Article

Diagnosis of intermittently faulty units at system level

Viktor Mashkov[1], Jiří Fišer [2], Volodymyr Lytvynenko[3*] Maria Voronenko[4*]

[1] Department Informatics, Scientific Faculty, University of J.E.Purkyne, Usti nad Labem, Czech Republic
viktor.mashkov@ujep.cz
Department Informatics, Scientific Faculty, University of J.E.Purkyne, Usti nad Labem, Czech Republic
jf@jf.cz
[3] Department Informatics and Computer Science Kherson National Technical University,Kherson, Ukraine
immun56@gmail.com
Department Informatics and Computer Science Kherson National Technical University, Kherson, Ukraine
mary_voronenko@i.ua
* Correspondence: immun56@gmail.com; Tel.: +380509955202

Abstract: System level diagnosis is an abstraction of high level and, thus, its practical implementation to particular cases of complex systems is the task which requires additional investigations, both theoretical and modeling. Mostly, diagnosis at system level intends to identify only permanently faulty units. In the paper, we consider the case when both permanently and intermittently faulty units can occur in the system. Identification of intermittently faulty units has some specifics which we have considered in this paper. We also suggest the method which allows distinguishing among different types of intermittent faults. Diagnosis procedure was suggested for each type of intermittent faults.

Keywords: complex systems; self-diagnosis; intermittent fault; diagnosis procedure

1. Summary

Recent advances in the semiconductor technology have made it possible to design powerful single-chip microprocessors. With the considerable reduction in the cost of microprocessors, it is now feasible to build increasingly sophisticated microprocessor systems with up to thousands of microprocessors.

New opportunities have emerged for developing many-core processors with novel architectures that support better communication among cores and allow for better algorithm parallelization, resulting in a dramatic increase in processor performance [1].

The rapid growth of many-core processors has brought about an increasing demand for high processor reliability and availability.

Implementing many cores on a single die is possible due to shrinking of processing elements. Modern nano-scale technologies make it possible to integrate billions of transistors on a single die. As die size and transistor density grow, the susceptibility of these processors to hardware faults grows as well. Permanent and

 doi:10.20944/preprints201811.0299.v1

intermittent hardware faults, caused by defects in silicon or metallization and wear out over time, lead to circuit reliability problems. Due to these circuit reliability problems, dependability becomes one of the major challenges for all future nano-scale technologies[1],[2],[16]. That is why fault-tolerance is becoming an essential property that must be integrated from the very beginning in every chip design.

To provide processor fault-tolerance different methods can be applied. All of them have some mutual features. First of all, error detection should be performed. Usually, error detection results in a signal or message which indicates about erroneous state of processor. After that, a recovery of processor state is performed (error handling). Error handling can be carried out either by way of backward error recovery or forward error recovery or masking. Masking requires different types of redundancy. Hardware redundancy implies redundant processor cores. Information redundancy exploits different coding techniques. Time redundancy consists in repeating some computation on the same hardware.

After error handling, fault handling can be performed. It is worth noting that, fault handling can be omitted. It depends on the chosen way of error handling. Main steps of fault handling are as follows: fault diagnosis; fault isolation; processor reconfiguration; and reinitialization.

For providing fault handling built-in testing capabilities (so-called built-in-self-test schemes) are often used. Built-in testing allows performing fault handling efficiently [4],[18]. Unfortunately the wide applicability of built-in testing is undermined by the need to have some part of the processor (called the hard-core) operational even in the presence of fault.

Self-diagnosis, which uses the ability of processor cores to test each other, is now actively studied as a promising technique for providing processor cores checking and diagnosis [7], [8], [12], [13], [14].

We assume that the suggested in the paper approach to diagnose intermittent faults can be used not only for many-core processors but also for wide range of complex systems, which employ self-diagnosis.

This paper is organized as follows. Section 2 describes the problem of classification of intermittent faults in the context of self-diagnosis. Section 3 is devoted to problems with developing diagnosis algorithms when intermittent faults are allowed. Conclusions and future work are given in the final section.

