hypothetical minimum.

The procedure of the search for hypothetically minimal communication delay can be stated in the following form.

1. Rearrange elements $d_{kl} \neq 0$ of matrix *D* on a single row $D' = \|d_{kl}^z\|$ so that $d_{k'l'}^{z_1} \le d_{k''l''}^{z_2} \Leftrightarrow z_1 > z_2$, where z_1 and z_2 are the consecutive numbers of the elements in D'.

2. Rearrange elements $m_{ij} \neq 0$ of matrix *M* on a single line $M' = \|m_{ij}^z\|$ so that $m_{i'j'}^{Z_1} \ge m_{i''j''}^{Z_2} \Leftrightarrow z_1 > z_2$, where z_1 and z_2 are the consecutive numbers of the elements in M'.

3. Calculate

$$T_{\inf} = \max\{m^z d^z\}, \tag{7}$$

where $z = \overline{1,|E|}$, m^z and d^z are elements having the same positions in the above vectors M' and D'.

Thus we shall search for such an allocation that minimizes the communication delay according to criterion (5) which is calculated in the same way as that expressed by formula (7):

$$T = \max\{m_{i,j} \cdot d_{ij}\} \tag{8}$$

where $i, j = \overline{1, N}$.

One of the possible ways to speed up the search for a better allocation is based on the purposeful rearrangement of rows and columns in DEM with the selection of a position to move element $m_{\alpha\beta}$ according to

the rule

$$d_{\alpha k} < d_{\alpha \beta}, \tag{9}$$

where $d_{\alpha k}, d_{\alpha \beta}$ are elements of matrix MDM; $m_{\alpha \beta}$ is the element of DEM to which the value $max\left\{m_{ij} \cdot d_{ij}\right\}$ found in a previous step of the rearrangement process corresponds.

The novelty of the above approach is that the average number of rearrangements can be decreased if unnecessary ones are rejected allowing the next row/col movement according to the additional criteria:

$$m_{\alpha k} \cdot d_{\alpha k} < m_{\alpha \beta} \cdot d_{\alpha \beta},$$

$$m_{\alpha k} < m_{\alpha \beta}. \tag{10}$$

On the basis of the given theoretical statements, the allocation procedure is formulated which consists of the following steps.

1. An arbitrary first allocation of the vertices of graph *G* is made. The allocation is performed by assigning the elements of DEM $M = \| m_{ij} \|_{N \times N}$ on the elements of MDM $D = \| d_{ij} \|_{N \times N}$. Then the maximum communication delay $T_{\text{н}}$ which corresponds to the given initial variant of program allocation is calculated according to formula (8).

2. To select an allocation by criterion (4) the value $T_{\inf}$ (see formula (7)) for graph *G* is calculated first. Then the following relation is evaluated characterizing how close is the current allocation to the theoretical optimum:

$$\eta_{\text{н}} = \frac{T_{\text{н}}}{T_{\inf}} \tag{11}$$

3. Starting with the element m_{ij} which corresponds to the attained $\max\{m_{i,j} \cdot d_{ij}\}$, we try to move its column to the place of another column so that after the rearrangement and the calculation of *T* according to formula (10) the following value η becomes less than $\eta_{\text{н}}$:

$$\eta = \frac{T}{T_{\inf}} \tag{12}$$

The target position to move the element m_{ij} is chosen by the following criterion:

$$d_{ik} < d_{ij} \tag{13}$$

where i, j are the indices of the element m_{ij} being moved; d_{ij} is the distance corresponding to the element m_{ij}; d_{ik} is a preferable distance to move the element m_{ij}; i, k are the indices of the element m_{ik} replaced with the element m_{ij}; *i* is the number of row in DEM and MDM within which the rearrangement of elements *j* and *k* occurs; *j*,*k* are the indices of rearranged columns (and rows) of DEM.

4. The following formula is applied to check out the degree of allocation improvement according to the criterion of communication delay reduction:

$$\sigma = \frac{\eta_{\text{н}}}{\eta} = \frac{T_{\text{н}}}{T} \tag{14}$$

To evaluate the developed program allocation method, dedicated software tools have been developed which allow to simulate the operation of the allocation algorithm, to calculate $\eta = \frac{T}{T_{\inf}}$ at every test rearrangement, and to fix the calculation time with a given precision.

The aim of the simulation was to determine the advantage value σ with respect to $T_{\text{н}}$ produced using the proposed method under different types and degrees of filling DEM corresponding to different code section interconnection density.

