Constraint-based System Level Diagnosis Of Multiprocessor Architectures

A. Pataricza[†,‡], K. Tilly[†], E. Selényi[†], M. Dal Cin[‡], A.Petri[†]

[†] Department of Measurement
and Instrumentation Engineering
Technical University of Budapest
Müegyetem rkp. 9
H-1521 Budapest, Hungary
email: pataric@mmt.bme.hu

[‡] Department of Computer
Science III
University of Erlangen-Nürnberg
Martensstr. 3
91058 Erlangen, Germany
email: dalcin@informatik.uni-erlangen.de

Abstract

In the latest years, new ideas appeared in system level diagnosis. Contrary to the traditional diagnosis models (like PMC, BGM etc.) which use strictly graph-oriented methods to determine the faulty components in a system, these new theories prefer AI-based algorithms, especially CSP methods. Syndrome decoding, the basic problem of self diagnosis, can be easily transformed to constraints between the state of the tester and the tested components, considering the test results. Therefore, the diagnosis algorithm can be derived from a special constraint solving algorithm. The “benign” nature of the constraints (all their variables, representing the fault states of the components, have a very limited domain; the constraints are simple and similar to each other) reduces the algorithm’s complexity so it can be converted to a powerful distributed diagnosis method with a minimal overhead. An experimental algorithm was implemented for a Parsytec GC tightly coupled multiprocessor system.

1.Introduction

1.1 Traditional methodology of self-diagnosis

The construction of dependable systems is hardly possible without the application of some form of self-checking. Therefore different models and algorithms were developed for system level (self-)diagnosis.The majority of them are based on graph theory derived from the first so-called “system level models” published in the mid-sixties.

These introductory models (PMC for symmetric and BGM for asymmetric test invalidation) are the most well-known and most widely used ones. Their mathematical apparatus is simple and well-elaborated; both theoretical investigations and practical implementations proved their usefulness.

However these models have some implicit limitations preventing their use in many important fields of application. The test invalidation model is oversimplified in order to assure a proper mathematical treatment decreasing the level of reality of the models and reducing their usability.

The rapid development of electronic technology and computer architectures radically modified the basic assumptions used in the diagnostic model:

– the fault rates are much lower and the dominating part of faults is transient;
– the complexity of other system components is comparable with the complexity of the CPU;
– the complexity of the systems and the number of the computing elements have been drastically increased.

Most insufficiencies result from the hardest simplification of the PMC and BGM models: the assumption of a homogenous system (built of identical components with identical test invalidation). This assumption effectively reduces the range of applicability due to the growing practical importance of inhomogenous multiprocessor systems.

1.2 Required features of a new self-diagnosis method

The new requirements resulting from the latest results in multiprocessor system design, characterize the expected features of a proper, general purpose self-diagnosis method:

- it should be applicable in inhomogenous systems as well as in homogenous ones (different components with different test invalidation models are considered);
- the diagnostic resolution should only loosely depend on the actual system topology and/or test invalidation model (the currently used methods have serious restrictions on the system topology due to the use of rigid, inadaptive algorithms);
- the algorithm should extract all the useful information from the elementary diagnostic results (e.g. for estimating the level of diagnosis at run-time);
- it should cope with the latest massively parallel processor systems with several hundreds or even thousands of system components, thus the algorithm should have an excellent efficiency even for a very high number of units under test .

These requirements need a new approach.A generalized test invalidation model for syndrome decoding and diagnosis in inhomogenous systems is published in [2]. This model contains all sufficient and necessary conditions of one-step and sequential diagnosis for the different test invalidation models. However, its mathematical apparatus is not optimal (it uses complex matrix operations, e.g. computation of transitive closure); therefore the efficiency of the algorithm becomes a crucial factor in large-scale system diagnosis.

The most important step of self-diagnosis is basically the process of finding the possible fault states of system components based on the syndrome information. A systematic search method is required for effective syndrome decoding. Many applications even demand on-the-fly diagnosis for maximal performance; it requires a diagnosis method that is able to identify the fault states of some units from partial syndrome information. This is hardly achievable with traditional algorithms.