2. Classification of intermittent faults in the context of self-diagnosis

Intermittent fault of a system unit can be defined as a fault, which randomly transfers from a latent state to an active state and vice versa. There exist several models, which describe the behavior of an intermittent fault [6], [19]. We use the model proposed in [19]. In this model, the behavior of intermittent fault is expressed by continuous Markov chain, where the time during which an intermittent fault stays in active, respectively passive state, is random value with exponential distribution (see Fig. 1).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 November 2018 doi:10.20944/preprints201811.0299.v1

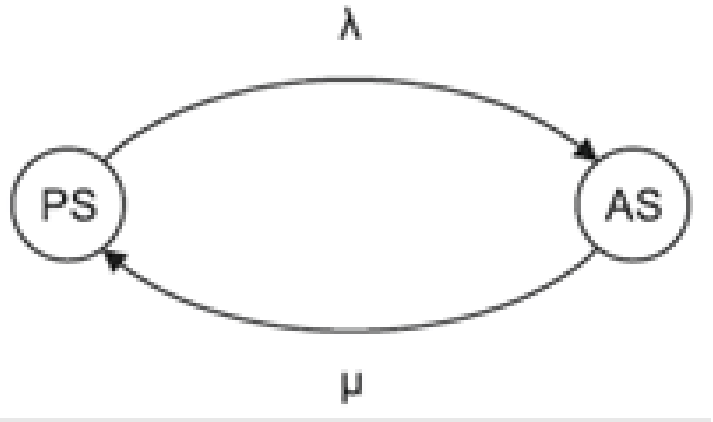

Figure 1. Model of intermittent fault

In Fig. 1, λ and μ denote the rates of transition from passive to active state and vice versa.

When an intermittent fault is in active state it can course an error in a system unit and affect the tests related to the erroneous unit.

System level self-diagnosis deals with the mutual testing. In this case, one of the system units performs tests on the other units. Convenient form for representing a mutual testing is graph model (testing graph). Example of testing graph is shown in Fig.2.

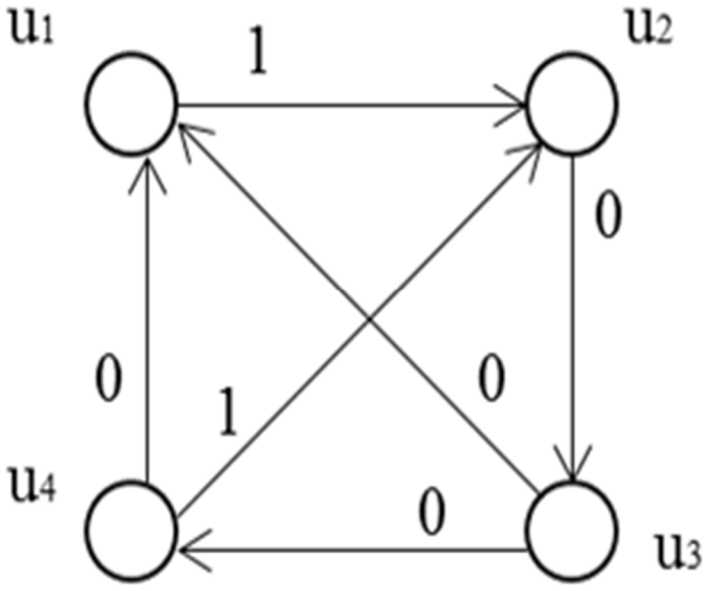

Figure 2. Example of testing graph

In the given case, diagnosable system consists of four units $U=\{ u_1, u_2, u_3, u_4 \}$. Each unit u_i, $u_i \in U$, is assigned a particular subset of units in U to test.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 November 2018 doi:10.20944/preprints201811.0299.v1

Generally, a unit can test itself. Although, we assume that a unit doesn't do this. In the corresponding testing graph, it means the absence of loops (i.e., testing graph doesn't contain the edges that connect a vertex to itself).

A test, which invokes two units, is represented in the testing graph by the edge between two vertices which correspond to testing and tested units. In Fig. 2, there are six edges. These edges correspond to tests τ_{12}, τ_{23}, τ_{34}, τ_{41}, τ_{31} and τ_{42}. Subscripts point to the units that are involved in the test. The complete collection of tests is called a testing assignment. Test result is represented by binary variable r_{ij} which can take values 0 or 1. Variable r_{ij} is equal to 0 if unit u_i evaluates unit u_j as fault-free. Otherwise, $r_{ij}=1$.

In Fig. 2, test results are shown next to the corresponding edges. The set of test results is called a syndrome. Identification of faulty units using a syndrome is called diagnosis.

