Improving Diagnostic Accuracy in Exploratory Testing of MIMD Systems

A. Derezińska, J. Sosnowski
Warsaw University of Technology, Institute of Computer Science
Nowowiejska str. 15/19, 00-665 Warsaw, Poland. E-mail: [adr, jss]@ii.pw.edu.pl

1. Introduction

We present an original approach to deriving diagnostic decisions for multiprocessor systems with MIMD architecture. As compared with other diagnostic approaches [1-3,7,8] we admit complex (multivalued) test results and additionally analyse test result propagation. Developing reasoning rules, we take into account a large class of situations which can appear in the real system. These rules assure more accurate and more detailed diagnostics (fault localisation, specification etc.).

Considered MIMD system consists of a network of processing nodes coupled to a host computer. The architecture corresponds to any system in which nodes have direct access to limited number of their neighbouring nodes (matrix, n-cube, etc.). A node can perform data processing in parallel to simultaneous transmission in all links (external channels). We assume finite bounds on the processing and communication delays of fault-free components.

Each system node performs autodiagnostic procedures and generates its test result in the form of a complex signature comprising node identification and detailed test results (of the node and its interconnections, e.g. CPU, RAM, communication channels). The host collects the test results and performs the diagnostic reasoning. These results have to be delivered to the host computer via the system network. Hence, local faults in nodes and communication channels may disturb signature generation and retransmission. In consequence some signatures may not reach the host or be corrupted. We concentrate on an exploratory algorithm of collecting test results [6]. This approach is competitive to the broadcast algorithm (flooding), which we analysed in our previous publication [4].

After basic information about exploratory algorithms (sec.2.) consecutive steps of test results analysis will be presented (sec.3). In the terms of state patterns from the proposed state space, the possible consequences of selected faults will be examined (sec.4). In the conclusions, we give some remarks about different levels of test results analysis.

2. Exploratory algorithms for MIMD systems

The general idea of exploratory algorithms is similar to the distributed spanning the tree during the depth-first traversal of the system graph (serial exploration), or parallel, breadth-first traversal (parallel exploration) [9]. The basic algorithm specification can be enhanced taking into account testing of communication links performed while spanning the tree.

In the exploratory algorithms, the host initiates reading signatures from the system nodes by propagating a special request message and collecting response messages. The collection tree is created dynamically, while each node appoints its *parent* and *children* nodes (offsprings). A node waits for a request from its neighbours on its entire links. It confirms one received request with a positive answer. If the testing sequence associated with these messages was transmitted correctly, the node that sent the request becomes a parent of the considered node. The node that has the appointed parent sends requests to its remaining neighbours, sequentially or in parallel. Positively accepted neighbours become its children. A neighbour who is after exploration neither a parent nor a child will be specified as a *strange* node. Having established relations with all its neighbours, the node appends their states to its own signature, which is sent to its parent. A node retransmits signatures obtained from the children to the parent. In this way, test results propagate along the paths in the created trees.

The transmission protocol is quite complex because it deals with various faulty conditions and dynamic propagation of messages [5]. Developed specification follows special rules in order to avoid deadlock during co-operation between neighbouring nodes in fault free situation or in some fault cases. Correct system behaviour was described be a set of invariant properties based on reachability space analysis. The main faulty conditions invalidating particular invariant property were also identified.

3. Analysis of test results

3.1. Diagnostic scheme

Analysis of test results is performed in the host using the set of accumulated signatures. Each signature of a node contains the following data:

- identifier of the node,
- states of its neighbouring nodes appointed by the exploratory algorithm (parent, child or strange),
- states of its interconnections tested during exploration,
- test results of the node modules (e.g. CPU, RAM) generated by the testing procedures,
- CRC code.

In the developed approach to the system diagnostics, the paths of collecting signatures in the system are reconstructed. Relations between nodes are recognised due to availability of signatures, test results of nodes and links and consistency of appointed states of neighbouring nodes. Any faults from a large class may occur in nodes or interconnections and disturb system operation. Gathered diagnostic data are reconsidered according to selected hypothesis about fault occurrences and system behaviour during testing and collecting of signatures, defined by given specification and invariant properties. Many possibilities of incorrect signatures collected in the host are taken into account: signature with internally inconsistent or incompatible relations to theirs neighbours, replicated signatures (received from one or several nodes) or indicating other deviations from the tree structure, signatures with ambiguous test results etc. The analysis routine incorporates the structural restrictions of the system and gives the classification of the potential conditions of particular system elements.

