Gradient Based System-level Diagnosis

Balázs Polgár Endre Selényi

January 17, 2007

Abstract

Traditional approaches in system-level diagnosis in multiprocessor systems are usually based on the oversimplified PMC test invalidation model, however Blount introduced a more general model containing conditional probabilities as parameters for different test invalidation situations. He suggested a lookup table based approach, but no algorithmic solution has been elaborated until our P-graph based solution introduced in previous publications. In this approach the diagnostic process is formulated as an optimization problem and the optimal solution is determined. Although the average behavior of the algorithm is quite good, the worst case complexity is exponential. In this paper we introduce a novel group of fast diagnostic algorithms that we named *gradient based algorithms*. This approach only approximates the optimal maximum likelihood or maximum a posteriori solution, but it has a polynomial complexity of the magnitude of $O\left(N \cdot NbCount + N^2\right)$, where N is the size of the system and $NbCount$ is number of neighbors of a single unit.

The idea of the base algorithm is that it takes an initial fault pattern and iterates till the likelihood of the actual fault pattern can be increased with a single state-change in the pattern. Improvements of this base algorithm, complexity analysis and simulation results are also presented.

The main, although not exclusive application field of the algorithms is *wafer-scale diagnosis*, since the accuracy and the performance is still good even if relative large number of faults are present.

1 Introduction

Diagnosis is one of the major tools for assuring the reliability of complex systems in information technology. In such systems the test process is often implemented on system-level: the 'intelligent' components of the system test their local environment and each other. The test results are collected, and based on this information the good or faulty state of each system-component is determined. This classification procedure is known as *diagnostic process*.

The early approaches that solve the diagnostic problem employed oversimplified binary fault models [1], could only describe homogeneous systems, and assumed the faults to be permanent. Since these conditions proved to be impractical, lately much effort has been put into extending the limitations of traditional

models [2, 3]. However, the presented solutions mostly concentrated on only one aspect of the problem.

In our previous research we applied the P-graph based modeling to system-level diagnosis [4] that provided a general framework for supporting the solution of several different types of problems, that previously needed numerous different modeling approaches and solution algorithms. Furthermore, we have not only integrated existing solution methods, but proceeding from a more general base we have extended the set of solvable problems with new ones. The representational power of the model was illustrated in paper [5].

Another advantage of the P-graph models is that it takes into consideration more properties of the real system than previous diagnostic models. Therefore its diagnostic accuracy is also better. This means that it provides almost good diagnosis even when half of the processors are faulty [6]. This is important for the field of wafer scale testing [7, 8, 9], which was the primary initiator of our research.

The only disadvantage of the P-graph based diagnosis is that it has an exponential worst case complexity although the average performance is quite good. That is why we developed this new algorithm-family starting from the same base but using different modeling technique and aiming only an *approximation* – although a good approximation – of the optimal solution while having *polynomial* complexity.

The paper is structured as follows. First an overview is given about system level diagnosis in multiprocessor systems. Then the likelihood of fault patterns and the change of likelihood upon state-changes in the fault pattern are discussed. This serves as base for the algorithm, which is presented next. Extensions of the algorithm are also suggested that can improve the accuracy. It is also shown how fault probability can be taken into account if it is known in order to have maximum a posteriori diagnosis. A possible implementation of the base algorithm is also given and the time and space complexity of it is determined. Finally simulation results are presented; the diagnostic accuracy of the algorithms and the relationship to other algorithms are analyzed here.

2 System-Level Diagnosis

System-level diagnosis considers the *replaceable units* of a system, and does not deal with the exact location of faults within these units. A *system* consists of an interconnected network of independent but cooperating *units* (typically processors). The fault state of each unit is either *good* when it behaves as specified, or *faulty*, otherwise. The *fault pattern* is the collection of the fault states of all units in the system. A unit may test the *neighboring* units connected with it via direct links. The network of the units testing each other determines the *test topology*. The outcome of a test can be either *passed* or *failed* (denoted by 0/1 or G/F); this result is considered *valid* if it corresponds to the actual physical state of the tested unit.

The collection of the results of every completed test is called the *syndrome*.

The test topology and the syndrome are represented graphically by the *test graph*. The vertices of a test graph denote the units of the system, while the directed arcs represent the tests originated at the *tester* and directed towards the *tested unit* (UUT). The result of a test is shown as the label of the corresponding arc. Label 0 represents the passed test result, while label 1 represents the failed one. See Figure 1 for an example test graph with three units.

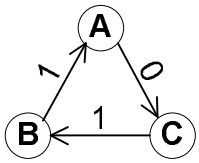


Figure 1: Example test graph (test topology with syndrome)

2.1 Traditional Approaches

Traditional diagnostic algorithms assume that

1. faults are permanent,
2. states of units are binary (*good*, *faulty*),
3. the test results of good units are always valid, i.e. good testers are perfect or in other words test coverage is 100%,
4. the test results of faulty units can also be invalid. The behavior of faulty tester units is expressed in the form of *test invalidation models*.

Fig. 2 shows the fault model of a single test and Table 1 covers the possible test invalidation models, where the selection of c and d values determines a specific model. The most widely used example is the so-called PMC (Preparata, Metze, Chien) test invalidation model [1] ($c = any, d = any$), which considers the test result of a faulty tester to be independent of the state of the tested unit. According to another well-known test invalidation model, the BGM (Barsi, Grandoni, Maestrini) model [10] ($c = any, d = faulty$) a faulty tester will always detect the failure of the tested unit, because it is assumed that the probability of two units failing the same way is negligible.

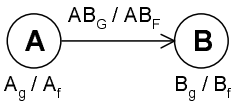


Figure 2: Fault model of a single test

The purpose of system-level diagnostic algorithms is to determine the fault state of each unit from the syndrome. The difficulty comes from the possibility

Table 1: Traditional test invalidation models

State of tester	State of UUT	Test result
good	*good*	*passed*
good	*faulty*	*failed*
faulty	*good*	$c \in \{passed, failed, any\}$
faulty	*faulty*	$d \in \{passed, failed, any\}$

that a fault in the tester processor invalidates the test result. As a consequence, multiple "candidate" diagnoses can be compatible with the syndrome. To provide a complete diagnosis and to select from the candidate diagnoses, the so-called *deterministic* algorithms use extra information in addition to the syndrome, such as assumptions on the size of the fault pattern or on the testing topology.

Alternatively, *probabilistic* algorithms try to determine the most probable diagnosis assuming that a unit is more likely good than faulty [11]. Frequently, this maximum likelihood strategy can be expressed simply as "many faults occur less frequently than a few faults." Thus, the aim of diagnosis is to determine the minimal set of faulty elements of the system that is consistent with the syndrome.

2.2 The Generalized Approach

In our previous work [4, 12, 5] we used a generalized test invalidation model, introduced by Blount [13]. In this model, probabilities are assigned to both possible test outcomes for each combination of the states of tester and tested units (Table 2). Since the good and faulty results are complementary events, the sum of the probabilities in each row is 1. The assumption of the complete fault coverage can be relaxed in the generalized model by setting probability p_{b1} to the fault coverage of the test. Probabilities p_{c0}, p_{c1}, p_{d0} and p_{d1} express the distortion of the test results by a faulty tester. Moreover, the generalized model is able to encompass *false alarms* (a good tester finds a good unit to be faulty) by setting probability p_{a1} to nonzero, however, it is not a typical situation.

Table 2: Generalized test model

State of tester	State of UUT	Probability of test result	
		0	**1**
good	*good*	p_{a0}	p_{a1}
good	*faulty*	p_{b0}	p_{b1}
faulty	*good*	p_{c0}	p_{c1}
faulty	*faulty*	p_{d0}	p_{d1}

Of course, the generalized test invalidation model covers the traditional models. Setting the probabilities as $p_{a0} = p_{b1} = 1$, $p_{c0} = p_{c1} = p_{d0} = p_{d1} = 0.5$, and $p_{a1} = p_{b0} = 0$, the generalized model has the characteristics of the PMC model, while the configuration $p_{a0} = p_{b1} = p_{d1} = 1$, $p_{c0} = p_{c1} = 0.5$ and

$p_{a1} = p_{b0} = p_{d0} = 0$ makes it behave like the BGM model. Analogically, every traditional test invalidation model can be mapped as a special case to this model.

3 Likelihood of Fault Patterns

3.1 Formulization of Likelihood of Fault Patterns

To determine the maximum likelihood diagnosis the $P(\textit{syndrome} \mid \textit{fault pattern})$ conditional probability should be maximized over the fault patterns. I.e. the fault pattern that produces the observed syndrome with the highest probability should be found.

Let's denote with $p_{z|\mathbf{st_i}}(z \mid \mathbf{st_i})$ the conditional probability mass function determining the distribution of the syndromes if $\mathbf{st_i}$ is the fault pattern.