Generally, for providing diagnosis at system level some assumptions are made, such as:

- tests can be performed only in the periods of time when system units do not perform their proper system functions (i.e., when they are in idle state). That is, a system unit is not tested continuously, and, therefore, there exists a probability of not detecting the failed unit;
- even if a unit is failed, a test not always detects this event. It depends on test coverage;
- result of a test is expressed as 0 or 1 depending on the evaluation of the testing unit about the state of the tested unit;
- tests in a system can be performed either according to a predefined testing assignment or randomly.

In this paper, we consider that test coverage is equal to 100%. We also assume that tests among system units are performed according to predefined testing assignment. It means that the total time of testing is known beforehand. Consequently, the periods of time when each unit is involved in tests are also known in advance.

In the given case, it is possible to consider parameters of intermittent fault model (i.e., λ and μ) in relation to the total time of testing, $t_{testing}$. Table 1 presents possible evaluations of values $1/\lambda$ and $1/\mu$ as compared to the value of $t_{testing}$. This comparison bears some resemblance to the techniques based on fuzzy logic. We evaluate the values of $1/\lambda$ and $1/\mu$ as "large" and "small" depending on the ratios of values $1/\lambda$ ($1/\mu$) and $t_{testing}$.

Table 1. Various types of intermittent faults

Ratio $t_{testing}/A$	**$A=1/\lambda$**	**$A=1/\mu$**
case 1 (class 1)	large	large
case 2 (class 2)	large	small
case 3 (class 3)	small	large
case 4 (class 4)	small	small

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 November 2018 doi:10.20944/preprints201811.0299.v1

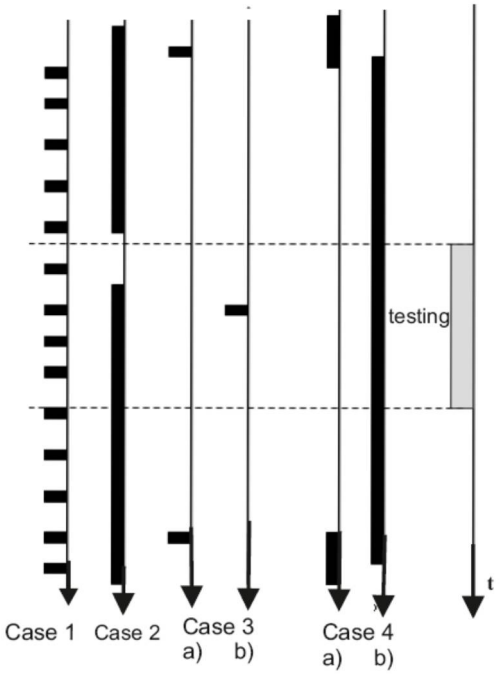

Figure 3. Different behaviors of intermittent faults

As a result of this consideration, it is possible to divide the considered intermittent faults into several classes.

Intermittent faults related to class 1 and class 2 can be detected with high probability during testing procedure. Detection of intermittent faults in case 3 is very improbable (problematic). Probability of detecting such faults is low. As concerns case 4, there exist two options – a) and b) (see Fig. 3). In the given case, probability of intermittent fault detection can be estimated as 0.5 when $1/\lambda \approx 1/\mu$.

Generally, system level self-diagnosis is aimed at detecting permanently faulty system units. Nevertheless, there exist the possibility to detect also intermittently faulty units when intermittent faults are related to classes 1,2 and 4.

3.Problems with developing diagnosis algorithms when intermittent faults are allowed

Diagnosis is performed on the basis of obtained syndrome. A syndrome is a set of test results. The result of test τ_{ij} is denoted as r_{ij} and can take the values 0 or 1 depending on the fact of how unit u_i evaluates the state of unit u_j.

In the paper, we accept the evaluation proposed by Preparata [17].

$$r_{ij} = \begin{cases} 0 & \text{if units } u_i \text{ and } u_j \text{ are fault-free} \\ 1 & \text{if units } u_i \text{ is fault-free and and } u_j \text{ is faulty} \\ X(0,1). & \text{when unit } u_i \text{ is faulty} \end{cases}$$

We also assume that if an intermittent fault is in active state, then unit with this fault behaves as permanently faulty unit.

To explain the problems with diagnosis made on the basis of obtained syndrome, let us consider a simple example with code in Julia programming language. We have implemented our solution in this programming language for

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 November 2018 doi:10.20944/preprints201811.0299.v1

illustration of how bitwise operations can effectively represent the operations on syndrome.