Resulting from the simulation, the σ vs $\eta_{\text{н}}$ (see Fig.1) and the η vs *Q* (see Fig.2) dependencies were obtained and analyzed.

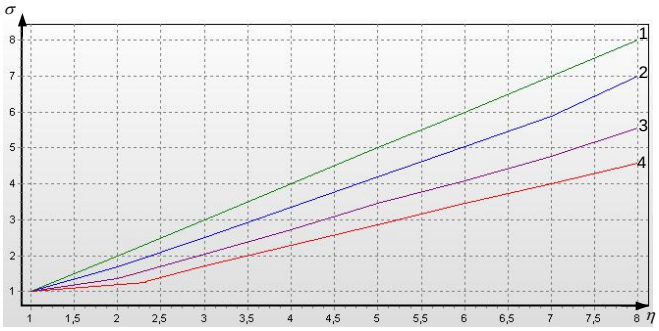

Fig. 1: σ vs $\eta_{\text{н}}$ simulation-based dependencies.

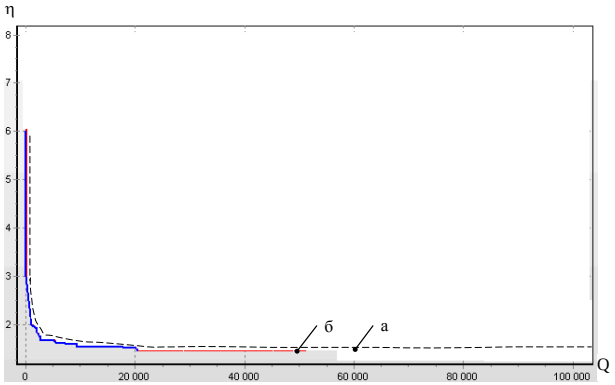

Fig. 2: Simulation-based dependencies of η from the number of permutations Q: a) according to criterion (9); b) according to criteria (10).

In Fig. 1, the following conventions are adopted: chart 1 corresponds to the case $\frac{m_{\min}}{m_{\max}} = \frac{1}{10}$; chart 2 corresponds to the case $\frac{m_{\min}}{m_{\max}} = \frac{1}{5}$; chart 3 corresponds to the case $\frac{m_{\min}}{m_{\max}} = \frac{1}{4}$; chart 4 corresponds to the case $\frac{m_{\min}}{m_{\max}} = \frac{1}{3}$.

On the basis of the given charts (figures 1, 2) we can presume that:

1) if $\eta_{\text{н}} \le 2$, the communication delay is reduced $\sigma = 1{,}3\ldots2$ times only;
2) because it is necessary to perform many row/col movements, the majority of which are fulfilled during the last part of the search procedure, when η stops to decrease significantly (fig.1), it makes sense to introduce a threshold $\eta_{\text{н}}$ on the allocation effectiveness which would cut the redundant permutations according to condition $\eta \le \eta_{\text{н}}$, where $\eta_{\text{н}}$ can be chosen out of the range $1.8 \le \eta_{\text{н}} \le 2$.

Note that $\eta_{\text{н}} = 2$ is an acceptable threshold for decision-making. In this case the allocation search time reduces significantly and provides $1.5\ldots4$ times delay minimization.

4. Implementing the proposed method on a PLD:

A PLD-based implementation helps us simulate the proposed method and prove its increased efficiency. The program to be executed is again presented by task interaction graph $G = < X, E >$, where the vertices $x_{qk} \in X$ correspond to code sections (branches), while the arcs $e_{ij} \in E$ correspond to the data transfers between them and are formally arranged into the adjacency matrix $M = \|m_{ij}\|_{N \times |E|}$ where $N = |X|$. The PLD topology is set by the following matrix form H:

$$H = \begin{Bmatrix} p_{1,1} & \cdots & p_{\varpi,\theta} & \cdots & p_{1,n} \\ p_{2,1} & \cdots & p_{\varpi,\theta} & \cdots & p_{2,n} \\ \cdots & & & & \\ p_{m,1} & \cdots & p_{\varpi,\theta} & \cdots & p_{m,n} \end{Bmatrix} \quad (15)$$

where $P_{\varpi,\theta}$ are separate units of the PLD ($\varpi=\overline{1,m}$, $\theta=\overline{1.n}$). It is supposed that

$$P_{\varpi,\theta}=F(O,X) \tag{16}$$

where $O=o_1,o_2,\ldots,o_\xi$ is the set of PLD input terminals, and $X=x_1,x_2,\ldots,x_\xi$ is the set of PLD output terminals.