Furthermore let's denote with functions $n_{a0}(\mathbf{st_i}, z)$, $n_{a1}(\mathbf{st_i}, z)$, $n_{b0}(\mathbf{st_i}, z)$, $n_{b1}(\mathbf{st_i}, z)$, $n_{c0}(\mathbf{st_i}, z)$, $n_{c1}(\mathbf{st_i}, z)$, $n_{d0}(\mathbf{st_i}, z)$, $n_{d1}(\mathbf{st_i}, z)$ the number of the different types of tests where $\mathbf{st_i}$ is the fault pattern and z is the syndrome (types are differentiated according to the states of the tester and tested unit and according to the test result; types are denoted with indices $a0$, $a1$, $b0$ etc. as in Table 2.).

Probability $P(\textit{syndrome} \mid \textit{fault pattern})$ can be expressed as the product of the conditional probabilities $P(\textit{test result} \mid \textit{state of tester, state of tested unit})$ if test results in the syndrome are independent [14]. Formally,

$$\begin{aligned} p_{z|\mathbf{st_i}}(z \mid \mathbf{st_i}) = & {p_{a0}}^{n_{a0}(\mathbf{st_i},z)} \cdot {p_{a1}}^{n_{a1}(\mathbf{st_i},z)} \cdot {p_{b0}}^{n_{b0}(\mathbf{st_i},z)} \cdot {p_{b1}}^{n_{b1}(\mathbf{st_i},z)} \cdot \\ & \cdot {p_{c0}}^{n_{c0}(\mathbf{st_i},z)} \cdot {p_{c1}}^{n_{c1}(\mathbf{st_i},z)} \cdot {p_{d0}}^{n_{d0}(\mathbf{st_i},z)} \cdot {p_{d1}}^{n_{d1}(\mathbf{st_i},z)} \end{aligned} \tag{1}$$

3.2 Change in Likelihood of Fault Patterns

In this section we determine the difference between the conditional probabilities of a given syndrome for two fault patterns that have 1 Hamming distance between them.

3.2.1 Effect of Changing the State of a Unit from Good to Faulty

Let's consider an arbitrary $\mathbf{st_i}$ fault pattern and an arbitrary unit (the unit that has index k; referred later as the k^{th} unit) that is in good state according to this fault pattern. Let's change the state of this unit to faulty and denote the resulted fault pattern with $\mathbf{st_i}^{k,\mathbf{f}}$.

As a result the values of functions n_{a0}, n_{a1}, ..., n_{d1} change: the tests related to the selected unit will have new types. For instance if this unit has tested another unit to be good then this test had *type a0* and it had a factor p_{a0} in the probability $p_{z|\mathbf{st_i}}(z \mid \mathbf{st_i})$. After the change it has *type c0* and has a factor p_{c0} in probability $p_{z|\mathbf{st_i}^{k,\mathbf{f}}}(z \mid \mathbf{st_i}^{k,\mathbf{f}})$. This means that the given test caused

a change in probability $P(\textit{syndrome} \mid \textit{fault pattern})$ of the amount $\frac{p_{c0}}{p_{a0}}$ as the result of the state change.

Table 3 summarizes the possible relationships between the selected unit and its neighbors and the effects of these in the conditional probability $P(\textit{syndrome} \mid \textit{fault pattern})$. The functions in the last column of the table have three input parameters: $\mathbf{st_i}$, z and k ($fn_{g0}(\mathbf{st_i}, z, k)$, ...). These functions determine the number of neighbors of the k^{th} unit having the given type if $\mathbf{st_i}$ is the fault pattern and z is the syndrome.

Table 3: Change in the number of tests of a given unit and given type and the effect of the change for the likelihood of the fault pattern if the state of the unit is changed from *good* to *faulty*

	kind	**state**	**test**	**type of test**		**clfp**[a]	**fnt**[b]
of the neighbor			**result**	**before**	**after**		
$\xrightarrow{0}\circ$	tested	good	0	a0 ($\circ \xrightarrow{0} \circ$)	c0 ($\bullet \xrightarrow{0} \circ$)	$\frac{p_{c0}}{p_{a0}}$	fn_{g0}
$\xrightarrow{1}\circ$	tested	good	1	a1 ($\circ \xrightarrow{1} \circ$)	c1 ($\bullet \xrightarrow{1} \circ$)	$\frac{p_{c1}}{p_{a1}}$	fn_{g1}
$\xrightarrow{0}\bullet$	tested	faulty	0	b0 ($\circ \xrightarrow{0} \bullet$)	d0 ($\bullet \xrightarrow{0} \bullet$)	$\frac{p_{d0}}{p_{b0}}$	fn_{f0}
$\xrightarrow{1}\bullet$	tested	faulty	1	b1 ($\circ \xrightarrow{1} \bullet$)	d1 ($\bullet \xrightarrow{1} \bullet$)	$\frac{p_{d1}}{p_{b1}}$	fn_{f1}
$\circ\xrightarrow{0}$	tester	good	0	a0 ($\circ \xrightarrow{0} \circ$)	b0 ($\circ \xrightarrow{0} \bullet$)	$\frac{p_{b0}}{p_{a0}}$	bn_{g0}
$\circ\xrightarrow{1}$	tester	good	1	a1 ($\circ \xrightarrow{1} \circ$)	b1 ($\circ \xrightarrow{1} \bullet$)	$\frac{p_{b1}}{p_{a1}}$	bn_{g1}
$\bullet\xrightarrow{0}$	tester	faulty	0	c0 ($\bullet \xrightarrow{0} \circ$)	d0 ($\bullet \xrightarrow{0} \bullet$)	$\frac{p_{d0}}{p_{c0}}$	bn_{f0}
$\bullet\xrightarrow{1}$	tester	faulty	1	c1 ($\bullet \xrightarrow{1} \circ$)	d1 ($\bullet \xrightarrow{1} \bullet$)	$\frac{p_{d1}}{p_{c1}}$	bn_{f1}

[a] *clfp = change in the likelihood of the fault pattern,* i.e. the change in the conditional probability $P(\textit{syndrome} \mid \textit{fault pattern})$ for the given type of test caused by the state-change of the selected unit

[b] *fnt = functions determining the number of tests of the given type*; The abbreviations come from the words ***f**orward **n**eighbour* and ***b**ackward **n**eighbour*; the index indicates the state of the neighbor and the result of the test

The relations between the conditional mass functions can be expressed with the functions defined in the table:

$$p_{z|\mathbf{st_i}^{k,\mathbf{f}}}(z \mid \mathbf{st_i}^{k,\mathbf{f}}) =$$
$$= \frac{{p_{b0}}^{bn_{g0}} \cdot {p_{b1}}^{bn_{g1}} \cdot {p_{c0}}^{fn_{g0}} \cdot {p_{c1}}^{fn_{g1}} \cdot {p_{d0}}^{bn_{f0}+fn_{f0}} \cdot {p_{d1}}^{bn_{f1}+fn_{f1}}}{{p_{a0}}^{bn_{g0}+fn_{g0}} \cdot {p_{a1}}^{bn_{g1}+fn_{g1}} \cdot {p_{b0}}^{fn_{f0}} \cdot {p_{b1}}^{fn_{f1}} \cdot {p_{c0}}^{bn_{f0}} \cdot {p_{c1}}^{bn_{f1}}} \cdot p_{z|\mathbf{st_i}}(z \mid \mathbf{st_i})$$

Let's introduce the notion $\Delta_{z,f}(\mathbf{st_i}, k)$ for the quotient of the two conditional probability:

$$\Delta_{z,f}(\mathbf{st_i}, k) = \frac{p_{z|\mathbf{st}_\mathbf{i}^{k,\mathbf{f}}}(z \mid \mathbf{st}_\mathbf{i}^{k,\mathbf{f}})}{p_{z|\mathbf{st_i}}(z \mid \mathbf{st_i})} =$$

$$= \begin{cases} 1, & \text{if } \mathbf{st_i}[k]{=}f; \\ \frac{{p_{b0}}^{bn_{g0}} \cdot {p_{b1}}^{bn_{g1}} \cdot {p_{c0}}^{fn_{g0}} \cdot {p_{c1}}^{fn_{g1}} \cdot {p_{d0}}^{bn_{f0}+fn_{f0}} \cdot {p_{d1}}^{bn_{f1}+fn_{f1}}}{{p_{a0}}^{bn_{g0}+fn_{g0}} \cdot {p_{a1}}^{bn_{g1}+fn_{g1}} \cdot {p_{b0}}^{fn_{f0}} \cdot {p_{b1}}^{fn_{f1}} \cdot {p_{c0}}^{bn_{f0}} \cdot {p_{c1}}^{bn_{f1}}}, & \text{otherwise.} \end{cases} \quad (2)$$