The analysis of the collected test results is performed in three steps:

1) system nodes and interconnections are classified according to received signatures,
2) signature transmission graphs are identified and various attributes are defined for the nodes and interconnections,
3) final specification of diagnostic results is performed.

3.2. Signature classification

The collected test results are preprocessed and stored within appropriate data structures comprising node and interconnection state descriptors. For the correct system and for majority of faulty situations only one signature per node is received in the host node. Then, for each node its type is specified, as follows:

- *missing signature* - no signature of the node was received within specified time-out,
- *structurally incorrect* - syntax format of the signature is incorrect but with recognisable node identifier,
- *structurally correct* – correct syntax format, where two cases are possible:
  - *good* - signature with correct test results of the node was received,
  - *not-good* – signature with faulty test results of the node was received.

Only a structurally correct signature comprises all data listed in sec. 3.1. The host can receive a corrupted signature, in which we try to read the part responsible for node identification. If it is possible, a signature is classified as structurally incorrect, but it does not present test results of the node. Moreover, we can not be sure that to a structurally incorrect signature its node was correctly assigned.

According to the appointed relations of a node, a structurally correct signature can be *consistent* if exactly one parent node is determined and all other neighbours are either child or strange nodes. Otherwise, the signature is *inconsistent*.

In the case of receiving many structurally correct signatures of a node (not frequent situations) preliminarily, a dominant type is defined (tab.1). Each received signature falls into one of four categories – enumerated as columns in tab. 1. All possible combinations of a set of received signatures are taken into account. In the rows disjoint groups of combinations are specified. For different groups of combinations, five dominant types are assigned. Symbol 0 in table entries denotes that no signature defined in this column was received. Symbol 1 describes all situations, when one or more signatures of given type were received. In the last case, for a not-good signature we add symbol = or ≠ if faulty test results of the node in all signatures of the related type are the same (or only one signature of that type was received) or different, respectively.

Symbol x in remaining fields defines all combinations when signatures of given type were or were not received (equivalent to 0 or 1).

Received signatures → / Dominant type ↓	good		not-good	
	consistent	inconsistent	consistent	inconsistent
good	1 1 0	x x 1	0 1, ≠ 0	x x x
good-ambiguous	1	x	1, =	x
not-good	0 0	x 0	1, = 0	x 1, =
not-good – ambiguous	0 0	0 0	1, ≠ 0	x 1, ≠
ambiguous	0	1	1, ≠	x

Table 1. Dominant types for structurally correct signatures

Three dominant types are ambiguous. For *good-ambiguous* type we received different consistent signatures: good as well as not-good with the unique test results. If received signatures are only not-good but with different test results, the type is *not-good-ambiguous*. The last ambiguous state is very low probable. In this case the host received at least three signatures with the same node identifier: a good but inconsistent and two consistent but with different faulty test results. After structural transmission analysis (step 2.), a signature classification will be reviewed and a dominant type can be updated in some ambiguous cases (sec. 3.3).

3.3. Reconstruction of signature collecting graphs

The diagnostic value of the collected test results depends on the status of nodes and interconnections participating in the retransmission of signatures. Hence, in the second step of our approach we analyse the paths used in the transmission of signatures. The developed diagnostic algorithm attempts to reconstruct the tree-like structure of signature collecting. States of relations stored in signatures of neighbouring nodes are pairwise examined checking their compatibility. Two neighbouring nodes u_i and u_j are (i,j)-*compatible* with respect to the collecting of their signatures via node u_x connected to the host node iff:

- structurally correct signatures of nodes u_i and u_j were obtained from the node u_x,
- signature of node u_i points to the child u_j,
- signature of node u_j points to the parent u_i,
- test result of interconnection i-j is correct.

It can be assumed that a pair of compatible nodes creates a part of the collecting tree. For different fault possibilities 4 types of communication structures for signature collecting are considered:

- valid x-tree,
- probable x-tree,
- probable x-graph,
- deduced x-graph,

were u_x is one of nodes directly connected to the host node.
Based on the set of signatures collected in the host and on the results from the first step, the possible trees or graphs are created.