Let system consists of five units and tests are performed according to a predefined schedule. This system is represented by graph shown in Fig. 4.

Obtained syndrome is

$$\mathrm{Rd} = \{r_{12}=0, r_{13}=0, r_{23}=1, r_{24}=0, r_{34}=1, r_{35}=0, r_{45}=1, r_{41}=0, r_{51}=0, r_{52}=0\}$$

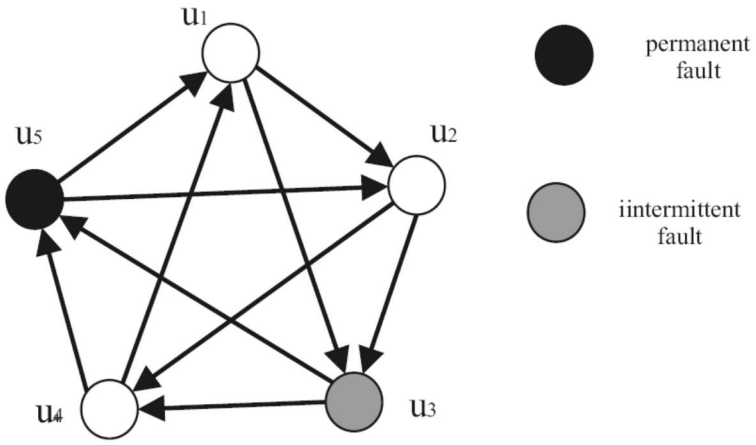

Figure 4. Testing graph for the considered system

There exist several methods for diagnosis of permanently faulty units [20], [21]. Most of them make the assumption about maximum possible number of faulty units in the system. In [17], there was proved that correct system diagnosis is possible if the total number of faulty units do not exceed the value t, where

$$t = \left\lfloor \frac{N-1}{2} \right\rfloor \tag{2}$$

It is easy to verify that no system state (i.e., no combination of permanently faulty and fault-free units in which the total number of permanently faulty units doesn't exceed the value t) can lead to obtaining such syndrome Rd.

For example, if units u_3 and u_5 are permanently faulty, result r_{13} should be equal to 1, whereas in syndrome Rd this result is equal to 0.

Thus, direct application of algorithms developed for diagnosis only permanently faulty units is not possible.

Such situation can be explained by specific behavior of some faulty units. Particularly, a unit can be intermittently faulty. It means that at one moment it behaves like a permanently faulty unit, whereas at other moments like a fault-free unit.

Thus, new methods for diagnosis intermittently faulty units at system level should be developed.

Testing graph of system can be represented by square adjacency matrix of dimension $N \times N$, where N is the total number of units in the system. Elements of this matrix are Boolean values. (If unit u_i performs test on unit u_j, the item on i-th row and j-th column takes the value true, otherwise false). Matrix of boolean values can be represented in packed form. In the given case, matrix items are represented by single bits and matrix of Boolean values is represented by bit arrays.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 November 2018 doi:10.20944/preprints201811.0299.v1

Julia language supports bit arrays with arbitrary dimension by using standard library of data structures.. Therefore the definition of new data type for data of adjacency matrix is straightforward.

```
struct TestingGraph # testing graph
                    #is structure
tests::BitArray{2} # encapsulating
                    #two-dimensional
                    #bit array
end
```

Julia supports the definition of function which is performed on a new data type:

```
# constructor of new testing graph
# n = number of units
# test = specification of tests in
# form of vector of tuples
function TestingGraph(n::Int, tests::Vector{Tuple{Int, Int}})
    tg = TestingGraph(falses(n,n))
    for (checking, checked) in tests
      tg.tests[checking, checked]=true
    end
    return tg
end
#auxiliary function(returns number of units)
size(tg::TestingGraph)=size(tg.tests,1)
```

Usually a syndrome is depicted in testing graph as weights of edges. In the given case, weights are results of tests. A result can take values either 0 or 1. In Fig. 5, example of syndrome for the system with five units is shown.