3.2.2 Effect of Changing the State of a Unit to the Opposite

Similarly to the previous section we can define $\mathbf{st}_\mathbf{i}^{k,\,\mathbf{g}}$ as the fault pattern derived from $\mathbf{st_i}$ with changing the state of the k^{th} unit to *good* and we can define the change in the conditional mass functions determining the likelihood of a syndrome in case of these fault patterns:

$$\Delta_{z,g}(\mathbf{st_i}, k) = \frac{p_{z|\mathbf{st}_\mathbf{i}^{k,\,\mathbf{g}}}(z \mid \mathbf{st}_\mathbf{i}^{k,\,\mathbf{g}})}{p_{z|\mathbf{st_i}}(z \mid \mathbf{st_i})} \quad (3)$$

Combining the two case we can introduce $\mathbf{st}_\mathbf{i}^{\overline{k}}$ as the fault pattern that differs from $\mathbf{st_i}$ exactly in the state of the k^{th} unit. Let's define function $\Delta_z(\mathbf{st_i}, k)$ as the function that determines the change in the likelihood $P(\textit{syndrome} \mid \textit{fault pattern})$ if the state of the k^{th} unit is changed to the opposite in fault pattern $\mathbf{st_i}$:

$$\Delta_z(\mathbf{st_i}, k) = \frac{p_{z|\mathbf{st}_\mathbf{i}^{\overline{k}}}(z \mid \mathbf{st}_\mathbf{i}^{\overline{k}})}{p_{z|\mathbf{st_i}}(z \mid \mathbf{st_i})} = \begin{cases} \Delta_{z,f}(\mathbf{st_i}, k), & \text{if } \mathbf{st_i}[k]{=}g; \\ \Delta_{z,g}(\mathbf{st_i}, k), & \text{if } \mathbf{st_i}[k]{=}f. \end{cases} \quad (4)$$

The value of function $\Delta_z(\mathbf{st_i}, k)$ belonging to $\mathbf{st_i}[k]{=}f$ is the reciprocal of the value belonging to $\mathbf{st_i}[k]{=}g$ because the likelihood of a fault pattern must be unchanged if the state of one of its unit is changed to the opposite and then back again. This and Eq. (2) implies the final form for the Δ-function:

$$\Delta_z(\mathbf{st_i}, k) = \begin{cases} \frac{{p_{b0}}^{bn_{g0}} \cdot {p_{b1}}^{bn_{g1}} \cdot {p_{c0}}^{fn_{g0}} \cdot {p_{c1}}^{fn_{g1}} \cdot {p_{d0}}^{bn_{f0}+fn_{f0}} \cdot {p_{d1}}^{bn_{f1}+fn_{f1}}}{{p_{a0}}^{bn_{g0}+fn_{g0}} \cdot {p_{a1}}^{bn_{g1}+fn_{g1}} \cdot {p_{b0}}^{fn_{f0}} \cdot {p_{b1}}^{fn_{f1}} \cdot {p_{c0}}^{bn_{f0}} \cdot {p_{c1}}^{bn_{f1}}}, & \text{if } \mathbf{st_i}[k]{=}g; \\ \frac{{p_{a0}}^{bn_{g0}+fn_{g0}} \cdot {p_{a1}}^{bn_{g1}+fn_{g1}} \cdot {p_{b0}}^{fn_{f0}} \cdot {p_{b1}}^{fn_{f1}} \cdot {p_{c0}}^{bn_{f0}} \cdot {p_{c1}}^{bn_{f1}}}{{p_{b0}}^{bn_{g0}} \cdot {p_{b1}}^{bn_{g1}} \cdot {p_{c0}}^{fn_{g0}} \cdot {p_{c1}}^{fn_{g1}} \cdot {p_{d0}}^{bn_{f0}+fn_{f0}} \cdot {p_{d1}}^{bn_{f1}+fn_{f1}}}, & \text{if } \mathbf{st_i}[k]{=}f. \end{cases}$$

In later sections we will refer to this $\Delta_z(\mathbf{st_i}, k)$ function as $\Delta_{z,ML}(\mathbf{st_i}, k)$, too, when this maximum likelihood version is compared to the maximum a posteriori version of the function.

4 Gradient Based Algorithm

Using the notion of the previous section we can state the following:

If the value of the function Δ_z for an arbitrary unit of an arbitrary fault pattern is greater than 1 then changing the state of this unit results a fault

pattern that has larger likelihood than the original one; thus, it is closer to the optimal solution.

The gradient based algorithm is based on this property as it is shown in this section.

4.1 The Base Algorithm

The steps of the base algorithm are the followings:

1. Take an *initial fault pattern* ($\mathbf{st_0}$, i.e. i=0).
2. Let's count the value of function $\Delta_z(\mathbf{st_i}, k)$ for every k ($k = 1..N$), i.e. let's determine the effect of changing the state of each single unit in the actual fault pattern upon the likelihood of it.
3. Let's choose the *maximal* Δ_z value: $\Delta_{z,max}(\mathbf{st_i}) = \max_k \Delta_z(\mathbf{st_i}, k)$.
4. If this value is *greater than 1*, then change the state of the corresponding unit in the fault pattern: this will be the next fault pattern ($\mathbf{st_{i+1}}$); and go back to step 2.
5. If the maximal value is *not greater than 1*, then ready, the result of the diagnosis is $\mathbf{st_i}$.

The efficiency of the algorithm is greatly depend on the initial fault pattern. Three main types can be identified:

- each unit is in *good* state
  ($\mathbf{st_0} = \mathbf{st_{allg}} = gg\ldots g$),
- each unit is in *faulty* state
  ($\mathbf{st_0} = \mathbf{st_{allf}} = ff\ldots f$),
- each unit is in *random* state
  ($\mathbf{st_0} = \mathbf{st_{rand}}$; $P(\mathbf{st_{rand}}[k] = g) = 0.5$, $k = 1..N$).

According to simulations the first one results quite good diagnosis, the second one results quite bad and the accuracy is highly varying in case of the third one. Thus, the first is the best choice, however, the third one has practical significance, too, as it will turn out later.

4.2 Algorithm Extension I: Changing the State of Multiple Units Simultaneously

The disadvantage of the base algorithm is that it searches for better solution only among fault patterns that are 1 Hamming distance far from the actual pattern. Thus it finds often only a local maximum. In order to find the global or a better local maximum the search can be extended in each round to fault patterns that are 2, 3 or more Hamming distance far from the actual pattern.

Let's change the state of at most H unit in each round. In this case function Δ_z should be defined different:

- Let's sum the different types of tests that have a selected unit either as a tester or as a tested unit and a non-selected as the other one (similarly as previously), but differentiate according to the state of the selected unit. The functions $fn_{g0,g}$, $fn_{g1,g}$, $fn_{f0,g}$, ... $bn_{f1,g}$, and $fn_{g0,f}$, $fn_{g1,f}$, $fn_{f0,f}$, ... $bn_{f1,f}$ are defined this way (see Table 3, too).

  As previously these functions have also three input parameters, but beside $\mathbf{st_i}$ and z the third one is not the index of a single unit (k), but the set of indices of the selected units ($\mathbf{k}$).

- Those tests should be summed by types, too, that have selected units both as tester and as tested unit. In these tests we assume that the state of both units will change. The number of these tests are defined by functions bs_{a0}, bs_{a1}, ..., bs_{d1}, see Table 4. Of course these functions have $\mathbf{st_i}$, z and $\mathbf{k}$ as input parameters.

Table 4: Change in the number of tests of different types and the effect of the change for the likelihood of the fault pattern if the state of both unit is changed to the opposite

state of tester	**state of tested unit**	**test result**	**type of test**		**clfp**[a]	**fnt**[b]
			before	**after**		
good	good	0	a0 (○ $\xrightarrow{0}$ ○)	d0 (● $\xrightarrow{0}$ ●)	$\frac{p_{d0}}{p_{a0}}$	bs_{a0}
good	good	1	a1 (○ $\xrightarrow{1}$ ○)	d1 (● $\xrightarrow{1}$ ●)	$\frac{p_{d1}}{p_{a1}}$	bs_{a1}
good	faulty	0	b0 (○ $\xrightarrow{0}$ ●)	c0 (● $\xrightarrow{0}$ ○)	$\frac{p_{c0}}{p_{b0}}$	bs_{b0}
good	faulty	1	b1 (○ $\xrightarrow{1}$ ●)	c1 (● $\xrightarrow{1}$ ○)	$\frac{p_{c1}}{p_{b1}}$	bs_{b1}
faulty	good	0	c0 (● $\xrightarrow{0}$ ○)	b0 (○ $\xrightarrow{0}$ ●)	$\frac{p_{b0}}{p_{c0}}$	bs_{c0}
faulty	good	1	c1 (● $\xrightarrow{1}$ ○)	b1 (○ $\xrightarrow{1}$ ●)	$\frac{p_{b1}}{p_{c1}}$	bs_{c1}
faulty	faulty	0	d0 (● $\xrightarrow{0}$ ●)	a0 (○ $\xrightarrow{0}$ ○)	$\frac{p_{a0}}{p_{d0}}$	bs_{d0}
faulty	faulty	1	d1 (● $\xrightarrow{1}$ ●)	a1 (○ $\xrightarrow{1}$ ○)	$\frac{p_{a1}}{p_{d1}}$	bs_{d1}

[a] *clfp = change in the likelihood of the fault pattern*

[b] *fnt = functions determining the number of tests of the given type*; The abbreviation comes from the phrase ***b**oth **s**elected*

Similarly to the previous notations let's denote with $\mathbf{st_i^{\overline{k}}}$ the fault pattern that we get from $\mathbf{st_i}$ with changing the state of units that are contained in set $\mathbf{k}$.