1.3 The use of AI-based methods and algorithms

The main intention of “artificial intelligence” (AI) methods is to find efficient solutions for difficultly solvable (to be more precise, generally NP-complete) or hard to represent problems. This gives a way to handle many practical but earlier unmanageable problems.

This aim is frequently reached by more sophisticated information management; it is often called “knowledge management” as it represents a high level of abstraction and a more flexible and efficient information extraction from elementary data.

Many well-elaborated, efficient and practically tested AI-based algorithms have been developed over the years. A group of them, the CS (Constraint Satisfaction) methods seem especially useful for a special self-diagnosis model [2].

Application of CS methods has already proven very attractive on fields closely related to system level diagnosis. For example, CS-based automated test pattern generation is presented in [3].

2. Constraint Satisfaction Problems and their handling

2.1 A brief definition of a CSP

A *constraint satisfaction problem* (CSP) can be described as an (X, D, C) tuple. $X = \{X_1,X_2,... X_n\}$ is a *set of variables*; $D = \{D_1,D_2,... D_n\}$ is a *set of domains* (each domain is a set associated with a variable and contains the allowed values of that variable) and $C = \{C_1,C_2,... C_k\}$ is a *set of constraints*. Constraints are *relations* between domains of variables, i.e. they are subsets of the Cartesian product of the affected variables' domains ($C_i \subset D^* = D_p \times D_q \times ... \times D_z$).

A *solution* of a CSP is a vector $x = [x_1,x_2,... x_n]$ of values that satisfies all the constraints. The *constraint satisfaction problem* itself is to find *one solution* or *all solutions* of a given CSP.

CSPs can be represented by a G(X,C) network where the elements of X are represented by *nodes* and the elements of C by *edges* of the network. In so-called *binary* CSPs every constraint affect at most two variables and the network is a graph; in the general case, however, the CSP network is a hypergraph. *Loop edges* represent unary constraints (affecting only one variable), *multiple edges* are different constraints affecting the same variables. Obviously, loop and multiple edges can be eliminated from binary CSPs.

A CSP is *discrete* if every D_i is a finite set, and *continuous* if some D_i-s are finite or infinite intervals. The CSP is *static* if both the constraint network topology and the constraints themselves are fixed and *dynamic* if they can change during the search for solutions. In the field of system-level diagnosis, only discrete CSPs are used.

Solving discrete CSPs is proved to be NP-complete [4], so simple exhaustive algorithms cannot be used to generate all the variable value sets and to select the solutions. Intelligent backtracking algorithms (backjumping, backmarking, forward checking etc.) can be used [7].

2.2 Solving CSPs: Preprocessing Methods

We can assume for simplicity without loss of generality that each of the n variable in the CSP has the same discrete domain so the search space $D^* = D^n$. Therefore the worst-case complexity of a trivial generate-and-test algorithm is $O(d^n)$ where d is the cardinality of D. The complexity can be obviously reduced by decreasing d or n.

Decreasing of d can be achieved by a kind of preprocessing called *consistency algorithms* [4] [5] [6]. Consistency means the elimination of locally inconsistent value combinations from the variables' domains, as they surely cannot participate in a globally consistent (correct) solution.

Consistency algorithms even reduce the number of "fruitless" backtrackings made every time when a locally inconsistent value is found. They work generally only on binary CSPs because every variable in such CSPs can be evaluated independently. Moreover a subsequent

evaluation of the current value of a variable and its neighbours is always sufficient to achieve global consistency. Therefore problem transformation to a binary CSP is preferable.

Consistency algorithms can be grouped according to the number of the nodes (vertices) they consider when searching for local inconsistencies.

2.2.1 Node-consistency or 1-consistency

Node consistency considers only a single vertex at a time; it simply checks unary constraints and deletes all values not allowed by them. As unary constraints can be previously eliminated from a CSP by restricting the domains, this algorithm is used only as a supplementary step in more complex consistency algorithms.

2.2.2 Arc-consistency or 2-consistency

Arc consistency considers two vertices X_i and X_j at a time, connected by a binary relation R_{ij}. It eliminates every value from the domain of X_i without a value of X_j satisfying R_{ij}. By checking apprppriate vertex pairs and relations, full consistency can be achieved.