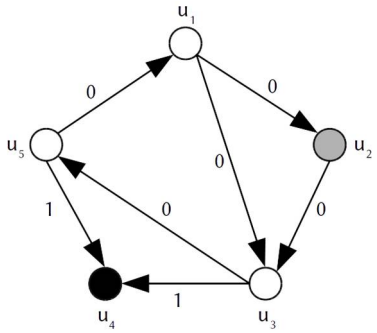

Figure 5. Instance of syndrome for the testing graph

This type of weighted graphs can be represented by a pair of Boolean matrices (in packed bit array representation). The first matrix (denoted as *mask*) is identical to adjacency matrix of testing graph it defines the positions of valid results (i.e., results equal to 0). The second matrix codes results of tests (*true* represents result equal to 0, *false* represents result equal to 1). Only results with true bit in the mask matrix are valid.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 November 2018 doi:10.20944/preprints201811.0299.v1

Definition of syndrome representation in Julia language with basic constructor is shown below. In the given case, results are specified as tuples of indexes of testing and tested units and numerical representation of test result.

```
    struct Syndrome
        mask:: BitArray{2}
        results::BitArray{2}
    end
    function Syndrome(n::Int, results:: Vector{Tuple{Tuple{Int, Int},
Int}})
    # both matrices are inicialized by falses
    s = Syndrome(falses(n,n), falses(n,n))
    s.results[checking, checked] = (result == 0) # set bit in results
s.mask[checking,checked]=true #and mask
    end
        return s
    end
```

For example the syndrome depicted in Figure 5 can be created by this call of the constructor:

```
R_d = Syndrome(5, [((1,2), 0), ((1,3),0), ((2,3), 0),((3,4), 1), ((3,5),
0), ((5,1), 0), ((5,4), 1)])
```

If we assume that in the considered example units u_2 and u_4 are permanently faulty, then the set of all potential syndromes will be obtained. This set R contains two syndromes and can be defined as follows

$$R = \{u_{12} = 0, u_{13} = 0, u_{23} = X, r_{34} = 1, u_{35} = 0, u_{51} = 0, u_{54} = 1\}, \text{ where } X \in \{0,1\}$$

Checking if syndrome is a member off set R for the system with faulty units in Julia is simple and effective. Such checking is based on the standard bitwise operations AND, XOR and OR. The set of syndromes is expressed by standard syndrome structure with slightly alternative meaning of bit mask. In the given case, true bit in mask is interpreted as unambiguous result of test. False bit is interpreted as ambiguous result of test (in set R, it is denoted as "X") or non-existent test.

```
    # syndrome membership in syndrome set #(s is element of s_set)
    function ∈(s::Syndrome, s_set::Syndrome)
    if s_set.mask .& s.mask != s_set.mask
         throw(ArgumentError("Incompatible syndrome"))
        end
        cover=(s_set.results. s.results).& s_set.mask
        return !reduce(|, cover)
    end
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 November 2018 doi:10.20944/preprints201811.0299.v1

The main part of checking if syndrome is a member of set R exploits the map-reduce approach. Here, XOR is applied between bit array of results of obtained syndrome and bit array of expected results of syndrome. Operation is performed only on valid bits of set R (see mask operation by AND). Reduce step is performed by bitwise OR (any true bit leads to denial of membership).

```
    # creation of syndrome set for given test # graph `dg`
    # and boolean array of units states #(false = faulty units)
possible_syndrome_set(tg::TestingGraph,
unit_states::BitArray{1})
        = Syndrome(tg.tests.&unit_states,   tg.tests .&
unit_states')
```

The mask array of set R is constructed by vector application of bitwise AND between adjacency matrix of testing graph and column vector of unit's states. That is, operation AND is applied on rows (because validity of testing is determined by testing unit. The result array of set R is outcome of application of transposed vector of unit's states and adjacency matrix (operation AND is applied on columns because results of testing are determined by tested unit).

Checking if syndrome R_d is a member of R for system with two faulty units is obvious

```
    R_d ∈ possible_syndrome_set(tg, BitArray([true, false,
true, false, true])
```

As can be seen, the syndrome R_d does not belong to the set R since any syndrome that belongs to R has r_{12} equal to 1, whereas in the syndrome R_d this result is equal to 0.

It can be explained by the fact that unit u_2 has an intermittent fault which transfers from AS (PS) to PS (AS) during testing procedure. In view of this, most of the algorithms developed for diagnosis of permanently faulty units cannot be directly used for diagnosis intermittently faulty units.

Nevertheless, there was suggested the method [10] based on summary (updated) syndrome, R_Σ. Summary syndrome R_Σ is obtained after performing m rounds of test routine (i.e., m repetition of tests). Summary syndrome R_Σ is computed as follows:

$$R_\Sigma = \{r_{ij}\}, \quad r_{ij} = \underset{l}{r^l_{ij}}, \tag{3}$$

where $r^l_{ij} \in R_l$, R_l -syndrome obtained in l -th round of repetition of test routine.

The summary syndrome is computable by reduction of syndrome list by binary operation of syndrome union. This operation is implemented by bitwise AND between result bit array of syndrome (result 1 is represented as false!).