Now function $\Delta_z(\mathbf{st_i}, \mathbf{k})$ can be defined as follows:

$$\Delta_z(\mathbf{st_i}, \mathbf{k}) = \frac{p_{z|\mathbf{st}_\mathbf{i}^{\overline{\mathbf{k}}}}(z \mid \mathbf{st}_\mathbf{i}^{\overline{\mathbf{k}}})}{p_{z|\mathbf{st_i}}(z \mid \mathbf{st_i})} =$$

$$= \frac{{p_{b0}}^{bn_{g0,g}} \cdot {p_{b1}}^{bn_{g1,g}} \cdot {p_{c0}}^{fn_{g0,g}} \cdot {p_{c1}}^{fn_{g1,g}} \cdot {p_{d0}}^{bn_{f0,g}+fn_{f0,g}} \cdot {p_{d1}}^{bn_{f1,g}+fn_{f1,g}}}{{p_{a0}}^{bn_{g0,g}+fn_{g0,g}} \cdot {p_{a1}}^{bn_{g1,g}+fn_{g1,g}} \cdot {p_{b0}}^{fn_{f0,g}} \cdot {p_{b1}}^{fn_{f1,g}} \cdot {p_{c0}}^{bn_{f0,g}} \cdot {p_{c1}}^{bn_{f1,g}}} \cdot$$

$$\cdot \frac{{p_{a0}}^{bn_{g0,f}+fn_{g0,f}} \cdot {p_{a1}}^{bn_{g1,f}+fn_{g1,f}} \cdot {p_{b0}}^{fn_{f0,f}} \cdot {p_{b1}}^{fn_{f1,f}} \cdot {p_{c0}}^{bn_{f0,f}} \cdot {p_{c1}}^{bn_{f1,f}}}{{p_{b0}}^{bn_{g0,f}} \cdot {p_{b1}}^{bn_{g1,f}} \cdot {p_{c0}}^{fn_{g0,f}} \cdot {p_{c1}}^{fn_{g1,f}} \cdot {p_{d0}}^{bn_{f0,f}+fn_{f0,f}} \cdot {p_{d1}}^{bn_{f1,f}+fn_{f1,f}}} \cdot$$

$$\cdot \frac{{p_{a0}}^{bs_{d0}} \cdot {p_{a1}}^{bs_{d1}} \cdot {p_{b0}}^{bs_{c0}} \cdot {p_{b1}}^{bs_{c1}} \cdot {p_{c0}}^{bs_{b0}} \cdot {p_{c1}}^{bs_{b1}} \cdot {p_{d0}}^{bs_{a0}} \cdot {p_{d1}}^{bs_{a1}}}{{p_{a0}}^{bs_{a0}} \cdot {p_{a1}}^{bs_{a1}} \cdot {p_{b0}}^{bs_{b0}} \cdot {p_{b1}}^{bs_{b1}} \cdot {p_{c0}}^{bs_{c0}} \cdot {p_{c1}}^{bs_{c1}} \cdot {p_{d0}}^{bs_{d0}} \cdot {p_{d1}}^{bs_{d1}}}$$

Using this $\Delta_z(\mathbf{st_i}, \mathbf{k})$ function the steps of the gradient based algorithm is modified in this extended version according to the followings:

1. Take an *initial fault pattern* ($\mathbf{st_0}$; i=0).
2. Count the value of function $\Delta_z(\mathbf{st_i}, \mathbf{k})$ for every set $\mathbf{k}$ that contains at least 1 and at most H units.
3. Choose the *maximal* Δ_z value.
4. If this value is *greater than 1*, then in the fault pattern change the state of each unit in set $\mathbf{k}$ that corresponds to the maximal Δ_z value: this will be the next fault pattern ($\mathbf{st_{i+1}}$); and go back to step 2.
5. If the maximal value is *not greater than 1*, then ready, the result of the diagnosis is $\mathbf{st_i}$.

With this extension the accuracy of the diagnosis can be improved: as H tends to $N-1$ the diagnosis tends to the maximum likelihood diagnosis. But increasing H increases the complexity, too. As it tends to $N-1$ the complexity tends to exponential.

4.3 Algorithm Extension II: Multiple Run

In this subsection such an extension is suggested that can improve the diagnostic accuracy without significantly increasing the complexity.

The main idea is to run the base algorithm multiple times with different initial fault patterns and choose the maximal maximum. The steps of the algorithm in more details are the followings:

1. Take an *initial fault pattern* ($\mathbf{st_{0,1}}$, i.e. i=0, j=1).
2. Run the base algorithm having $\mathbf{st_{0,j}}$ as the initial fault pattern; denote the result of it with $\mathbf{st_{sol,j}}$.
3. Determine the likelihood of the solution, i.e. the conditional probability $p_{z|\mathbf{st_{sol,j}}}(z \mid \mathbf{st_{sol,j}})$ (see Eq. (1)).

4. If this likelihood is bigger than the likelihood of the best solution found till the moment then this will be the best solution ($\mathbf{st_{sol}} = \mathbf{st_{sol,j}}$).

5. If j has not reached a certain bound, the so-called *run-number* then take a new initial fault pattern ($\mathbf{st_{0,j+1}}$) and go back to step 2.

6. If j has reached the run-number then ready; the result of the diagnosis is $\mathbf{st_{sol}}$.

In this extension to choose random fault patterns as initial ones is satisfactory if the run-number is big enough, although it is worth to choose $\mathbf{st_{allg}}$ as the first pattern, because it results quite a good diagnosis in itself.

Although with every further round the final solution approximates the optimal one better and better, we have to determine the run-number somehow. It can be *constant*, although a better choice is if it *depends on the size of the system* or it is determined *adaptively*, i.e. the algorithm is stopped if no better solution is found in a given number of trials after the last 'best-solution update' in step 4. In the later case relatively few rounds is enough if we found a good solution early, but should try much more further if we found each time only a little bit better solution compared to the previous one.

Simulations showed that with this extension the optimal solution can be approximated quite well only with a small increase in the complexity.

4.4 Model Extension: Maximum a Posteriori Diagnosis

In case of maximum a posteriori diagnosis the $P(\textit{fault pattern} \mid \textit{syndrome})$ conditional probability should be maximized over the fault patterns [15]. I.e. the fault pattern that has the highest probability in case of the observed syndrome should be found. In this case that fault pattern should be chosen for which the value

$$p_{z|\mathbf{st_i}}(z \mid \mathbf{st_i}) \cdot P(\mathbf{st_i})$$

is maximal. If we suppose that the units fail independently then the probability of fault pattern $\mathbf{st_i}$ can be expressed as the product of the probabilities of the states of the units determined by the fault pattern. If we suppose a homogeneous system, i.e. each unit has the same fault probability p_f, then it turns to the following form:

$$P(\mathbf{st_i}) = \prod_{k=1}^{N} P(\mathbf{st_i}[k]) = (1-p_f)^{N_g(\mathbf{st_i})} \cdot {p_f}^{N_f(\mathbf{st_i})}, \tag{5}$$

where functions $N_g(\mathbf{st_i})$ and $N_f(\mathbf{st_i})$ determine the number of good and faulty units in the fault pattern $\mathbf{st_i}$. This implies that *maximum a posteriori diagnosis can be determined only if the fault probabilities of units are known.*