There are three basic versions of general purpose arc consistency algorithms (in the order of decreasing worst case time complexity):

- AC-1 updates all the variables whenever any of the variable domains has changed;
- AC-3 updates the domains of the variables adjacent to the changed variable;
- AC-4 updates only those adjacent variables that are affected by the change of a variable domain. (It requires some bookkeeping of the relations and the variable domains affected by them.) [3] [4]

2.2.3 Path consistency or 3-consistency

Path consistency between two vertices X_i and X_j connected by a binary relation R_{ij} means that all value pairs in a solution of the CSP allowed by R_{ij} must be also allowed by all paths between X_i and X_j. The whole constraint network is path consistent if every pair of directly connected vertices is path consistent.

Full path consistency in a complete constraint graph is equal to the consistency for length 2 paths [3]. Since any constraint network can be extended to a full constraint graph with dummy (“always true”) constraints, checking path consistency is equal to checking only length 2 paths.

There are also three basic versions of path consistency algorithms; differences among them are similar to the differences among arc consistency algorithms:

- PC-1 updates domains of every vertex, vertices along every arc and every length 2 path if any vertex has changed;
- PC-2 updates domains of those length 2 paths that contain the changed vertex;
- PC-3 updates only the length 2 path affected by the changes of a vertex domain. It uses similar bookkeeping about the influence of variables and edges like AC-4. [4]

2.2.4 k-consistency

A set S_k of k variables is considered at a time. If a completely consistent subset of value (k-1)-tuples exist on $S_{k-1} \subset S_k$ (with k-1 variables) then any value from the domain of the kth variable can be eliminated that cannot form a consistent value set with any of the consistent (k-1)-tuples. Global consistency can be achieved by successive elimination for increasing values of k until all variables are involved or the some of the domains become empty; in this case

the CSP is not satisfiable. The most well-known k-consistency algorithm is the *invasion* procedure [3].

2.3 Formulating a self-diagnosis problem as a CSP

There are many similarities between methods of self-diagnosis and constraint satisfaction. Actually the final goal is very similar: we want to know the fault state of the system components that conforms to our diagnostic model, the test invalidation rules and the actual test results (syndrome pieces). These restrictions can be represented by binary relations between the state of processors in a test pair. The exact relation is determined by the test result, thus a set of relations can be built from the syndrome information and can be applied to find the possible fault states of the system.

The use of *relations* instead of *logical functions* is advantageous, because the diagnostic uncertainy appearing in some test invalidation (e.g. in the PMC model a faulty unit may be tested as good) can be handled as well. The relations can be handled by a uniform mechanism, independently from the invalidation rules, system topology, the considered number of faults and special properties of syndromes. So this representation is very flexible and is applicable on a wide range of systems.

Therefore a self-diagnosis problem can be very easily reformulated to a constraint satisfaction problem. The variables of the CSP represent the fault state of the system components. The constraints represent the restrictions from the model by the test invalidation relations and by the actual syndrome pieces. If one-pass diagnosis is allowable, a static binary CSP is produced. In the case of diagnosis on the fly only a few syndrome pieces are present so the complete set of relations cannot be built at the beginning. Every incoming test result, however, reduces the solution space of possible fault states so the previously constructed relations (constraints) remain valid, just new constraints have to be added. Therefore a kind of dynamic CSP can represent this case.

This reformulation gives a way to handle self-diagnosis problems very comfortably, with the well-elaborated toolset of CSP solution methods. With a sufficient diagnostic model, a very flexible method can be constructed.

The constraint-based approach is also very similar to the approach of the Selényi algorithm [1], whose syndrome decoding process consists of two phases. In the first phase all the deterministic information is extracted from the syndrome. This information contains all possible combinations of the fault states (CSP solution also produces this). In the second phase those units are identified that remained unclassified in the first phase if one-step diagnosis is requested. This means excluding the unwanted solutions from the set of all possible solutions given in the first phase. It requires, however further restrictions in the diagnostic model (assumptions on maximal number of faults, exclusion of certain faults, etc.).