```
    Function ⊆ (s1::Syndrome, s2::Syndrome)
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 November 2018 doi:10.20944/preprints201811.0299.v1


```
        if s1.mask != s2.mask
            throw(ArgumentError("Incompatible syndromes"))
        end
        return Syndrome(s1.mask, s1.results & s2.results)
    end
    R_all = reduce(∈, list_of_syndromes)
```

When summary syndrome R_{Σ} is a subset of set R_0 (i.e., $R_{\Sigma} \in R_0$), the algorithms developed for diagnosing permanently faulty units can be also used for considered faulty situation. R_0 is a set of syndromes that can be obtained when only permanently faulty units can take place (considering that real number of faulty units do not exceed the maximum number of faulty units which is allowable for diagnosable system). The set R_0 is constructible by mapping of function possible_syndrome_set over iterator all possible permutation of states of units in diagnosable system (bit arrays of all permutations of unit's states are obtained from binary numbers between 0 and 2^n).

```
    function iter_possible_states(tg::TestingGraph)
     n = size(tg)
     t = (n-1)÷ 2  # integer divison
     return (set for set in
            (BitArray(digits(i, base=2, pad=n)) for i=2^n-1:-1:0)
                if n-count(set) <= t) # only with maximal t faulty
end
```

In the given case, the task arises to determine the number of test routine repetitions, l. Neither is determined what should be done if after l repetitions the condition $R_{\Sigma} \in R_0$ is not true.

To solve these tasks, we suggest the following decision. It is suggested to repeat the test routine several times, l. Concrete number of repetitions of test routine depends on the total number of units in the system N, on the classes of intermittent faults, which are going to be detected, and on the required credibility of diagnosis. If an intermittent fault belongs to class 4, the value of l does not influence the test results. If an intermittent fault belongs to class 3, a unit with such fault behaves either as fault-free or as faulty only during one test. Any next test will show that this unit behaves as fault-free. Thus, two test are sufficient to form r_{ij} which make condition $R_{\Sigma} \in R_0$ true. It also concerns an intermittent fault of class l. In case 2, a unit with such intermittent fault with high probability will behave as permanently faulty. There is low probability that one of the tests will show that this unit is fault-free. Although, any other test will show that this unit is faulty. Thus, two tests are enough to form R, which satisfies the above mentioned condition.

If after several rounds of tests repetitions the condition $R_{\Sigma} \in R_0$ is not true, then it is needed to determine a consistent set of units, K_u. Set K_u contains all of the units

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 November 2018 doi:10.20944/preprints201811.0299.v1

that, according to the summary syndrome, are diagnosed as fault-free. Units that belong to the set K_u evaluate each other as fault-free. In order to determine the set K_u , it is needed to remove from summary syndrome R_Σ all test results which are equal to 1. Remaining results allow to form a Z-graph.

Z-graph is formed as follows. If r_{ij} in R_Σ is equal to 0, then there is an edge between vertices v_i and v_j in Z-graph directed from v_i to v_j .

Credibility of diagnosis result will be greater when Z-graph is connected [15]. In Fig. 6, examples for connected and disconnected Z-graphs are shown.

a) Z-graph is disconnected b) Z-graph is connected

Figure 6 . Examples of Z-graph

Connection of Z-graph depends considerably on testing assignment and on the allowed number of faulty units (i.e., on the maximum number of faulty units that still allows to obtain the correct result of diagnosis).

We consider the case when tests among system units are performed in accordance with a pre-set schedule (i.e., defined a priori). Having the testing graph, it is possible to investigate if Z-graph is connected under the condition that maximum number of faulty units doesn't exceed value *t*.

Since each edge of graph involves two vertices, the minimal number of edges which still allows to provide system t-diagnosability [7], L_{min}, is equal to

$$L_{\min} = \left\lceil \frac{N(t+1)}{2} \right\rceil \tag{4}$$

Next, we can examine if the value $L_{\min}$ is sufficient in order that Z-graph be connected.