In most situations, a *valid x-tree* including uniquely defined nodes and connections can be reconstructed for each root node u_x. A valid x-tree comprises only nodes with consistent signatures received via the node u_x and connections corresponding to pairs of compatible nodes with respect to u_x. For example, a host received one structurally correct signature from each node in a system containing 9 nodes: u_1 .. u_9. All signatures are consistent, except node u_5 with an inconsistent signature. After analysis of the signatures in the host, two valid trees were constructed: 1-tree and 7-tree (fig. 1). Black arrows show a direction of signature flow along the connections assigned to a tree. For each node, information about its neighbouring nodes is graphically represented. A black circle attached to a node indicates to an appointed parent and an

open arrow denotes an appointed child. This information was read from the received signatures. In inconsistent signature of node u_5 (crossed square) information about two appointed parents was included. This node does not belong to any valid tree.

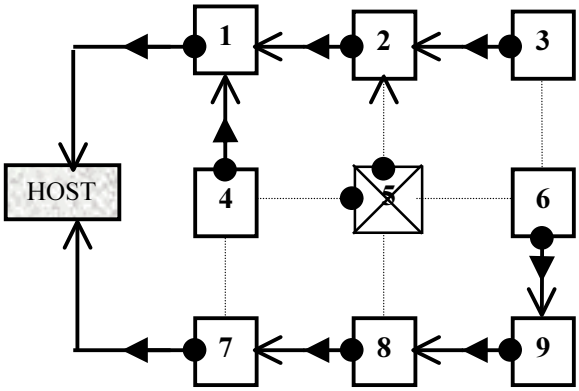

Figure 1. Valid trees example

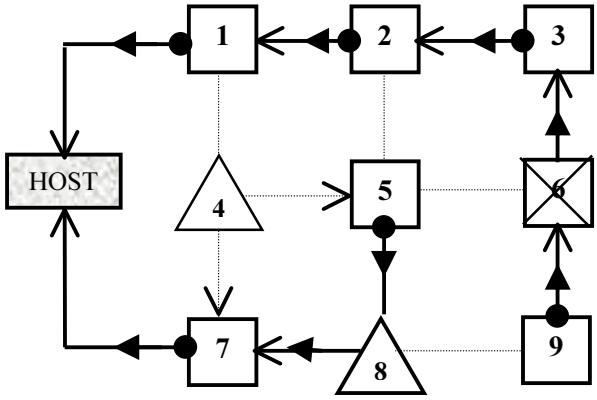

Figure 2. Probable trees example

If not all nodes were assigned to any valid tree, a *probable x-tree* is created. It comprises all nodes and connections that can be unambiguously assigned to it. To a probable x-tree we can append a node with inconsistent, structurally incorrect or missing signature if its unique parent node with a consistent signature belongs to the x-tree. In the system of fig. 1. node u_5 and connection between nodes u_5, u_2 will be attached to probably 1-tree, due to (2,5)-compatibility of these nodes. Further, nodes with consistent signatures can join a probable x-tree, when they are children of nodes with such incomplete signatures. Consequently, the probable x-tree can be recursively expanded linking alternately nodes with complete and incomplete signatures. Such probable trees are presented in fig.2. In this system the host received a signature for each node, but signatures of nodes u_4 and u_8 are structurally inconsistent (illustrated by triangle shape). Node u_8 was assigned to probable 7-tree based on information included in the signatures of its neighbouring nodes. Finally, node u_5 was added to 7-tree because node u_8, pointed as its parent, belongs to 7-tree. Similarly, a node u_6 with inconsistent signature and node u_9 were assigned to probable 1-tree.

