Constraint Based System-Level Diagnosis of Multiprocessors

J. Altmann[1], T. Bartha[2], A. Pataricza[2], A. Petri[2] and P. Urbán[2]

[1] University of Erlangen, Dept. of Computer Science III, Martensstr. 3, 91058 Erlangen, Germany
[2] Technical University of Budapest, Dept. of Measurement and Instrument Eng., Műegyetem rkp. 9, H-1521 Budapest, Hungary

Abstract. The paper presents a novel modelling technique for system-level fault diagnosis in massive parallel multiprocessors, based on a re-formulation of the problem of syndrome decoding to a constraint satisfaction problem (CSP). The CSP based approach is able to handle detailed and inhomogeneous functional fault models to a similar depth as the Russel-Kime model [18]. Multiple-valued logic is used to deal with system components having multiple fault modes. The granularity of the models can be adjusted to the target diagnostic resolution without altering the methodology. Two algorithms for the Parsytec GCel massively parallel system are used as illustration in the paper: the centralized method uses a detailed system model, and provides a fine-granular diagnostic image for off-line evaluation. The distributed method makes fast decisions for reconfiguration control, using a simplified model.

1 Introduction

The large number of components built into *massively parallel multiprocessor systems* increases the probability of a component faults. Since reliable operation over a long time period is also necessary for complex computations, the system must be able to mask the effect of occurring errors by *fault tolerance*. The underlying diagnostic principle is generally *system-level diagnosis*, followed by reconfiguration and recovery in case of a detected fault.

Different models and algorithms were developed for system-level diagnosis typically originating in the first graph theory based “system-level models” (PMC for symmetric and BGM for asymmetric test invalidation) published in the late-sixties. Their mathematical apparatus is simple and well-elaborated and practical implementations proved their usefulness as well. However, the implicit limitations — for instance the oversimplification of the test invalidation mechanism in order to assure a proper mathematical treatment — decrease the level of reality in the models. Moreover, the rapid development of electronic technology and computer architectures radically modified the basic assumptions used in diagnostic models [16]:

- fault rates are in general low and the dominating part of faults is *transient*;
- the complexity of additional components of the system (interface and communication circuits) is comparable with the that of the processing elements;

- the number of the components in the systems is drastically increased.

The majority of insufficiencies result from the hardest simplification of the PMC-type models: the assumption of a homogenous system and test structure (identical components with the same test invalidation over the entire system). This reduces their applicability due to the increasing practical importance of inhomogeneous systems.

1.1 Required Features of a New Diagnosis Method

The new requirements involved by the latest results in multiprocessor system design characterize the expected features of a general purpose self-diagnosis method:

- it should be applicable to inhomogeneous systems as well as to homogenous ones (components with different test invalidation models are to be considered);
- neither the actual system topology nor the test invalidation model should limit the diagnostic resolution (current methods use rigid, inadaptive algorithms seriously restricting the target system features);
- the algorithm should extract all of the useful information from the elementary diagnostic results (e.g. for estimating the level of diagnosis at run-time);
- it should be able to work in massively parallel computers 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;
- many applications demand “on-the-fly” diagnosis for a maximal performance, able to identify the fault states of certain units even from partial syndrome information (i.e. before receiving all of the test results).

These requirements necessitate a new approach. A generalized test invalidation model and syndrome decoding algorithm for inhomogeneous systems is published in [1]. However, the efficiency of the algorithm becomes to a crucial factor in case of large-scale systems due to the employed mathematical apparatus — operations on matrices of the dimension of the number of processor in the system.

Syndrome decoding is the most important step in diagnosis, determining the actual fault states of system components from the syndrome. This systematic search is in general NP-complete. The main intention of “artificial intelligence” (AI) methods is to find efficient solutions for difficult to solve problems. A group of them, the CSP (Constraint Satisfaction Problem) solving methods seem especially useful for system-level diagnosis [2].

2 System-level Diagnosis and CSP

2.1 Definition of the CSP

Constraint satisfaction problems (CSP) deal with the estimation of a single or all consistent solutions in large-scale relation systems. More formally, a CSP is a *(X, D, C)* tuple, where $X = \{X_1, X_2, \ldots, X_n\}$ is a *set of variables*, defined over the set $D = \{D_1, D_2, \ldots, D_n\}$ *domains*, and $C = \{C_1, C_2, \ldots, C_k\}$ is a *set of constraints*. Con-

straints are *relations* between the variables, that is they are subsets of the Cartesian product of the corresponding variables' domains ($C_i \subseteq D^* = D_p \times D_q \times \ldots \times D_z$). A *solution* of a CSP is a vector $x = [x_1, x_2, \ldots, x_n]$ of values that satisfies all the constraints.