3. Diagnostic model of the implemented algorithm

3.1 Implementation environment

The experimental implementation of the CSP-based diagnosis algorithm was created on a Parsytec GCel massively parallel reliable multiprocessor machine (**Fig.1.**). The computing elements are Inmos T805 or T9000 transputers. They are grouped by 16 to clusters; these clusters are the basic building blocks of a machine that is scalable up to 16384 transputers.

Each transputer has 4 physical data links. These are connected to C104 routing chips that provide a very fast, reliable and deadlock-free message routing and connection management.

Each cluster has 4 routing chips; every C104 chip has 32 connection ports (16 for the internal interconnection of the cluster, 12 for the connection between clusters and 4 for I/O control and other purposes).

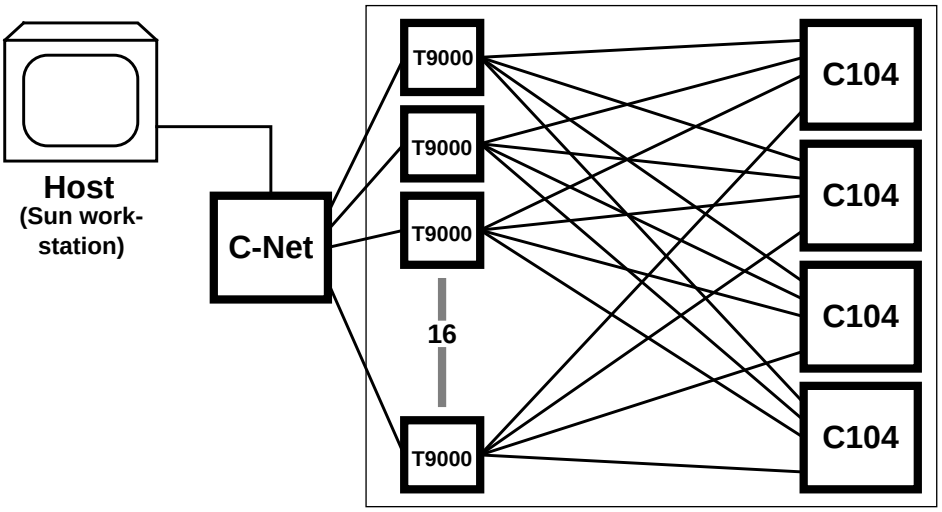

Figure 1. The structural layout of the Parsytec GCel (1 cluster)

Despite of the 4 physical data connections, each transputer can communicate with an arbitrary number of other transputers via so-called virtual links. These are managed by a special unit of the T9000 by multiplexing data packets on the physical connections. The highly configurable routing chips make allocation of virtual links very easy; complete virtual topologies are supplied with development libraries. The physical topology of each cluster is a 4x4 two-dimensional mesh.

The machine has even low-level fault-tolerant features: every cluster has a 17th spare processor, the local memory of the transputers is fault-tolerant with an ECC device.

Peripherial I/O management is done by a separate host machine (e.g. a Sun workstation) connected to the Parsytec.

The machine has a Control Network (C-Net): every transputer has a direct link to a special group of transputers directly connected to the host machine. This separate group is used for dynamic configuration management and job control.

(Due to the delayed development of the T9000 transputers, the actual Parsytec machine was equipped with T805 transputers. These transputers have a slightly different hardware architecture but the development system is claimed 100% source code compatible with future T9000-based Parsytec machines.)

3.2 The fault model

In order to validate the concepts described above a simple fault model was developed for the Parsytec machine and a syndrome decoding algorithm was implemented using the standard test procedures available.

The used fault model, additionally to the processor faults, includes the faults in interprocessor links and routing chips as well.

Testing of these system components is done by mutual time-out protected periodical <I'm alive> messages between neighboring processors. The asynchronous communication mode is used for message exchange because it does not block the sender processor (time-out detection is possible).