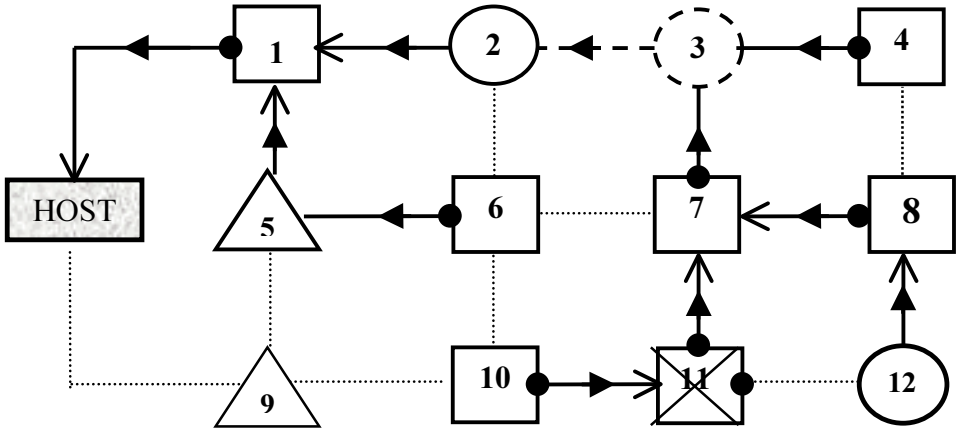

Figure 3. Probable and deduced graph example

If a tree structure can not be reconstructed due to faults, a *probable x-graph* (pseudo-tree) is created. Each node with unique, consistent signature, which does not belong to any structure, is assigned to the graph. Starting from these nodes, the sub-trees are expanded under the same rules as probability trees. For example, in the system illustrated in fig.3. one probable 1-tree is created (nodes u_1, u_2, u_5, u_6), where a circular node denotes a missing signature. Then, two nodes u_4 and u_7 are assigned to probably 1-graph. Next, a sub-tree rooted in node u_7 can be reconstructed. The final probably 1-graph is marked by a bold, solid line. However, for missing or conflicting signatures we can obtain structures, which are separated from the host node. In this case, a *deduced x-graph* is used for investigation of probable data flow and relations between nodes.

Trying to join the separate subgraphs, we add nodes and connections which might have been collaborated during creating tree process and retransmitting signatures. In the considered system (fig.3.) node u_3 and connection u_3-u_2 belong to deduced 1-graph (dashed lines). Node u_9 with structurally incorrect signature was classified neither as a parent nor as a child by its neighbouring nodes (host, u_{10}, u_5). This node does not belong to any defined structure for signature collecting.

For some rarely, malicious faults the same node can be connected to a deduced x-graph pointing to more than one potential parent (the tree structure will be not preserved). A node can also belong to two deduced graphs of different root nodes.

Creating state descriptors for nodes and interconnections, we define some additional attributes for each of them. These attributes are derived from the local relations and the signature collection process within identified structure types (comprising considered nodes or interconnections). Some of the introduced attributes define the probability of a correct usage of specified nodes or interconnections during signature collecting process. These attributes can confirm the correctness of a node or an interconnection, or allow deducing their correct behaviour if no direct test results are available. For example, node u_8 with structurally incorrect signature (fig.2.) belongs to a probably 7-tree. We assume that it correctly retransmits signatures, because of consistent signature of node u_5 delivered by node u_7. In result, an attribute *correct retransmission* will be assigned to node u_8.

Some nodes and interconnections can be also given special attributes in the case of a faulty behaviour during retransmission of other signatures: *loosing, replicating* or *corrupting* them. Definition of such attributes may depend on assumed hypothesis about system resistance on retransmission faults. It is estimated, who is probably more responsible for obtaining an erroneous signature: a node generating it or intermediary nodes. The above mentioned attributes can be assigned to nodes along the path of reconstructed tree starting from the node with a missing signature or with a questionable one. In this case, we obtain a set of suspicious nodes, which might include a faulty node. Components of these set can be examined in order to reduce its cardinality. For this purpose algorithms have been developed, differing in complexity and the assured accuracy of the system diagnostics. If there are more sets of nodes (paths) with the same attribute in a tree, we can look for common nodes. Especially a root node of a sub-tree formed by these paths can be chosen as a most probably faulty node. An attribute can be then assigned only to this node and not set for other nodes belonging to considered paths.

If attributes indicating faults during retransmission are associated with good signatures, then we can assume existence of intermittent faults or permanent faults which were not covered by the test procedures. Similarly checking whether test results stored in not-good or ambiguous signatures agree with the derived attributes we can confirm additionally the identified faults or assume their intermittent nature (inconsistent results). If some attributes confirm correct usage of a node or interconnection and other attributes indicate possible faults, then most probably this results from an intermittent fault.