The structure of CSPs can be represented by a $G = (X, C)$ hypergraph where the variables are represented by *nodes* and the relations defined between them by *edges* of the network. *Binary* CSPs constitute a special subclass of CSPs, where each constraint affects at most two variables and the network becomes to a simple graph. *Loop edges* represent unary constraints (affecting only a single variable), *multiple parallel edges* are different constraints affecting the same variables.

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.

2.2 CSPs solution methods

Solving discrete CSPs is proved to be NP-complete [10], so exhaustive algorithms are impractical for large-scale systems. Intelligent search algorithms (using backjumping, conflict based backtracking, forward checking, etc.) [14] offer a better average time complexity, yet their worst-case complexity is still exponential.

Let us assume for simplicity, that each of the n variables in the CSP has a discrete domain of the same cardinality d, so the search space is $D^* = D^n$. The worst-case complexity of a simple exhaustive search is $O(d^n)$. This complexity can be reduced only by decreasing d. (A decrease in n would imply that the CSP contains redundant variables, i.e. an improperly formulated CSP). Decreasing of d can be achieved by preprocessing the CSP problem prior to the solution procedure. The so-called *consistency algorithms* [10, 11, 12] exclude locally inconsistent value combinations from the domains of variables, since these values surely cannot appear in a globally consistent solution. These methods work generally only on binary CSPs, because every variable can be evaluated independently in this subclass of CSPs. Moreover, such an evaluation of the variables guarantees the global consistency as well.

Consistency algorithms can be grouped according to the number of the nodes they consider during an elementary step, while searching for local inconsistencies:

- **Node-consistency or 1-consistency** considers only a single node at a time. It simply checks unary constraints, and deletes all values not complying to them. As unary constraints can be previously eliminated by restricting the domains, this algorithm is used only as a supplementary step in more complex algorithms.
- **Arc-consistency or 2-consistency** considers at a time two variables X_i and X_j and a binary relation R_{ij} between them. Every value is excluded from the domain of X_i without a value of X_j satisfying R_{ij}. Full consistency can be achieved by checking appropriate vertex pairs and relations. There are three basic versions of general purpose arc consistency algorithms (enlisted in the order of decreasing worst case time complexity):
  - **AC-1** updates all the variables whenever any of the variable domains has changed. Its time complexity is $O(d^3nc)$, where c is the number of constraints;

- **AC-3** updates only the domains of the variables adjacent to the changed variable. Its complexity is $O(d^2n)$;
- **AC-4** updates only those adjacent variables which are affected by the change of a variable domain. It reaches the proven optimal complexity of $O(d^2c)$, at the price of some bookkeeping of the relations and the variable domains [10].

- **Path consistency or 3-consistency** algorithms check the transitivity of the consistency of a candidate value assignment, thus path consistency between two variables X_i and X_j connected by a binary relation R_{ij} means that all value pairs in a solution allowed by R_{ij} must be also allowed by *all* paths between X_i and X_j. The entire constraint network is path consistent if every vertex pair is path consistent. Full path consistency in a complete constraint graph is equal to the consistency for length 2 paths. Checking path consistency requires only the checking of all length 2 paths, since any constraint network can be virtually extended to a complete constraint graph by inserting dummy ("always true") constraints.
  There are also three basic versions of path consistency algorithms similarly to arc consistency algorithms:
  - **PC-1** updates domains of every vertex, vertices along every arc and every length 2 path if any vertex has changed. Its time complexity is $O(d^5n^5)$;
  - **PC-2** updates domains of those length 2 paths that contain the changed vertex. Its complexity is $O(d^5n^3)$;
  - **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 [10]. It is also proven to be optimal, its complexity is $O(d^3n^3)$.
- ***k*-consistency** examines 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 which cannot form a consistent value set with any one 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 a domain becomes empty indicating the insatisfiability of the CSP. The most widely used k-consistency algorithm, the *invasion procedure* [9] has a time complexity of $O(cd^{f+1})$, where f is the maximal number of new nodes found traversing the graph.

2.3 Formulating the Diagnosis Problem as a CSP

The ultimate goal of syndrome decoding and CSP is very similar: the algorithm must classify the fault state of system components in a consistent way that conforms to the given diagnostic model, test invalidation rules and the actual outcomes of the elementary tests (syndrome elements). These restrictions can be represented as binary relations between the fault state of the tester and the tested units. Note that the use of *relations* instead of *logical functions* supports the handling of diagnostic uncertainty appearing in some test invalidation models (e.g. in symmetric invalidation the outcome of a test executed by a faulty tester is non-deterministic).

A diagnosis problem can be very easily reformulated to a constraint satisfaction problem. The *variables* of the CSP represent the *fault states* of the system components.