The following fault states are considered for the components:

Table 1:

Unit	Fault state and its notation		Behavior	Possibly faulty component(s)
Processor	fault-free	0p	correct operation	-
	faulty	1_p	incorrect test result evaluation	memory
	dead	c_p	no communication	CPU configuration, virtual link, C-Network, hardware exceptions
Data link	live	$L_{p,R}$	correct message transfer	-
	broken	$\overline{L}_{p,R}$	no message transfer	wires/connectors, CPU data link circuit
Router	fault-free	-	correct operation	-
	single port fault	$L_{R,p}$	no message transfer via the faulty port	router data port circuit
	dead	m_R	all ports are faulty	internal routing scheme, clock

The possible test results are: *good* (the <I'm alive> message was correctly received within the time-out limit), *faulty* (the <I'm alive> message was received within the time-out limit but was corrupted) or *dead* (no message was received).

The effective diagnosis algorithm is running on the host machine (centralized diagnosis is assumed). It is possible because the processors have a separate data connection to the host machine (via the C-Net) that is independent from the routing chips and can be considered fault-free.

The developed algorithm is for *intra-cluster* diagnosis (we assume only a single routing chip between 2 processors) but it can be easily extended hierarchically to diagnose the whole Parsytec machine.

3.3 The test invalidation scheme and implication rules

The PMC type (symmetric) test invalidation was used for the algorithm. It was mandatory due to the testing with <I'm alive> messages; other, more sophisticated test methods make more optimistic invalidation possible.

Syndrome decoding is driven by implication rules, represented by constraints. They originate from the system structure (fault domination rules), the test invalidation model and the actual syndrome pieces (**Fig.2**.).

All the constraints are binary to achieve greater simplicity: the test results (syndrome bits) are eliminated from them as variables, only the fault state of the tester and the tested compo-

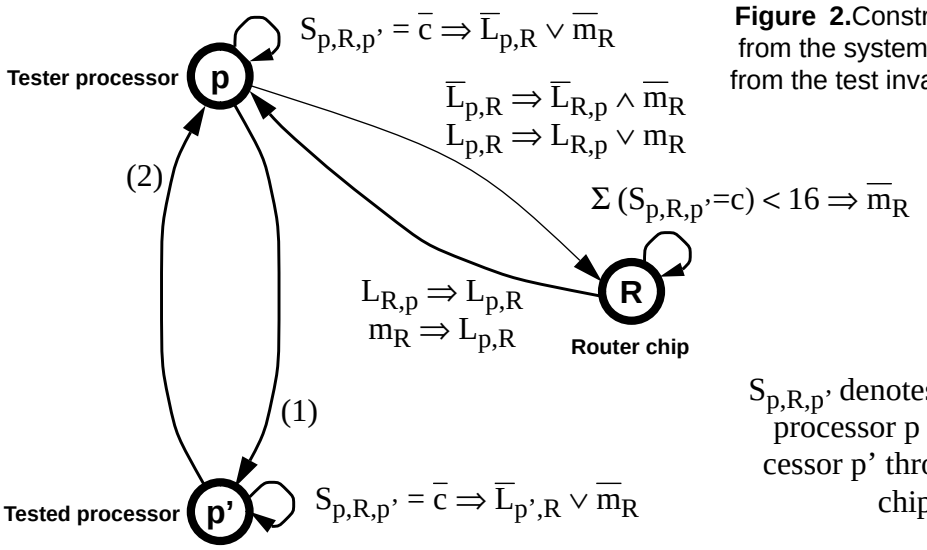

Figure 2. Constraints resulting from the system structure and from the test invalidation model

$S_{p,R,p'}$ denotes the result of processor p testing processor p' through routing chip R.

nent are variables. However, test results are already known before the syndrome decoding starts so they can be treated as constants.

The constraints from the implication rules are as follows:

- (1) Forward (implication from the state of the tester to the state of the tested)
  - $S_{p,R,p'} = 0 \wedge 0_p \Rightarrow 0_{p'}$ (if the tester processor is fault-free and the test result is "good" then the tested processor is also fault-free);
  - $S_{p,R,p'} = 1 \wedge 0_p \Rightarrow 1_{p'}$;
  - $S_{p,R,p'} = c \wedge 0_p \Rightarrow L_{p,R} \vee m_R \vee L_{p',R} \vee c_{p'}$.
- (2) Backward (from the state of the tested to the state of the tester)
  - $S_{p,R,p'} = 0 \wedge 1_{p'} \Rightarrow 1_p$ (if a faulty unit is tested as good then the tester is faulty);
  - $S_{p,R,p'} = 1 \wedge 0_{p'} \Rightarrow 1_p$;
  - $S_{p,R,p'} = c \wedge \bar{c}_{p'} \Rightarrow L_{p,R} \vee m_R \vee L_{p',R} \vee \bar{0}_p$.