Other attributes relate to nodes with received signature of any type (e.g. *replicated* signatures) or only to nodes with structurally correct signatures, as *consistency* feature and *compatibility* with neighbours. A special case refers to multiple consistent signatures of the same node but with different appointed states of neighbours, especially with different parent. This can be a result of a malicious fault, but it has very low probability. In this case, analysis of possible graphs for signature collecting can be performed either using algorithms that are more complex or algorithms for bounded set of nodes.

Some attributes are considered only when no structurally correct signature of a node was obtained (no direct test results of the node are available). *Correct co-operation* during tree construction process (even between strange nodes) can be established from the signatures received from the neighbouring nodes (if available). If we can not check consistency of a node signature, a consistency of states referring to the desired node can be determined from signatures of neighbouring nodes. Neighbourhood structure is *inconsistent* when there are more than one neighbour pointing to the node as to its child or all neighbours points to it as to their parents. For example, in figure 2. node u_4 with structurally incorrect signature has inconsistent neighbourhood structure (two potential parents), but node u_8 has consistent neighbourhood.

At the end of the second step, a final signature classification is performed if any signature has one of three ambiguous dominant types (tab. 1.). Taking into account obtained collection graphs and consistency with the neighbours, the most probable signature will be chosen and the dominant type will become unambiguous. In some rare cases, for different test results, we can not select appropriate test results and dominant type remains ambiguous.

3.4. Final specification

Depending upon the assumed model of faults and test responses the defined space of possible diagnostic states can be quite large. The diagnostic states are primarily defined by the signature consistency and the obtained test results. Additional characteristics are based on the reconstructed communication structure and coincidence of defined attributes. Some of these states relate to precise diagnosis, they are generated in case of simple faults, others specify most probable and ambiguous situations. Within this space some typical states are distinguished:

- correct node - has a consistent, good signature, belongs to a valid tree,
- node with detected faults, mostly permanent – has an unambiguous, consistent, not-good signature, belongs to a valid tree,
- dead node – has no signature, has an attribute about no co-operation with neighbours, is pointed by all neighbours as a strange node, no graph was identified.

The exhaustive state space is defined by all possible combinations of above mentioned specifications of a node. However, many of these elementary states are low probable and it is neither worthwhile nor efficient to characterise all of them independently. We can merge groups of elementary states with common, most distinctive features into more general states. During this merging we take into account only possible elementary state, because some node characteristics are correlated or mutually excluding.

A proposal of distinguishing such merged states is given in tables 2 and 3. Columns denoted from a to z correspond to these derived, disjoint states. States in tables 2, 3 cover the entire state space under the following assumption. No multiple consistent signatures of the same node with differently appointed neighbourhood structure were received in the host. This very low probable case is not covered for better readability. Each state is defined in a column by indicating selected features (graph types and attributes) specified in the rows. Field with sign 1 denotes, that at least one of considered features was associated with the node assigned to this state. Sign x shows, that the related attribute may be set or not (both possibilities are considered). Empty fields correspond to not assigned features.

Signature types →			Structurally correct											
			Good							Not-good				
										unambiguous			ambig.	
States →			a	b	c	d	e	f	g	a'- e'	h	i	j	k
Graph type	Valid tree		1	1						1	1			
	Probable tree or graph				1		1	1						
	Deduced graph					1							1	
	Unidentified graph								1			1		1
Attributes	Consistent signature		1	1	1	1	1			1			x	x
	Correct retransmission		x	x	x	x	x	x		...	x		x	
	Incompatible			1			1	x	x		x	x	x	x
	Replicated signature							x	x		x	x	1	1
	Possible faults during retransmission	loosing						x			x		x	
		corrupting						x			x		x	
		replicating						x			x		x	

Table 2. States of nodes with structurally correct signature

Typically, nodes with at least structurally correct signature (tab. 2) have their state determined by test result values (fault-free, unambiguously faulty with detailed test results, or ambiguously faulty). The identified graphs make the test results more or less probable. In addition, we can identify (using derived attributes) probable intermittent faults or probable permanent faults. Below we comment briefly groups of determined states.