The *constraints* correspond to the restrictions derived from the *test invalidation model* and the current *syndrome elements*, thus the test invalidation rules determine a set of candidate relations, from which the actually used one has to be selected, when knowing the actual test outcome. For instance, the relation over the tester-tested unit fault state value pair consists in the case of PMC test invalidation {(0,0), (1,0), (1,1)} of the outcome is 0, and {(0,1), (1,0), (1,1)} if the outcome is 1, respectively.

If *one-pass diagnosis* is required, simply a static binary CSP is produced after performing all tests. However, in the case of *on-the-fly* diagnosis, newly generated test results must be processed immediately. Only a few syndrome elements are present at the beginning, so the complete set of relations cannot be created at once. Every incoming test result adds new relations to this set, while the previously constructed ones remain still valid. Thus the solution space is gradually reduced, in other words a *monotone* dynamic CSP is produced.

The resulting constraint network is a "clear" mathematical structure, details such as the applied fault model or the system topology appear only implicitly in the relations and the variable domains. This representation is extremely flexible and applicable to a wide range of systems. Moreover, a well-elaborated toolset of CSP solution methods is available, so the diagnosis problem can be solved computationally efficiently.

If not all the components does have a sure classification after the transitive closure, the remaining ones are classified using further restrictions on the fault model (limitation on maximal number of faults, exclusion of certain faults, etc.).

3 The Modelled System

The diagnosis model of the algorithms presented in this paper were derived from the Parsytec GCel massively parallel reliable multiprocessor (Fig. 1).

The processing elements are Inmos T805 transputers. Sixteen transputers are grouped together in one *cluster*, forming the basic building block of a machine. Each transputer has 4 physical data links connected to C004 link switch chips. Each cluster incorporates 4 link switch chips each of them having 32 connection ports, so inside a cluster arbitrary interconnection topology can be realized.

Four clusters constitute a so-called *GigaCube*. The topology of the clusters is a static 2×2 two-dimensional grid. A GigaCube forms a physical unit: it has its own temperature and voltage monitoring facilities, and a separate transputer working as a *control node*. GigaCubes can be arranged in a $4 \times 4 \times 8$ spatial array, so in its full configuration the system is scalable up to 16,384 transputers.

As a default, transputers are connected in a two-dimensional grid topology. Yet, despite the only 4 physical data connections each transputer can communicate with an arbitrary number of other transputers via *virtual links*.

The machine has built-in fault-tolerant mechanisms. In every cluster, there is an additional spare processor, replacing the faulty one in case of a detected error. The local memory of the transputers is ECC-protected, and soft memory errors are eliminated by memory scrubbing. These actions, as well as booting, job control and

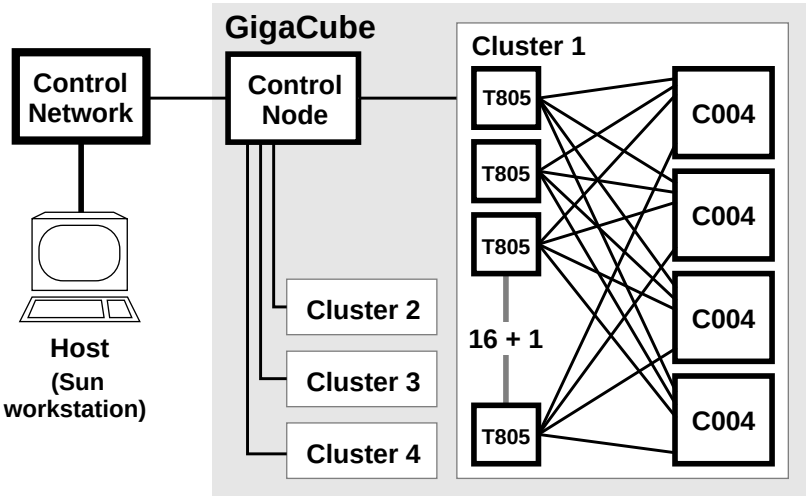

Fig. 1. The structural layout of the Parsytec GCel

dynamic configuration management are supervised by control nodes. They communicate over a separate interconnection network called the *Control Network* (C-Net).

Peripheral I/O management is done by a stand-alone host computer (usually a Sun workstation) connected to the Parsytec GCel machine.

4 Fault Model

Different diagnostic goals require this complex multiprocessor to be modelled at separate abstraction levels. A centralized diagnostic algorithm is intended for off-line monitoring of the system. Diagnosis is made at the replacement unit level, hence the fault model should provide a detailed view about the multiprocessor. The model can be rather complex, as no tight time limits are imposed on the algorithm.

On the other hand, the system should take reconfiguration measures quickly in the case of faults, in order to minimize the effects of *fault propagation* and the error-related loss of performance. The efficiency of the diagnosis algorithm is crucial, necessitating a simple fault model, supporting a diagnostic resolution only down to the reconfiguration unit level. Also, the diagnosis algorithm should avoid using any central resources to maintain scalability and hence be executed locally at every processing element (i.e. distributed diagnosis is needed).