Similarly to Sec. 3.2.2 let's define $\Delta_{z,MAP}(\mathbf{st_i}, k)$ as the function that determines the change in conditional probability $P(\textit{fault pattern} \mid \textit{syndrome})$ if we

change the state of the k^{th} unit to the opposite in the fault pattern $\mathbf{st_i}$. This function can be formulated in the following form:

$$\begin{aligned}\Delta_{z,MAP}(\mathbf{st_i},k) &= \frac{p_{z|\mathbf{st}_\mathbf{i}^{\bar{k}}}(z \mid \mathbf{st}_\mathbf{i}^{\bar{k}}) \cdot P(\mathbf{st}_\mathbf{i}^{\bar{k}})}{p_{z|\mathbf{st_i}}(z \mid \mathbf{st_i}) \cdot P(\mathbf{st_i})} = \\ &= \Delta_{z,ML}(\mathbf{st_i},k) \cdot \frac{(1-p_f)^{N_g(\mathbf{st}_\mathbf{i}^{\bar{k}})} \cdot {p_f}^{N_f(\mathbf{st}_\mathbf{i}^{\bar{k}})}}{(1-p_f)^{N_g(\mathbf{st_i})} \cdot {p_f}^{N_f(\mathbf{st_i})}} = \\ &= \begin{cases} \Delta_{z,ML}(\mathbf{st_i},k) \cdot \frac{p_f}{1-p_f}, & \text{if } \mathbf{st_i}[k]{=}g; \\ \Delta_{z,ML}(\mathbf{st_i},k) \cdot \frac{1-p_f}{p_f}, & \text{if } \mathbf{st_i}[k]{=}f. \end{cases}\end{aligned} \tag{6}$$

This implies that in the algorithms described in previous sections only the Δ-values should be modified with factor $\frac{p_f}{1-p_f}$ or $\frac{1-p_f}{p_f}$ according to the state-change and the result will be *maximum a posteriori diagnosis.*

Of course, homogeneity is not a requirement; if fault probabilities are specific for units then always that fault probability should be used during counting the value $\Delta_{z,MAP}$ that belongs to the unit the state of which is to be changed.

4.5 Implementation of the Base Algorithm

Among the steps of the base algorithm given in Sec. 4.1 only the evaluation of function $\Delta_z(\mathbf{st_i},k)$ needs further discussion; the implementation of all others is trivial (for choosing the maximum we use the simplest linear search).

To evaluate function $\Delta_z(\mathbf{st_i},k)$ the functions $fn_{g0}(\mathbf{st_i},z,k)$, $fn_{g1}(\mathbf{st_i},z,k)$ etc. should be determined, i.e. we have to count that in how many tests of the given type are the units involved. But it seems to be simpler to iterate over the tests and include the value determined by the type of the test to the Δ-value of the two affected units (the phrase 'Δ-value of the k^{th} unit' is an abbreviation for $\Delta_z(\mathbf{st_i},k)$, where $\mathbf{st_i}$ and z are the actual fault pattern and syndrome). Moreover, this iteration have to be done only once for the initial fault pattern, in later steps only the Δ-values of the selected unit and its neighbors should be modified, all others remain unaltered. It was shown that state change of a unit reciprocates its Δ-value, thus in the followings only the effect for the Δ-values of the neighbors should be determined.

Table 5 summarizes the change in the Δ-values of neighbors in the case when the state of the selected unit is changed from *good* to *faulty.* The opposite change in the state of the selected unit will result a reciprocal change in the Δ-values of neighbors similarly to the change in the likelihood of the fault pattern (see Sec. 3.2.2).

Let's introduce the following notations:

$$DIFF0 = \frac{p_{a0} \cdot p_{d0}}{p_{b0} \cdot p_{c0}} \qquad \text{and} \qquad DIFF1 = \frac{p_{a1} \cdot p_{d1}}{p_{b1} \cdot p_{c1}}.$$

Table 6 summarizes with these notations the change in the Δ-values of the neighbors in the different cases. It can be observed that this change – beside

Table 5: Change in the Δ-values of neighbors having different types resulted from the state-change of the selected unit (the state change is from *good* to *faulty*).

	kind	**state**	**test res.**	**Δ-value of the neighbor belonging to this test**		**change in Δ-value of**
of the neighbor				**before change**	**after change**	**neighbor**
	tested	good	0	$\frac{p_{b0}}{p_{a0}}$	$\frac{p_{d0}}{p_{c0}}$	$\frac{p_{a0}}{p_{b0}} \cdot \frac{p_{d0}}{p_{c0}}$
	tested	good	1	$\frac{p_{b1}}{p_{a1}}$	$\frac{p_{d1}}{p_{c1}}$	$\frac{p_{a1}}{p_{b1}} \cdot \frac{p_{d1}}{p_{c1}}$
	tested	faulty	0	$\frac{p_{a0}}{p_{b0}}$	$\frac{p_{c0}}{p_{d0}}$	$\frac{p_{b0}}{p_{a0}} \cdot \frac{p_{c0}}{p_{d0}}$
	tested	faulty	1	$\frac{p_{a1}}{p_{b1}}$	$\frac{p_{c1}}{p_{d1}}$	$\frac{p_{b1}}{p_{a1}} \cdot \frac{p_{c1}}{p_{d1}}$
	tester	good	0	$\frac{p_{c0}}{p_{a0}}$	$\frac{p_{d0}}{p_{b0}}$	$\frac{p_{a0}}{p_{c0}} \cdot \frac{p_{d0}}{p_{b0}}$
	tester	good	1	$\frac{p_{c1}}{p_{a1}}$	$\frac{p_{d1}}{p_{b1}}$	$\frac{p_{a1}}{p_{c1}} \cdot \frac{p_{d1}}{p_{b1}}$
	tester	faulty	0	$\frac{p_{a0}}{p_{c0}}$	$\frac{p_{b0}}{p_{d0}}$	$\frac{p_{c0}}{p_{a0}} \cdot \frac{p_{b0}}{p_{d0}}$
	tester	faulty	1	$\frac{p_{a1}}{p_{c1}}$	$\frac{p_{b1}}{p_{d1}}$	$\frac{p_{c1}}{p_{a1}} \cdot \frac{p_{b1}}{p_{d1}}$

Table 6: Change in the Δ-values of neighbors having different types resulted from the state-change of the selected unit (the state change is arbitrary).

	kind	**state**	**test res.**	**change in Δ-value of the neighbor, if the state of the selected unit changes**	
of the neighbor				**from *good* to *faulty***	**from *faulty* to *good***
	tested	good	0	$DIFF0$	$\frac{1}{DIFF0}$
	tested	good	1	$DIFF1$	$\frac{1}{DIFF1}$
	tested	faulty	0	$\frac{1}{DIFF0}$	$DIFF0$
	tested	faulty	1	$\frac{1}{DIFF1}$	$DIFF1$
	tester	good	0	$DIFF0$	$\frac{1}{DIFF0}$
	tester	good	1	$DIFF1$	$\frac{1}{DIFF1}$
	tester	faulty	0	$\frac{1}{DIFF0}$	$DIFF0$
	tester	faulty	1	$\frac{1}{DIFF1}$	$DIFF1$

the test result – depends only on the fact that the units involved in the test are in similar or in different states.

Taking all these into account a possible implementation of the base algorithm can be found in Table 7.

Table 7: Implementation of the base version of the gradient based algorithm

(a) Parameters, variables, functions

Input:	N	size of the system
	$NbCount$	number of neighbors
	$TestRes(i,k)$	result of the k^{th} test of the i^{th} unit
	$NeighbourInd(i,k)$	index of the k^{th} tested neighbor of the i^{th} unit
	$BacklinkInd(i,k)$	index of the unit that has the i^{th} unit as the k^{th} tested neighbor
	$Prob(st_1, st_2, tr)$	probability of test result tr if the tester is in state st_1 and the tested unit is in state st_2, i.e. the result of it is one of the values p_{a0}, p_{a1}, $\dots p_{d1}$
Used functions:	$GetIniState() : stateArray$	determines the initial fault pattern
	$CountDelta(stateArray) : deltaArray$	determines the Δ_z values for each unit ($deltaArray$) if the states of units are determined by $stateArray$
	SelectMax(array, out maxElement, out maxInd)	determines the maximal element ($maxElement$) of the $array$ and the index of it ($maxInd$)
	$Neg(state) : state$	returns the negation of the $state$
Inner variables:	$stateArray$	array with N element that holds the actual fault pattern (the i^{th} element determines the state of the i^{th} unit)
	$deltaArray$	array with N element; the i^{th} element determines the Δ_z value belonging to the state-change of the i^{th} unit (it corresponds to the actual fault pattern)
	$maxDelta$	maximal Δ-value in the given round
	$maxInd$	index of the unit that has maximal Δ-value in the given round (i.e. it is the selected unit)
	$nbInd$	index of a neighbor of the selected unit
Output:	$stateArray$	at the end of the algorithm it contains the diagnosed fault pattern