For a correct node in state (a) we have no reason to suspect any faults. Similarly, we classify as correct nodes in states (c, d), especially if an attribute about correct retransmission was set. In most situations assigning a good, consistent signature to a graph different from a valid tree is caused by the faults in nodes belonging to the path from the node to the host node (sec. 4). State d) is possible only if more nodes are faulty or many signatures of neighbouring nodes are effected by intermittent faults in a node retransmitting signatures.

Nodes with consistent signatures and correct test results can have attributes denoting possibly faulty behaviour during generating or sending signatures (states b, e). In this case, we suspect most probably intermittent faults, according to derived attributes and precision of their assignment.

In spite of a collected good signature, a node in states (f, g) is faulty. A node incorrectly performs its protocol during creating a collection tree, due to intermitted faults or permanent faults not covered by testing procedures. In addition, in state (f) behaviour of a node during retransmitting of signatures is characterised by derived attributers. In other, vary rare situations, we can obtain these states after malicious corruption of the signature by another node.

A faulty node with permanent faults detected by a testing procedures will be typically in state (a'). If more nodes are faulty in the system, states (c', d') are also possible. A faulty node with detailed test results can also perform incorrectly the exploratory algorithm (states b', e'). This incorrect behaviour can be in accordance with the obtained test results. Otherwise, most probably some intermittent faults occurred in the node (apart from detected permanent faults).

An evidence about incorrect protocol execution supplements detected permanent faults of nodes in states (h, i). This information can be or can be not confirmed by detailed test results. As in above cases, additional attributes pointing out to possible intermittent faults are also possible.

Ambiguously faulty nodes (states j, k) are very low probable. Replicated signatures with different faulty test results can be obtained due to malicious faults in the node or in other nodes. For state (j) deceiving nodes may proceed the node in the same deduced graph. Derived attributes in this state can also specify possible errors during algorithm execution.

Signature type →		Structurally incorrect							Missing signature						
States →		l	m	n	o	p	q	r	s	t	u	w	x	y	z
Graph	Probable tree or graph	1			1	1			1			1	1		
	Deduced graph		1							1					
	Unidentified graph			1			1	1			1			1	1
Attributes	Consistent neighbourhood	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	Correct protocol	1	1	1	x				1	1	1	x			
	Correct retransmission				x							x			
	Possible faults during retransmission: loosing				1							1			
	Possible faults during retransmission: corrupting														
	Possible faults during retransmission: replicating														
	No co-operation						1							1	

Table 3. States of nodes without structurally correct signature

In the classification of nodes without any structurally correct signature (tab. 3) we base on information obtained from the neighbouring nodes and from the reconstructed collection graphs, the nodes belongs to. In

partially correct nodes faults can occur after correct realisation of the protocol, but before sending of the own signature, which can be corrupted (states l, m) or lost (s, t). This is more probable for the sequential than for parallel exploration, because in the first case the own signature is send later, not before servicing all neighbouring links. For parallel exploration it is more probable that the node is non-faulty in states (l, s), especially when all three attributes are set, but the signature was not collected due to faults in one of nodes proceeding the node in the reconstructed collection graph.

A node in states (n, u) correctly co-operates during protocol execution, but besides it is probably faulty, especially if it has a neighbouring node with a consistent signature belonging to a valid or probable tree.

A node is faulty but non-blocked in states (o, w). However, its incorrect behaviour is specified only by derived attributes, due to missing detailed test results. Similarly about nodes in states (p, x) we can only assume, that they co-operate with the neighbours performing exploratory algorithm, but probably corrupt (p) or not send (x) their signatures. There is no definite information about their correct or incorrect behaviour.

A node in state (q) is probably faulty, was temporarily blocked during the protocol performing and sent corrupted signature. However, especially for inconsistent neighbourhood and neighbours with identified faults we have no data to accurate diagnose this node.

A dead node is not able to send signatures and co-operate with the neighbouring nodes (state y).

A non-diagnosed node (states r, z) can be isolated by other faulty nodes or by nodes with lost signatures. We can only declare, that in state (r) a node was non-blocked, but in state (z) could have been blocked.