Given $L_{\min}$, we can determine the minimal number of edges in Z-graph, $K_{\min}$, that ensures connection among its vertices. $K_{\min}$ can be determined as follows

$$K_{\min} = \left\lceil \frac{(t+1)}{2} \right\rceil \tag{5}$$

Z-graph is connected if

$$K_{min} > \frac{t(t-1)}{2} \; or \; 2t+1 > t^2 \tag{6}$$

This inequality is true for t < 3 (that is, when total number of system units is less than seven).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 November 2018 doi:10.20944/preprints201811.0299.v1

In case when N ≥ 7, value L_{min} is not sufficient in order that Z-graph be connected. In the given case, we can determine the number of additional edges that must be added to Z-graph in order that this graph becomes connected.

Using results presented in [15], it is possible to conclude that Z-graph maximally may have

$$\pi = \left\lfloor \frac{(N-t)}{2} \right\rfloor \quad (7)$$

components each of which consists either of two vertices for $N = 3+4a, a = 0,1,2,\ldots$, or of two vertices except the one consisting of three vertices for $N = 5+4a, a = 0,1,2,\ldots$

In order to connect these π components (π-1) additional edges are necessary. The choice of the pair of units for performing a test that corresponds to the additional edge in Z-graph can be carried out according to existing algorithms [15].

For Z-graph it is possible to form the matrix M_R. Matrix M_R is square matrix presentation of the subset of R_Σ which has only zero results. If result r_{ij} is an element of the resulting subset of R_Σ, then element m_{ij} of matrix M_R has value of 0. Otherwise, element m_{ij} is denoted as dash.

There could be used the diagnosis algorithm presented in [9]. This algorithm is based on the matrix which is similar to matrix M_R and can identify all faulty units (on the condition that total number of faulty units does not exceed the value t). Handling matrices like M_R is also presented in [3], [5].

In the given case, it is needed to calculate the total number of 0 in each column. Then, obtained numbers $S_i, i = 1\ldots, N$, should be compared with value t. If $S_i \geq t$, then unit $u_i \frac{1}{n}$ is diagnosed as fault-free. If condition $S_i \geq t$ is not true for all $i \in \{1, \ldots, N\}$ then it is needed to find in Z-graph a simple directed cycle of length $t+1$. Such cycle can be determined from matrix M_R. All units, which are in this cycle, should be identified as fault-free.

Units that are not identified as fault-free are either permanently faulty or have an intermittent fault. It should be noted that there is low probability of incorrect diagnosis. This probability can be evaluated in relation to different total number of system units, different classes of intermittent faults and different number of test routine repetitions.

4. Conclusions

Mostly, system level self-diagnosis deals with the diagnosis of permanently faulty units. Although, there are many real situations when intermittently faulty units can occur in the system. In the given case it is necessary to take into account various behaviors of such faulty units.

Behavior of intermittent fault can be modeled by continuous Markov chain with two states – Passive (PS) and Active (AS). If an intermittent fault is in PS, unit acts as fault-free. If an intermittent fault is in AS, unit acts as fault (i.e., as if it has a permanent fault).

During system level self-diagnosis both permanent and intermittent faults can occur. Each test evaluates the state of a unit either as fault-free or as faulty. In the latter case, it does not discriminate the permanent and intermittent faults.

Diagnosis algorithms that deal with the sets of test results can consider (take into account) the testing time and, thus, potentially they could discriminate the permanent and intermittent faults. For this, the testing procedure must be very long. In reality, it is needed to perform the testing procedure in acceptable time for each concrete system. This leads to the situation when it is very difficult to discriminate between permanent and intermittent faults. In view of this, suggested diagnosis algorithm can identify only fault-free system units. All units that are not diagnosed as fault-free should be considered as faulty without further specification.

Suggested diagnosis is developed on the basis of consistent sets of system units. Units that belong to consistent set with high probability can be considered as fault-free. When situation allows to extend the testing time, it is possible to gain higher probability that units of a consistent set are fault-free.