4. Implementation of the algorithm

The self-diagnosis algorithm is implemented in two parts: the low-level tester mechanism (sending and receiving the <I'm alive> messages) runs on the Parsytec machine, the effective diagnosis program runs on the host machine .

The test process is controlled by the host: it initiates the test sequence (starts the Parsytec processors to exchange <I'm alive> messages), collects the results from the processors, maintains the dynamic CSP data structure, runs the CSP solver algorithm and displays the results. The two program parts communicate through sockets.

4.1 The CSP solver

The CSP solver engine of the diagnosis algorithm is based on a public domain universal CSP solver library from the University of Atlanta [7]. Different backtracking and preprocessing (consistency) algorithms are implemented and can be easily applied.

The solver is able to maintain certain dynamic CSPs: the constraints themselves can be altered during the solution process but their number cannot. Initially only the "fixed", syndrome-independent constraints are generated, the others are "always-true" (the solver always works with complete constraint graphs so the unnecessary constraints have to be "always-true").

There are 20 variables in the CSP model; they represent the fault states of the processors with their data connections and the routing chips. The cardinality of the variable domains is 48. (A processor can be fault-free, faulty or dead and each of its 4 data connections can be live or broken, it gives $3x2^4=48$ possible states. The routing chips have 20 possible states but the solver requires equal domain sizes.)

Many sophisticated enhancements assure a maximal efficiency of the CSP solution. Only those variables/processors are considered that we have information about, so we get results only from the necessary units. The untested data links of the processors are assumed live until exact information is received about them. The indistinguishable error classes[1] are unified (only one of them appears in the results). This modifications decrease the number of variables

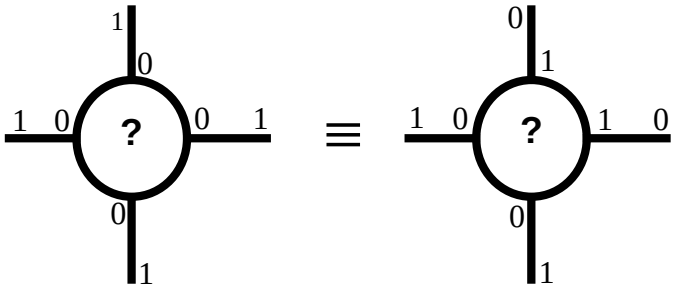

Figure 3. Example for indistinguishable error classes

processed and the number of value combinations checked and assure that only the valuable results are supplied.

Moreover, further considerations can be adapted to the CSP solver very easily. For example if we consider a limited number of faulty units, the CSP solver can check whether this consideration holds and even automatically increases the error limit. This possibility makes the system extremely fast with a few errors and still usable and fast with more errors than the limit.

5.Results of a test run

The CSP-based diagnosis algorithm was tested with a logical fault injector: the host machine generated a random fault pattern for the Parsytec processors and downloaded it with the testing initialization messages. The low-level testing mechanism on the Parsytec processors interpreted the fault pattern and acted according to the fault state: “fault-free” processors tested their neighbors and sent the results back to the host, “faulty” processors did the testing but reported a random result and “dead” processors did nothing. This construction was necessary because no physical fault injection was available for the Parsytec machines equipped with T805 transputers and the fault injector developed for T9000 was unusable due to the hardware structure difference.

Figure 4. shows the results of a typical test run. In this case the fault pattern contained a single faulty processor. The upper curves display the number of the solutions found by the CSP solver and the number of processors that have already sent some test results and the lower ones display the number of consistency checks made as a measure for the computational efficiency.

---

1. These error classes are resulted by fault domination; e.g. a CPU data link circuit fault, a mechanical contact fault in the link wires or a routing chip data port fault gives the same test result.