4. Fault analysis

Diagnostic accuracy depends on the completeness of the set of collected signatures. Therefore in many cases the final identification can not be restricted to separate node states, but the system should be studied as a combination of correlated node states. It is especially important if faulty and suspicious nodes belong to the same reconstructed graph (tree) or to neighbours of the graph.

In order to better specify the possible system states we consider possible manifestations of faults in the system. Different state patterns can be recognised depending on the system configuration. As an example, we study possible impact on signature collecting process of single faults or faults sparsely distributed in the system. The consequences of selected faults expressed in possibly occurring states of nodes (notation from tab.2.,3.) are presented in tables 4.,5. For faulty node u_i two cases are recognised: when faults are covered by the testing procedures and their results stored in the signature (column “covered”), or testing results give no information about occurring faults (column “uncovered”). The later case is possible, for example, when permanent faults were not detected by the testing procedures or intermittent faults occur during collecting of signatures. A faulty node correctly generates and sends its own signature, if not stated otherwise.

Table 4. describes faults, which do not cause any malicious behaviour of the node. In this case, a corrupting of signature signifies changing a structurally correct signature to a structurally incorrect one. If a node u_i is a leaf in the collecting tree (has no children), we do not consider rows in the table describing faulty retransmitting signatures of other nodes. Otherwise, if the faulty node is a parent node engaged in signature retransmitting from its offsprings, all rows describing faulty behaviour are considered. For a parent node, some attributes (impossible for a leaf node) are additionally set in given states. They are written in italic font. In the last column the possible states of correlated nodes are listed. Nodes creating a path from node u_i to the host node can obtain an appropriate attribute specifying possible faults. Then their state will be changed from a to b (column “path”). For a parent node in additional column, states of children nodes of the faulty node and all their offsprings are listed. For a dead node, there are some correlated nodes in the isolated state z*) only if this node belongs to a structurally unique path directing from this nodes to the host node.

Table 5. describes less probable situations resulting in maliciously changed (falsified) signatures due to faults in node u_i. Such faulty node can send a false signature which has a correct syntax format and good CRC. We distinguish three elementary cases of a false signature: with a false identifier, false appointed states of neighbours or false test results. For each case, there are two rows in tab. 5 describing two different situations. A faulty node (a leaf or a parent in the tree) sends own false signature. If the node is a parent, it retransmits correctly other signatures. In the second situation, a faulty node is a parent node which falsifies any received signature (changes one part of it) and sends the false signature to its parent.

In the case of a false node identifier, a correlated node (pointed by the false identifier) can belong not only to the same graph as node u_i but even to another graph. Signature of the correlated node will be

replicated ([r)] in tab.5). For this faulty case, we can obtain the rare situation, when two consistent signatures of the same node with different neighbourhood relations were collected. Additionally, states of offspring nodes in the tree can be changed if the faulty node is a parent node (because of missing signature of node u_i). When the remaining faulty cases are considered (false states of neighbours or false test results) only states of offsprings nodes are effected. Then only these nodes can belong to correlated nodes if the faulty node is a parent node.

Non-malicious faults in node u_i			Node u_i state		Correlated nodes	
			covered	uncovered	path	offsprings
Faults with no communication impact			a’	a	-	
Faults disturbing communication	sending corrupted signatures to its parent	own signature	l, *correct retransmission*		b	c
		signatures of other nodes	b’	b		children l; m, n, r
			possible corrupting during retransmission			
		all signatures	l, *correct retransmission*		b	m, n, r
	loosing signatures having correctly appointed parent	own signature	s, *correct retransmission*		b	c
		signatures of other nodes	b’	b		children s; t, u, z
			possible loosing during retransmission			
		all signatures	s		b	t, u, z
	sending own signature to incorrectly appointed parent		a’-i	a-g	-	c, d
	appointing parent incorrectly, sends no signature		x- z		-	
	replicating signatures	own signature	b’	b	b	-
			replicated signature			
		signatures of other nodes	b’	b		b, replicated signature
			possible replicating during retransmission			
	dead node		y		-	z [*)]