The input terminals of some PLD units get connected to the output terminals of the other units (Sean Lee, H.H., 2003; IA–32, 2006; IA–32, 2006; Kanter, D., 2006; Tendler, *et al*, 2002). The total amount of signals which are transmitted in a PLD from one unit to another determines the overall communication delay which must be minimized to increase the PLD performance. A simplified representation of a PLD can be seen in Fig. 3.

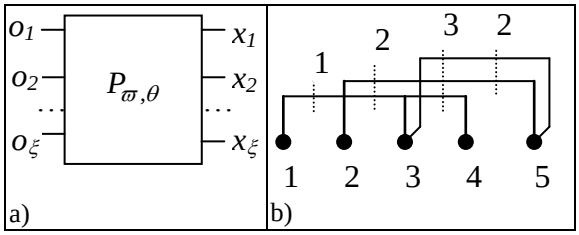

Fig. 3: a) a PLD VLSI external presentation, b) - interconnections inside a PLD.

As we can see in Fig. 3(a), the value ξ (the number of inputs and/or outputs), also used in (16), depends on the given scheme realization of the PLD and is not known in advance.

Let us consider interconnections in a PLD VLSI as a route from one pin o_i or x_j ($i=\overline{1,\xi}$, $j=\overline{1,\xi}$) to another. In Fig. 3(b), there is an example of interconnection pattern where the black points are the outputs of PLD units. The numbers next to the dotted line denote the interconnection capacity between the pairs of adjacent pins.

To formally define the allocation of a program in a PLD VLSI, we introduce the matrix of chains (MC) which is a rectangular matrix $V=\left|v_{i,j}\right|_{n,\alpha}$ where $i=\overline{1,N}$, $j=\overline{1,\alpha}$, $n=|X|$, α are the total amount of interconnections received as a result of program section execution in the PLD. The example of MC representing the pattern of Fig. 3(b) is given in Fig. 4.

	1	2	3
1	1	0	0
2	0	1	0
3	1	0	1
4	1	0	0
5	0	1	1

Fig. 4: Matrix of chains created according to Fig.3(b).

The rows of the matrix represent the outputs of the PLD units and in the columns define the interconnections corresponding the given placement variant.

The cardinality of set $|P|$ depends on the number of interconnections received as a result of placing units in a PLD. The parameter α is the number of interconnections and is not known in advance. The following requirement must be taken into account:

$$\alpha\rightarrow\min \tag{17}$$

Thus the program placement in a PLD VLSI can be stated as

$$\beta_s=\left\{x_{s_{q,k}}\right\}\rightarrow\left\{p_{q,k}\right\} \tag{18}$$

where $S=\overline{1,N!}$. In (18), the symbol → means a one-to-one mapping of $x_{S_{q,k}}\in X$ to $p_{q,k}\in H$. Here *s* is the consecutive number of the next rearrangement corresponding to the *s*th variant of placement. The cardinality of set $\psi=\{\beta_s\}$ containing various mappings (18) is equal to the number of various rearrangements of program sections $x_{qk}\in X$ in matrix *M*: $|\psi|=N!$.

Let Ψ be the set of all possible mappings of form (18). Then the allocation problem can be formulated as the search for such a mapping $\beta^* \in \psi$ that

$$T_{\beta^*} = \min_{\Psi}\{\max_{\beta_s \in \Psi}\{T_{\beta_s}(|V|_{p_{q,k}})\}\}, \tag{19}$$

where $T_{\beta_s}(|V|_{p_{q,k}})$ stands for the delay of data transmission within unit $p_{q,k}$ for the mapping β_s. The term $\max_{\beta_s \in \Psi}$ in the above formula (19) means the search for maximum delay in $p_{q,k}$ where $q = \overline{1,n}$, $k = \overline{1,\alpha}$; the term $\min_{\Psi}$ corresponds to the search for the minimum value of the delay for the maximum $\max_{\beta_s \in \Psi}\left\{T_{\beta_s}\left(|V|_{p_{q,k}}\right)\right\}$.

We shall define an adjacent pin (AP) as such an arrangement of outlets of a PLD VLSI if the following condition holds:

$$\theta = \left|v_{i,j} - v_{i,j-1}\right| = 1 \tag{20}$$

While searching for an allocation in a PLD the following condition must be done:

$$\sum \theta_{\Delta} \to \max \tag{21}$$

where $\Delta = \overline{1,n}$.