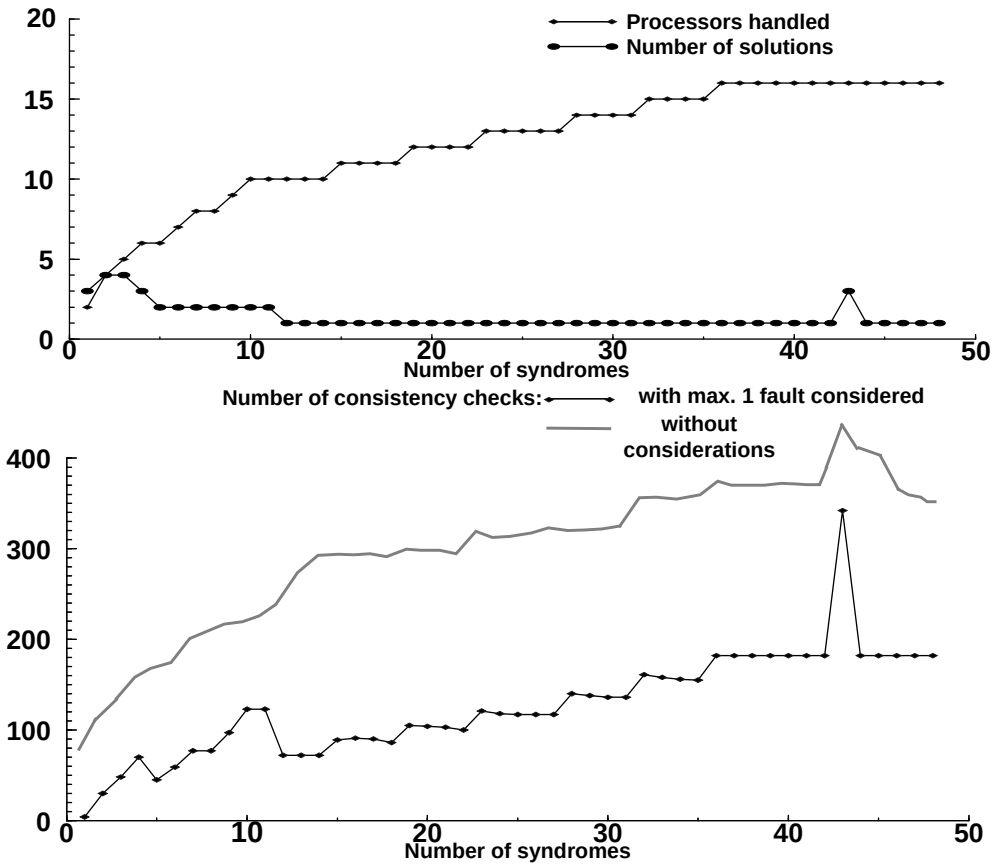

Figure 4. Test results of the CSP diagnosis algorithm

6.Conclusions

The CSP-based syndrome decoding algorithm has proved its proper operation during the tests and the correctness of the concepts behind it. Additional tests showed that the constraint solver is up to five times faster than an exhaustive search (the average processing time of a test result was 88 µs vs.448 µs in a test series). However, he applied CSP solving algorithm (graph-based backjumping [7]) has not yet theoretically proved to be optimal for this application; further work is needed to find the most efficient strategy for the solver.

References

[1]A. Pataricza, K. Tilly, E. Selényi, M. Dal Cin: *A Constraint Based Approach to System Level Diagnosis*

[2]E. Selényi: *System Level Fault Diagnosis in Multiprocessor Systems with a General test-invalidation Model*

[3]K. Tilly: *Constraint Based Logic Test Generation* (Ph.D. Thesis)

[4]U. Montanari: *Networks of Constraints: Fundamental Properties and Applications to Picture Processing*, Information Sciences vol. 7, 1974, pp. 95-132.

[5]R. Mohr, T. C. Henderson: *Arc and Path Consistency Revisited*, Artificial Intelligence, vol. 28 (1986), pp. 225-233.

[6]A. Mackworth, E. C. Freuder: *The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems*, Artificial Intelligence, vol. 25 (1985), pp. 65-74.

[7] Peter van Beek: *A Binary CSP Solution Library* (Available by FTP from ftp.cs.ualberta.ca)