Table 4. Consequences of permanent or intermittent faults in node u_i

Malicious faults in node u_i	Node u_i state		Correlated nodes	
	covered	uncovered	covered	uncovered
Faulty node falsifies node identifiers				
A node sends own false signature to its parent	s		b’ [r)]	b [r)]
			b	
A parent node sends other false signatures to its parent	a’	a	b’ [r)]	b [r)]
			s- u, x, z	
Faulty node falsifies relations with neighbours				
A node deceives during appointing process	b’-i	b-g	b-d, s-u, z	
A parent node falsifies relations in other signatures	a’, b’	a, b	b-g, b’-i incompatible	
Faulty node falsifies test results				
A node falsifies test results in own signatures	a, a’	a’	-	
A parent node falsifies test results in other signatures	a’	a	a’	

Table 5. Consequences of malicious faults in node u_i

In tables 4, 5 only the elementary faulty behaviours are discussed. A combination of listed effects encountering in the same node can result in the sum of possible states of the considered nodes. For some

faults (especially intermittent and malicious ones) it can be noticed, that more signatures with correct test results can be collected using broadcasting algorithm [4] than exploratory algorithms. However, retransmitting signatures along all possible paths generates more traffic in the system and complexity of analysis algorithms is higher in the case of broadcasting.

5. Conclusions

Considering algorithms for test results analysis we can observe that some of them are applicable only in special situations. We create more sophisticated collecting graph only if the simpler one does not cover all possible nodes. Moreover, some graphs are build under certain conditions, e.g. a deduced graph can be created only if there are minimum two neighbouring nodes without structurally correct signatures (necessary but not sufficient condition). We can also disregard some attributes or choose simpler algorithm for their determination. Usually, if ambiguous, inconsistent or incompatible signatures were collected, more sophisticated analysis should be used. In general, diagnostic reasoning can be performed on three levels (with increasing computational complexity):

1^0 *elementary analysis* - based on the direct signature interpretation and simple derived attributes,

2^0 *structural transmission analysis* - based on trees reconstruction for restricted situations and node neighbourhood analysis,

3^0 *detailed analysis* - based on exhaustive signature propagation analysis under assumed hypothesis about probably impact of faults on system behaviour.

Level 1 analysis is effective for permanent faults (single or sparsely distributed). Ambiguous signatures and a large number of incorrect signatures are handled with level 2 analysis. Level 3 is recommended for malicious behaviour of faulty modules. In majority situations with most probable fault distribution, the algorithms with lower complexity are sufficient. For many rarely appearing complex faults more sophisticated algorithms can give better accuracy of system state. As compared with the known approaches the obtained diagnostic results are more precise and comprise additional information useful in system repair and error recovery processes.

References

1. J. Altman, T. Barta, A. Pataricza, A.Petri, P.Urban: *Constraint based system-level diagnosis of multiprocessors*, in : Proc. of Dependable Computing EDCC-2, LNCS-1150, Springer 1996, pp.403-420
2. M. Barboak, M. Malek A. Dahbura: *The consensus problem in fault-tolerant computing*, ACM Computing Surveys, vol.25, no.2, June 1993, pp.172-220.
3. O. Benkhala, C. Aktouf, C. Robach: *Distributed off-line testing of parallel systems*, in IEEE Proc. of 4th Asian Test Symp., 1995, pp.2-8.
4. A. Derezińska, J. Sosnowski: *Enhanced diagnostics of MIMD systems*, Proc of EWDC-10, Vienna, May, 1999, pp.129-134
5. A. Derezińska: *Parallel software for exploratory diagnostics in MIMD systems*, Proc. of Int. Conf. on Parallel Computing in Electrical Eng, Parelec'98, Białystok, Sept., pp.109-114. 1998
6. A. Derezińska, J. Sosnowski: *Exploratory diagnostics of MIMD systems*, in Proc. of Euromicro'95, 1995, pp.421-428.
7. E. P. Duarte, T. Nanya: *A hierarchical adaptive distributed system-level diagnosis algorihm*, in: IEEE Transactions on Computers, vol.47, no.1 January 1998.
8. S. Rangarajan, A. T. Dahbura, E. A. Ziegler: *A distributed system-level diagnosis algorithm for arbitrary network topologies*, IEEE Trans. on Computers, vol. 44, no.2, February, 1995, pp.312-333.
9. M. Raynal, J-M. Helary: *Synchronization and control of distributed systems and programs*, J.Wiley & Sons, 1990.