The search for the best allocation β^* by criterion (19) is an NP-hard problem. One of the solutions is to apply purposeful rearrangements of rows and columns of MC so that formulae (17) and (20) are satisfied.

One way to speed up the search is to lower the target communication delay value corresponding to the best results. To fulfill this, it is necessary that during the search process the decreasing degree of the communication delay (19) is under control to identify when we can finish purposeful continuation of matrix operations.

The suggested procedure is based on the calculation of the communication delay theoretical minimum $T_{\inf}$ which can be taken by the assumption that the topology of graph *G* and *H* are the same. Below a procedure for calculating $T_{\inf}$ is formulated.

1. Put elements $m_{kl} \neq 0$ ($k = \overline{1,N}$, $\overline{l = 1,N}$) of matrix $M = \|m_{ij}\|_{N \times |E|}$ to a vector $M' = \|m_{kl}^z\|$ where *z* is the index of an element in M', $k = \overline{1,N}$, $l = \overline{1,N}$.

2. Calculate

$$T_{\inf} = \sum_{i=1}^{j=|E|} m_i^z \tag{22}$$

where $i = \overline{1,|E|}$, $z = \overline{1,|E|}$, $|E|$ is the cardinality of *E*; m^z are the elements to vector M'.

So we can formulate a procedure for program allocation consisting of the following stages:

1. An arbitrary preliminary placement of graph *G* is produced. The placement is realized by mapping the MA matrix onto the MC matrix, and the delay $T_{\text{н}}$ is found corresponding to the given placement.

2. To compare different variants of placement according to criterion (19) the search of decreased value of $T_{\text{н}}$ is calculated at the beginning by the threshold calculation algorithm. Then the closeness of $T_{\text{н}}$ corresponding to the first variant of placement is calculated:

$$\eta_{\text{н}} = \frac{T_{\text{н}}}{T_{\inf}} \tag{23}$$

3. The matrix row movement is realized so that after the rearrangement and calculation of *T* according to (22) the following relation

$$\eta = \frac{T}{T_{\inf}} \tag{24}$$

is reducing compared to $\eta_{\text{н}}$ (23) and the values of η of the previous variants of placement.

4. The degree of the communication delay minimization is then calculated to compare the last allocation to the theoretically best one according to the following formula:

$$\sigma = \frac{\eta_{\text{н}}}{\eta} = \frac{T_{\text{н}}}{T} \tag{25}$$

To investigate the suggested method, dedicated simulation software tools were developed which allow to analyze the influences of the proposed criteria on the allocation quality and the time required for the generation of the PLD topology. The aim of research was to determine the value σ obtained using the developed method with different MC values corresponding to different connectivity levels of code sections. The initial and the attained deviation of delay *T* from T_{inf}, the number of permutations to perform, and the time spent on the search were the main results of the simulation.

In Fig. 5, simulation-based dependencies of σ from η_{H} and of η from *Q* are shown.

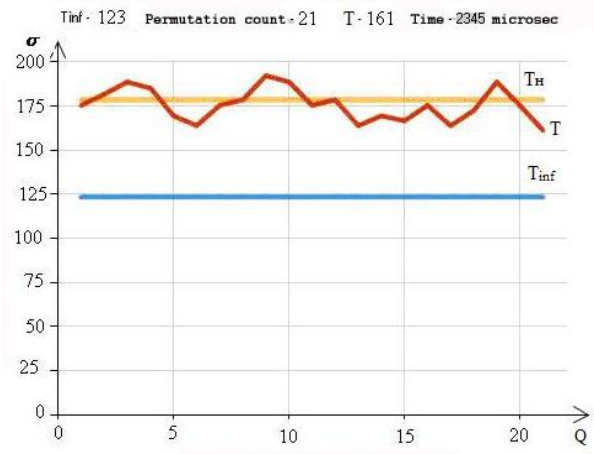

Fig. 5: The dependencies of σ from η_{H} and η from *Q*.

As we can see from the figure, as a result we managed to lower the value of the communication delay from 175 to 161, having moved it closer to the theoretical minimum T_{inf}, which is equal to 123.
In Table 1, the results of simulation for different MC patterns are shown.

Table : The required number of permutations *Q* at different sizes of the matrix of chains.