Author Contributions: Viktor Mashkov wrote the theoretical part of the paper Jiří Fišer developed the algorithms in Julia language Volodymyr Lytvynenko contributed to theoretical part and developing the codes Maria Voronenko prepared and formed the manuscript

References

1. A. Avizienis, J.C. Laprie, B. Randell.Fundamental concepts of dependability. Technical Report Series University of Newcastle upon Tyne, Computing Science 1145 (010028), 2001, pp. 7-12.
2. A. Avizienis, J.C. Laprie, B. Randell. Dependability and its threats. A Taxonomy. IFIP 18th World Computer Congress "Building the Information Society", France, 2004, pp.91-120.
3. S. Babichev, V. Lytvynenko, M. Korobchynskyi, M. A. Taif. Objective clustering inductive technology of gene expression sequences features. *Communications in Computer and Information Science*, 2017, pp.359-372.
4. G. Hetherington, et. al. Logic BIST for large industrial design: real issues and case studies. *ITC*, 1999.
5. S. Babichev, M. A. Taif, V. Lytvynenko, V. Osypenko. Criterial analysis of gene-expression sequences to create the objective clustering inductive technology, *Proc. of IEEE 37th International Conference on Electronics and Nanotechnology, ELNANO*, 2017, pp. 244-248.
6. Mashkov supervised of the project
7. S. Kamal, C. V. Page. Intermittent fault: a model and a detection procedure. *IEEE Trans. Comput.* Vol.C-23, No.7, 1974. pp.713-719.
8. L.E. Laforge, K. Huang, V.K. Agarwal. Almost sure diagnosis of almost every good elements.*IEEE Trans. on Computers,* Vol.43, No.3, 1994, pp. 295-305.
9. [8] P. Maestrini, P. Santi. Self-diagnosis of processor arrays using a comparison model. *In Proc. Of 14th IEEE Symposium on Reliable Distributed Systems,* 1995, pp. 218-228.
10. V. Mashkov. Selected problems of system level self-diagnosis. *Lviv: Ukrainian Academic Pres*s, 2011, 184 pages.
11. S. Mallela, G. Masson, Diagnosable systems for intermittent faults. *IEEE Trans. Comput.*, Vol.C-27, No.6, 1978. pp.560-566.
12. [11] V. Mashkov. Task Allocation among Agents of Restricted Alliance. *Proc. of IASTED ISC'2005 conference,* pp. 13-18, Cambridge, MA, USA, 2005.
13. V. Mashkov, J. Barilla, P. Simr. Applying Petri Nets to Modeling of Many-Core Processor Self-Testing when Tests are Performed Randomly, *Springer, Journal of Testing*, 2013, DOI: 10.1007/s10836-012-5346-8.
14. V.A. Mashkov, O. V. Barabash Self-testing of multimodule systems based on optimal check-connection structures // *Engineering Simulation*, 1996, Vol.13, pp. 479-492

15. V.A. Mashkov, O. V. Barabash Self-checking of modular systems under random performance of elementary checks // *Engineering Simulation*, 1995, Vol. 12, pp. 433-445
16. V. Mashkov, J. Pokorny. Scheme for comparing results of diverse software versions. In Proc. of ICSOFT Conference. Barcelona, Spai, 2007, pp. 341-344.
17. J.C. Laprie, editor. Dpendability: Basic Concepts and Terminology. Springer-Werlag, 1992.
18. T. Preparata, G. Metze, R. Chien. On the connection assignment problem of diagnosable system. *IEEE Trans. on Electronic Computers*, Vol. EC-16, No. 12, 1967, pp. 848-854.
19. M. Psarakis, D. Gizopoulos, E. Sanchez, M. Sonza Reorda. Microprocessor Software-Based Self-Testing. *IEEE Design & Test of Computers*, Vol.27, No.3, 2010, pp. 4-19.
20. S. Su, I. Koren,.K. Malaiya. A continuous-parameter Markov model and detection procedures for intermittent faults. *IEEE Trans. Comput*. Vol.C-27, No.6, 1978. pp.567-570
21. G. Sullivan. An O ($t^3 + |E|$) fault identification algorithm for diagnosable systems. *IEEE Trans. Comput*., Vol. C-37, 1988, pp. 388-397.
22. V. Vedeshenkov. On organization of self-diagnosable digital systems. *Automation and Computer Engineering*, Vol. 7, 1983, pp. 133-137.
23. H. Fujiwara, K. Kinoshita. Some existence theorems for probabilistically diagnosable systems. *IEEE Trans. on Comp*., Vol. C-27, No. 4, 1981, pp. 297-303.