(b) Function *CountDelta*

CountDelta*(stateArray):deltaArray* **begin**	
for $i := 1$ **to** N **do** $deltaArray[i] := 1.0$;	Initialization
for $i := 1$ **to** N **do**	Loop on units
for $k := 1$ **to** $NbCount$ **do** **begin**	Loop on the tests of the actual unit
$nbInd := NeighbourInd(i,k)$; $st_{tr} := stateArray[i]$; $st_{td} := stateArray[nbInd]$; $tr := TestRes(i,k)$;	Temporary variables: index of the neighbor state of the tester state of the tested unit test result
$deltaArray[i] := deltaArray[i] \cdot \frac{Prob(Neg(st_{tr}),st_{td},tr)}{Prob(st_{tr},st_{td},tr)}$;	Δ-value belonging to this test of the i^{th} unit ($\rightarrow$ the state of the i^{th} unit changes)
$deltaArray[nbInd] := deltaArray[nbInd] \cdot \frac{Prob(st_{tr},Neg(st_{td}),tr)}{Prob(st_{tr},st_{td},tr)}$;	Δ-value belonging to this test of the unit having index $nbInd$ ($\rightarrow$ the state of the $nbInd^{th}$ unit changes)
end; **end;**	

(c) The Body of the Algorithm

$stateArray := GetIniState();$ $deltaArray := CountDelta(stateArray);$	Initialization
$SelectMax(deltaArray, maxDelta, maxInd);$	Determination of the maximal Δ-value
while $maxDelta > 1.0$ **do** **begin**	Loop while the diagnosis can be improved
for $k := 1$ **to** $NbCount$ **do** **begin**	Loop on the neighbors

$nbInd := NeighbourInd(maxInd, k);$
if $stateArray[nbInd] = stateArray[maxInd]$ **then**
if $TestRes(maxInd, k) = g$ **then**
$deltaArray[nbInd] := deltaArray[nbInd] \cdot DIFF0;$
else
$deltaArray[nbInd] := deltaArray[nbInd] \cdot DIFF1;$
else
if $TestRes(maxInd, k) = g$ **then**
$deltaArray[nbInd] := deltaArray[nbInd]/DIFF0;$
else
$deltaArray[nbInd] := deltaArray[nbInd]/DIFF1;$
	Modification of the Δ-value of the k^{th} tested neighbor of the selected unit

$nbInd = BacklinkInd(maxInd, k);$
if $stateArray[nbInd] = stateArray[maxInd]$ **then**
if $TestRes(nbInd, k) = g$ **then**
$deltaArray[nbInd] := deltaArray[nbInd] \cdot DIFF0;$
else
$deltaArray[nbInd] := deltaArray[nbInd] \cdot DIFF1;$
else
if $TestRes(nbInd, k) = g$ **then**
$deltaArray[nbInd] := deltaArray[nbInd]/DIFF0;$
else
$deltaArray[nbInd] := deltaArray[nbInd]/DIFF1;$
	Modification of the Δ-value of the tester unit that has the selected unit as the k^{th} tested neighbor

end;	
$stateArray[maxInd] := Neg(stateArray[maxInd]);$ $deltaArray[maxInd] := 1/deltaArray[maxInd];$	Modification of the state and Δ-value of the selected unit
$SelectMax(deltaArray, maxDelta, maxInd);$	Determination of the maximal Δ-value
end.	

4.6 Complexity of the Algorithm

4.6.1 Time Complexity

The complexity of the functions used in the base algorithm is different:

- Functions $TestRes(i,k)$, $Prob(st_1, st_2, tr)$, $NeighbourInd(i,k)$ and $BacklinkInd(i,k)$ are executed in *constant* time as these return only an element of an array.
- Functions $GetIniState()$ and $SelectMax()$ are executed in time $O(N)$.
- The initialization of the arrays behind functions $NeighbourInd(i,k)$ and $BacklinkInd(i,k)$ need steps in the magnitude $O(N \cdot NbCount)$.

  The complexity of running function $CountDelta(stateArray)$ once is $O(N + N \cdot NbCount) = O(N \cdot NbCount)$.

Now using these information let's determine the complexity of the steps of the base algorithm:

- The determination of the initial fault pattern needs $O(N)$ time.
- To count the initial Δ-values $O(N \cdot NbCount)$ time is needed.
- The maximum of the initial Δ-values can be chosen in $O(N)$ time.
- The body of the *while* loop runs in $O(NbCount)+O(N) = O(NbCount + N)$ time.

  That is a more complex task to determine the number the loop will iterate. Theoretically it can happen that the state of certain units will change back and forth many times till the solution is found. But it is more probable that the state of most units has to be changed at most once and there are only a few units the state of which has to be changed back because of the changes in the environment. The reason why this is more probable is that a state change results a reciprocal Δ-value what – considering that it was the highest Δ-value – results an expectably small value. This can grow only in small amounts through the state changes of the neighbors thus a lot of changes are needed in the environment this value to be increased above 1 and the state of the unit to be able to change again.

  For instance if the initial fault pattern contains only good states then practically only the state of the faulty units changes once and the state of those good units changes twice that are surrounded by faulty units. Accordingly we can state – and simulations validate it – that the loop-body runs

  - N_f times if initial fault pattern is $\mathbf{st_{allg}}$ (N_f is the number of faulty units),
  - $N/2$ times on the average if initial fault pattern is $\mathbf{st_{rand}}$,
  - but at most N times in worst case.

Thus it can be stated that the *while* loop depending on the initial fault pattern needs $O\left(NbCount \cdot N_f + N \cdot N_f\right)$, but in worst case at most $O\left(NbCount \cdot N + N^2\right)$ time.

Summarizing the complexity of the steps the following arises: the average time *complexity of the base algorithm is* $O\left(N \cdot NbCount + N_f \cdot NbCount + N \cdot N_f\right)$ *if initial fault pattern contains only good states and it is at most* $O\left(N \cdot NbCount + N^2\right)$ *in other cases.*

4.6.2 Space complexity

In the base algorithm we use variables and one-dimensional arrays (*stateArray* and *deltaArray*) that have N elements. Further arrays needed for the functions $TestRes(i,k)$, $NeighbourInd(i,k)$ and $BacklinkInd(i,k)$ that have $N \cdot NbCount$ elements. Thus *the space complexity of the base algorithm is* $O\left(N \cdot NbCount\right)$.

5 Simulation results

We have implemented a simulation program and the gradient based and other diagnostic algorithms. The program simulates the behavior of multiprocessor systems and measures various properties of the algorithms and compares them.

In the simulation program the most general toroidal mesh topology is implemented having size and number of neighbors as parameters. It generates the fault pattern using a given fault probability and generates a syndrome for it using the probabilities defined in the Blount test invalidation model (these are also input parameters). The program determines the diagnosis according to the different algorithms, compares it to the original fault pattern and calculates several accuracy and complexity related properties of the diagnosis that are cumulated over several simulation rounds. In this section two accuracy related properties are presented:

- the average number of misdiagnosed processors relative to the system size ($mavg$) and
- the rate of rounds that contained at least one misdiagnosed processor (mn).

During simulations we used fault probability as varying parameter over a wide range. In everyday systems the maximum value should be below a few percent but in wafer scale diagnosis a larger part of the system can be faulty: it can even happen that much more than the half of the processors are faulty.

5.1 Comparison and Analysis of Gradient Based Algorithms

5.1.1 Parameters

Size of mesh: 10×10

Number of tested neighbors : 4

Fault probability: $0,1 - 0,2 - 0,3 - ... - 0,9 - 1$

Test invalidation: $p_{a0} = 1 \quad p_{b0} = 0 \quad p_{c0} = 0{,}5 \quad p_{d0} = 0{,}5$

Number of simulation rounds: 500

Algorithms: base algorithm, $\mathbf{st_0} = \mathbf{st_{allg}}$ (ba_g)
algorithm extended for 10 runs, $\mathbf{st_0} = \mathbf{st_{rand}}$ (mra_{r10})
algorithm extended for 100 runs, $\mathbf{st_0} = \mathbf{st_{rand}}$ (mra_{r100})
algorithm extended for 10 runs, $\mathbf{st_0} = \begin{cases} \mathbf{st_{allg}}, & 1\text{x}; \\ \mathbf{st_{rand}}, & 9\text{x}. \end{cases}$ $(mra_{g,r9})$
algorithm extended for 100 runs, $\mathbf{st_0} = \begin{cases} \mathbf{st_{allg}}, & 1\text{x}; \\ \mathbf{st_{rand}}, & 99\text{x}. \end{cases}$ $(mra_{g,r99})$
algorithm extended for 2 Hamming distance search (Ha_2)
algorithm extended for 3 Hamming distance search (Ha_3)

5.1.2 Discussion

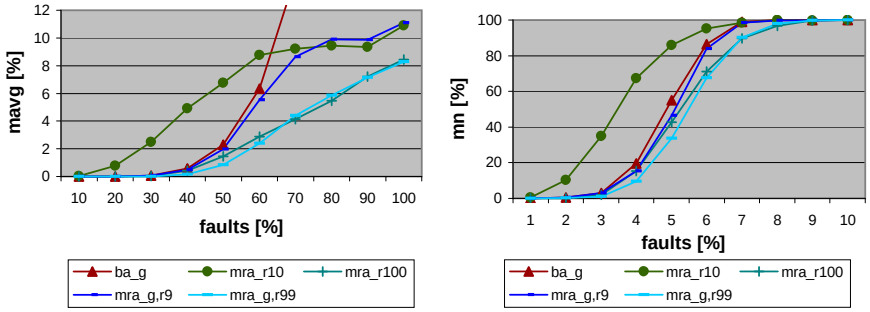
The diagnostic accuracy related properties are shown on Fig. 3 and in Table 8. The table and the figure contains the same data except that Ha_2 and Ha_3 are included only in the table.