V	5	9	10	20	25	50	60
ψ	5!	9!	10!	20!	25!	50!	60!
Q	5	16	21	51	66	141	171

In the rows of the table, the data for exhaustive search for all possible rearrangements ψ and the required number of permutations *Q* using the developed method are shown. From the analysis of Table 1, we can come into conclusion that for a small number of inner units in a PLD and a small number of pins (5-10), the number of permutations required to search for a better allocation variant is not that high, but as they increase the difference between Q and ψ increases significantly.

5. Conclusion:

In the paper, a method for the fault-tolerant configuration of multiprocessor-based computer systems is presented. Because fault-tolerant CS operation requires a reconfiguration procedure to be performed in a limited time, re-execution of some parallel compiler operations such as code section interprocessor allocation is proposed. The allocation is made with dedicated hardware support to perform fault-tolerant CS reconfiguration as fast as possible to meet real-time requirements. The allocation operation reassigns program branches onto healthy processors and rebuilds the message routing graph to use non-faulty units and links only.

REFERENCES

Anderson, G.A., L.D. Jensen, 1975. Computer interconnection structures, taxonomy, characteristics and examples // Computing Surveys of AC., 7(4): 197-213.

Aztec. A Massively Parallel Iterative Solver Library for Solving Sparse Linear Systems. – http://www.cs.sandia.gov/CRF/aztec1.html

Buyya, R., 1999. High Performance Cluster Computing: Systems and Architectures. Volume 1, Prentice Hall PTR, NJ.

Chen, C.L. and G.M. Chiu, 2001. "A Fault-Tolerant Routing Scheme for Meshes with Nonconvex Faults," IEEE Trans. Parallel and Distributed Systems, 12: 467-475.

Genetic Algorithm Based Parallel Jobs Scheduling / S. Shapovalov, G. Tarasov // In Proceedings of First Russia and Pacific Conference on Computer Technology and Applications (RPC 2010). - Vladivostok: IACP FEB RAS. - 2010. - P. 211-216.

Gochman, S., *et al*., 2003. The Intel Pentium M Processor: Microarchitecture and Performance. Intel Technology Journal, 7(2).

Ho, C.T. and L. Stockmeyer, 2002. "A New Approach to Fault-Tolerant Wormhole Routing for Mesh-onnected Parallel Computers," Technical Report RJ 10265, IBM Almaden Research Center, San Jose, Calif., Nov.

IA–32, 2006. Intel Architecture Optimization Reference Manual. Intel.

IA–32, 2006. Intel Architecture Software Developer's Manual. Intel.

Ian Foster. Designing and Building Parallel Programs.–http://www.hensa.ac.uk/parallel/books/ addison-wesley/dbpp http://rsusu1.rnd.runnet.ru/ ncube/design/dbpp/book-info.html

Kanter, D., 2006. AMD's K8L and 4x4 Preview. Real World Technologies.

Keller, A., A. Reinfeld, 2001. Anatomy of a Resource Management System for HPC Clusters. Preprint. To appear in: Annual Review of Scalable Computing, 3.

Maltsev, I., 2009. Atomistic and mesoscale simulations of free solidification in comparison.// Ilya Maltsev *et al*., 2009, Modelling Simul. Mater. Sci. Eng., 17, 055006 (10pp), doi: 10.1088/0965-0393/17/5/055006.

MPI: The Complete Reference. –http://rsusu1.rnd.runnet.ru/ncube/mpi/mpibook/mpi-book.html

Sean Lee, H.H., 2003. P6 & NetBurst Microarchitecture. School of ECE, Georgia Institute of Technology.

Sobolev, S.I., 2008. New generation architecture of the X-Com metacomputing system // Distributed computing and GRID-technologies in science and education. Book of Abstracts of the Third International Conference. Dubna, June 30 - July 4. P. 111.

Sokolinskaya, I.M., L.B. Sokolinsky, 2009. Hybrid method for solving incomplete linear optimization problem on distributed memory multiprocessor system // Pro-ceedings of the 2009 International Conference on High Performance Computing, Networking and Communication Systems (HPCNCS-09), July 13-16 2009, Orlando, FL, USA. –ISRST 2009. P. 18-20.

Tendler, *et al*, 2002. J. POWER4 system microarchitecture. IBM Journal of Research and Development, 46(1).

The Cost Effective Computing Array (COCOA). – http://cocoa.aero.psu.com

Wittie, L.D., 1981. Communication structures for large networks of microcomputers // IEEE Transactions on Computers, 30(4): 264-273.