It can be seen that all algorithms except mra_{r10} provide always the good diagnosis if the fault probability is 10%. It means that these algorithms can be used in installed systems where the aim of the diagnosis is to keep the integrity of the network of cooperating processors.

If the algorithms are used in wafer scale diagnosis then higher fault rates are also interesting. Even the base algorithm with initial fault pattern containing good states (ba_g) provides a quite good diagnosis, but it can be improved if the algorithm is run multiple times having random initial fault patterns from the second run. However, the best results are achieved by extending the algorithm for multiple Hamming distance search. Nevertheless it is not sure that this is the most useful algorithm, because it has weak performance properties: while the base algorithm needs about 0.2 seconds to run 100 times, the Ha_2 algorithm needs some seconds and the Ha_3 algorithm needs minutes for a single run.

Comparing algorithms mra_{r10} and $mra_{g,r9}$ it can be seen that it is worth on every account to use $\mathbf{st_{allg}}$ as initial fault pattern. The difference decreases as the number of initial random patterns increases but it can be observed in case of algorithms mra_{r100} and $mra_{g,r99}$, too.

The results that can be seen on Fig. 4(a) and 4(b) verify the considerations described in Sec. 4.6.1. Fig. 4(c) shows the average time needed to run the



(a) Average number of misdiagnosed processors relative to system size

(b) Rate of rounds containing misdiagnosed processors

Figure 3: Accuracy related properties of gradient based algorithms

Table 8: Accuracy related properties of gradient based algorithms

(a) Average number of misdiagnosed processors relative to system size

mavg[%]	***fault probability [%]***									
algorithm	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
ba_g	0	0.01	0.07	0.58	2.3	6.34	14.4	22.3	30.8	35.6
mra_{r10}	0.03	0.8	2.5	4.92	6.77	8.77	9.23	9.46	9.36	10.9
mra_{r100}	0	0	0.03	0.41	1.44	2.88	4.13	5.46	7.21	8.44
$mra_{g,r9}$	0	0.01	0.05	0.49	1.96	5.52	8.64	9.91	9.86	11.1
$mra_{g,r99}$	0	0	0.01	0.17	0.85	2.38	4.39	5.85	7.15	8.27
Ha_2		0		0.16		2.02		10.18		17.16
Ha_3		0		0.1		1.2		5.81		10.17

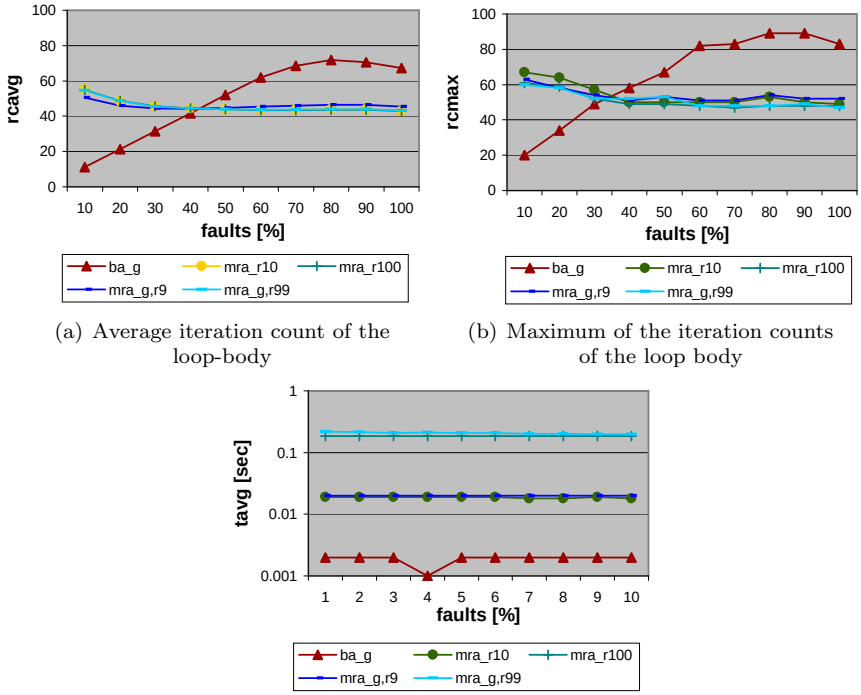
(b) Rate of rounds containing misdiagnosed processors

mn[%]	***fault probability [%]***									
algorithm	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
ba_g	0	0.4	3	19.6	55	86.4	99	100	100	100
mra_{r10}	0.4	10.2	35	67.4	86	95.2	98.6	100	99.8	100
mra_{r100}	0	0	2	15.4	42.6	71.2	89.8	96.8	99.8	100
$mra_{g,r9}$	0	0.6	3	15.4	46.4	83.8	98.4	99.8	99.8	100
$mra_{g,r99}$	0	0.2	1	9.6	33.6	67.6	90.2	98	99.8	100
Ha_2		0.4		8.8		58.2		98.8		100
Ha_3		0		6.8		48.4		96.4		99.8

algorithms. It can be seen on the first two figures that only the complexity of the ba_g algorithm depends on the number of the faulty units.

5.2 Comparison of Gradient Based Algorithms to Other Algorithms

There is only one solution in the literature for the Blount test invalidation model, namely the one that Blount himself suggested. It is based on a lookup table that is calculated in advance and which contains the solution for all situations. Although this is the fastest solution that can be created, it has no practical significance because of its memory consumption and preprocessing work. Con-



(a) Average iteration count of the loop-body

(b) Maximum of the iteration counts of the loop body

(c) Average time (sec)

Figure 4: Complexity related properties of gradient based algorithms

sequently, we can compare our algorithms only to solutions that are based on the traditional test invalidation models, and to our previously developed P-graph based algorithm.

During simulations the well-known algorithms taken from the literature were the LDA1 algorithm of Somani and Agarwal [16, 17], the Dahbura, Sabnani and King (DSK) algorithm [18] and the limited multiplication of inference matrix (LMIM) algorithm developed by Bartha and Selenyi [19, 20] from the area of local information diagnosis.

We have examined three situation:

- expectations of other algorithms are met, i.e. PMC test invalidation is used (good testers are perfect, faulty testers can say anything with equal probability)
- expectations of other algorithms are nearly met, i.e. PMC-like test invalidation is used (good testers are perfect, faulty testers can say anything but with different probabilities)
- test coverage is less than 100% (i.e. good testers are not perfect)

5.2.1 Parameters

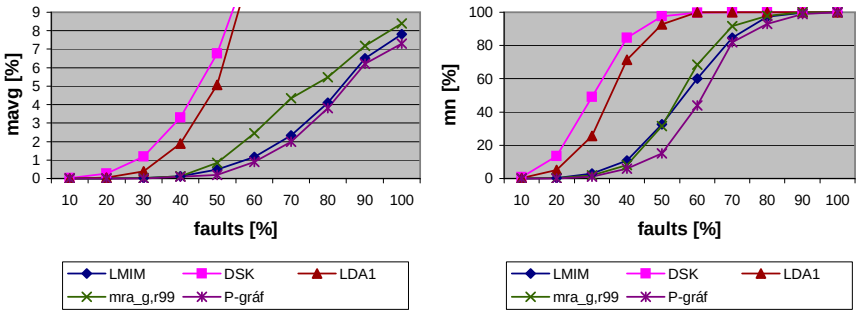
Size of mesh: 10×10
Number of tested neighbors : 4
Fault probability: $0,1 - 0,2 - 0,3 - ... - 0,9 - 1$
Test invalidation:

$p_{a0} =$	1	1	1
$p_{b0} =$	0	0	0,2
$p_{c0} =$	0,5	0,9	0,5
$p_{d0} =$	0,5	0,1	0,5

Number of simulation rounds: 500
Algorithm: LMIM, DSK, LDA1, P-graph, $\text{mra}_{\text{g,r99}}$

5.2.2 Discussion

The results when PMC test invalidation is assumed are shown on Fig. 5. It is trivial that P-graph based algorithm provides better results, because it determines the optimal diagnosis using the Blount test invalidation model. But it can be seen that gradient based algorithms can provide nearly the same results, but the complexity of it is about $O\left(n^2\right)$-$O\left(n^3\right)$ (in later case the run-number is directly proportional to the system size) in contrast to the worst-case exponential complexity of the P-graph based algorithm.

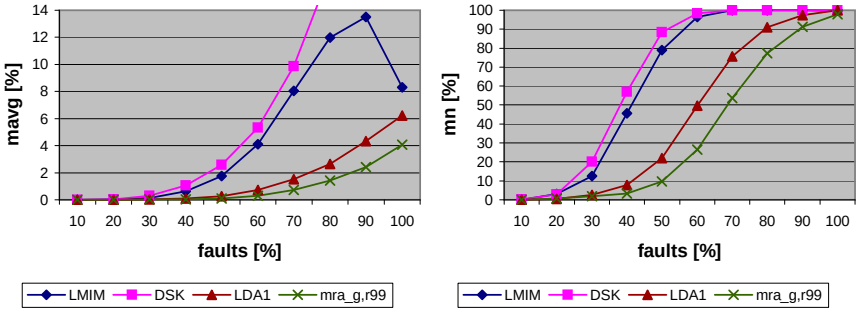


(a) Average number of misdiagnosed processors relative to system size

(b) Rate of rounds containing misdiagnosed processors

Figure 5: Gradient based algorithm compared to other algorithms assuming PMC test invalidation

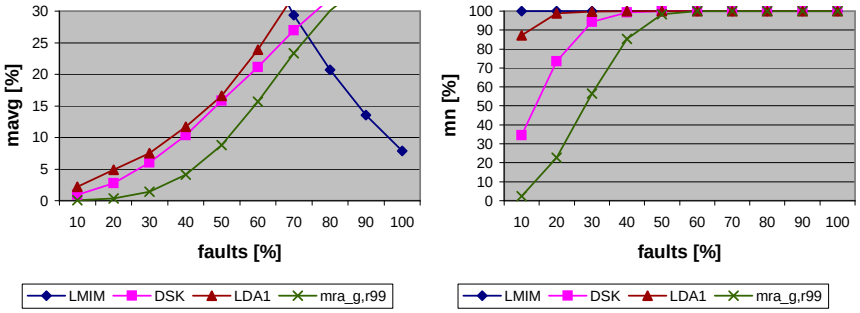
The results on Fig. 6 and 7 and other performed simulations showed that in most cases other algorithms could handle the non-standard situations, too, but the diagnostic accuracy of these changed very differently in different situations. The diagnostic accuracy was varying in case of the gradient based algorithm, too, but it provided the best results in every situation.



(a) Average number of misdiagnosed processors relative to system size

(b) Rate of rounds containing misdiagnosed processors

Figure 6: Diagnostic accuracy of the different algorithms if PMC like test invalidation is assumed ($p_{c0} = 0,9$ and $p_{d0} = 0,1$)



(a) Average number of misdiagnosed processors relative to system size

(b) Rate of rounds containing misdiagnosed processors

Figure 7: Diagnostic accuracy of the different algorithms if fault coverage of tests is 80%

6 Conclusions

In this paper a new family of solution algorithms, called *gradient based algorithms* are presented that provide system level diagnosis in multiprocessor systems. The novelty of the algorithms is that it approaches the problem from a more general base than traditional algorithms, namely it uses the Blount test invalidation model that describes the behavior of the testing process with probabilities. The base algorithm differs from our previous P-graph based algorithm in that it is very fast, though on the other hand it only approximates the maximum likelihood diagnosis even if it provides a good approximation.

The idea of the base algorithm is that it takes an initial fault pattern and examines the change in the likelihood if the state of a unit is changed in the pattern. It selects the maximal likelihood-delta and performs the corresponding state-change. This process is iterated till the delta is greater than 1, i.e. till the

likelihood of the actual fault pattern can be increased with a single state-change.

Two extensions of the algorithm are also presented. The first tries to increase the likelihood with multiple state-changes in the actual fault pattern. Its accuracy tends to the optimal if cardinality of the multiple state-changes tends to the system size, but its complexity meanwhile tends to the exponential. The second extension is a quite straightforward one: the base algorithm has to be run multiple times started from random initial fault patterns and the best should be chosen from the set of resulted fault patterns that have locally maximum likelihoods. According to simulations with this second extension the optimal solution can be approximated quite well.

It was also demonstrated how the fault probability should be inserted into the calculations in order to determine the maximum a posteriori diagnosis.

A further remarkable strength of the algorithms is that these can be applied not only in installed systems, but also in wafer scale diagnosis where the fault probability can be even around or above 30-40%. The simulations performed and partially presented in the last section confirmed these good properties and the usefulness of the developed algorithms.

The next step in our research is the elaboration of a new version of the gradient based approximation algorithm that can solve problems formalized with P-graph models.

References

[1] F. P. Preparata, G. Metze, and R. T. Chien. On the connection assigment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, EC-16(6):848–854, December 1967.

[2] M. Barborak, M. Malek, and A. Dahbura. The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.

[3] T. Bartha and E. Selényi. Probabilistic system-level fault diagnostic algorithms for multiprocessors. *Parallel Computing*, 22:1807–1821, February 1997.

[4] B. Polgár, Sz. Nováki, A. Pataricza, and F. Friedler. A process graph based formulation of the syndrome-decoding problem. In *Proc. of DDECS2001, the 4th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop, Gyr*, pages 267–272, 2001.

[5] B. Polgár, E. Selényi, and T. Bartha. On the extension and applicability of the p-graph modeling paradigm to system-level diagnostic problems. *Scalable Computing: Practice and Experience*, 6(2):45–57, 2005.

[6] B. Polgár and E. Selényi. Efficiency of p-graph based syndrome decoding. In *Proceedings of DDECS2002, the 5th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop, Brno, Czech Republic*, pages 374–377, 2002.

[7] S. Rangarajan, D. Fussell, and M. Malek. Built-in testing of integrated circuit wafers. *IEEE Transactions on Computers*, 39(2):195–204, February 1990.

[8] A. Caruso, S. Chessa, and P. Meastrini. Comparison-based diagnosis of vlsi wafers. In *3rd IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pages 227–232. IEEE, 2000.

[9] B. Sallay. *Constraint-Based Architectural Test Pattern Generation.* PhD thesis, Budapesti Mszaki s Gazdasgtudomnyi Egyetem, Budapest, 2000.

[10] F. Barsi, F. Grandoni, and P. Maestrini. A theory of diagnosability of digital systems. *IEEE Transactions on Computers*, C-25(6):585–593, June 1976.

[11] S. H. Maheswari and S. L. Hakimi. On models for diagnosable systems and probabilistic fault diagnosis. *IEEE Transactions on Computers*, C-25(3):228–236, March 1976.

[12] B. Polgár, T. Bartha, and E. Selényi. Modeling uncertainty in system-level fault diagnosis using process graphs. In *Proc. of Dapsys2002, 4th Austrian-Hungarian Workshop on Distributed and Parallel Processing, Linz, Austria*, pages 195–202. Kluwer Ac. Pub. Group, September 29–October 2 2002.

[13] M. L. Blount. Probabilistic treatment of diagnosis in digital systems. In *7th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 72–77, June 1977.

[14] B. Polgár. Comparison based diagnostics as a probabilistic deduction problem. In *Proc. of ICCS 2004, International Conference on Computational Science, Krakow, Poland*, pages 1153–1161, 2004.

[15] L. Schnell, editor. *Jelek s rendszerek mrstechnikja.* Mszaki Knyvkiad, Budapest, 1985.

[16] A. Somani and V. Agarwal. Distributed syndrome decoding for regular interconnected structures. In *19th IEEE Int. Symp. on Fault-Tolerant Computing*, pages 70–77, New York, 1989. IEEE.

[17] A. Somani and V. Agarwal. Distributed diagnosis algorithms for regular interconnected structures. *IEEE Transactions on Computers*, 41(7):899–906, July 1992.

[18] A. Dahbura, K. Sabnani, and L. King. The comparison approach to multiprocessor fault diagnosis. *IEEE Transactions on Computers*, C-36(3):373–378, March 1987.

[19] T. Bartha and E. Selényi. Probabilistic fault diagnosis in large, heterogeneous computing systems. *Periodica Polytechnica*, 43(2):127–149, 2000.

[20] T. Bartha. *Efficient System-Level Fault Diagnosis of Large Multiprocessor Systems*. PhD thesis, Budapesti Mszaki s Gazdasgtudomnyi Egyetem, Budapest, 2000.