These fault models can be constructed by a *stepwise model refinement*; gradually refining a basic initial model to provide a more sophisticated view of the system.

4.1 Centralized Diagnosis

The modelled system is a simplified version of the Parsytec GCel massively parallel multiprocessor (see Fig. 2) consisting of processing elements and communication links in the form of a two-dimensional grid. Each processor mutually tests all of its neighbors. Test results are sent to the host computer performing syndrome decoding.

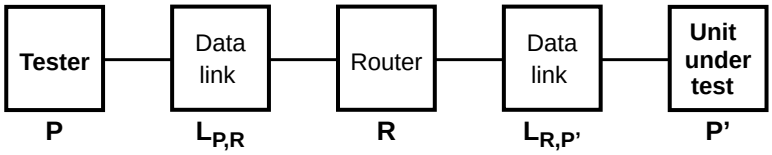

Fig. 2. Components involved in a test

For simplicity, host—transputer communication facilities are assumed to be reliable in the model. The used fault model (Table 1) includes the faults in inter-processor links and routing chips as well, additionally to the processor faults [4].

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

Table 1: Component fault states

Testing of system components is done by time-out protected mutual <I'm alive> messages, sent periodically between neighboring processors. Asynchronous transfer mode is used for message exchange, because it does not block the sender processor. The possible test results are:

- **good** (0) if a correct <I'm alive> message was received within the time limit;
- **faulty** (1) if a corrupted <I'm alive> message was received;
- **dead** (c) if no message was received within the predefined time limit.

An extended PMC-like test invalidation [17] is used (Table 2) to describe the fault state–test result relationships.

Diagnosis	Tester	Data links	Router	UUT	Test result
centralized	0_P	$\overline{L}_{P,R} \wedge \overline{L}_{R,P}$,	0_R	0_P,	0
				1_P,	1
				c_P,	c
		$L_{P,R} \vee L_{R,P}$,	X	X	c
		X	$c_R \vee 1_{P,R} \vee 1_{R,P}$,	X	c
	1_P	X	X	X	1
	c_P	X	X	X	c
distributed	0_P	$\overline{L}_{P,R} \wedge \overline{L}_{R,P}$,		0_P,	0
				1_P,	1
		$L_{P,R} \vee L_{R,P}$,		X	1
	1_P	X		X	1

Table 2: Test invalidation model used in diagnosis (X indicates a *don't care* value)

4.2 Distributed Diagnosis

Diagnosis is made in the distributed case on each individual transputer of the multiprocessor. Processors broadcast test results across the network after performing the tests. At the end, every processor receives every *syndrome message*. The dynamic specification of the broadcasting mechanism (e.g. an upper limit on the communication delays) assures that after a specific time limit there are no pending messages [7].

Only syndrome messages containing a "faulty" test result will be sent by the testers to their neighbors and no message transfer is invoked in the "fault-free" case. This is a significant reduction in the number of messages, as typically only a few faults occur.

Broadcast messages require the interaction of intermediate processors, due to the limited neighborhood of the message source processor. Syndrome messages from other processors will be potentially altered or lost by a faulty processor during this broadcast. Corrupted messages can be detected by a simple error detecting protocol; altered messages are ignored[1]. Thus the diagnosing program assumes that no undetected changes will occur.

The other type of syndrome losses cannot be detected in such a simple way, so the diagnosing processor will never or only occasionally receive fault reports from unfor-

1. Theoretically, an altered message indicates a fault in the message forwarding chain of elements, but in order to avoid cross-interference between test and syndrome decoding time faults, we neglect corrupted messages.

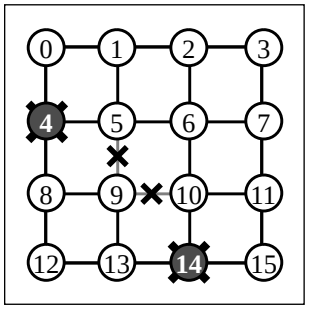

Fig. 3. The system is cut into two isolated parts
(Processor 0 never receives messages from Processor 12)

tunately placed – good or faulty – processors [6]. (Fig. 3 is an example for such a situation).

No routers are considered in the fault model, as router faults are assumed to turn reconfiguration within a processor cluster impossible so faulty routers must be handled by reconfiguration at cluster level. Also, no distinction between "faulty" and "dead" fault states and test outcomes is made for simplicity. The lower part of Table 2 describes the simplifications to the fault model in the centralized approach.

5 Implementation

5.1 Centralized Diagnosis

5.1.1 Transformation of the Implication Rules into Constraints

Syndrome decoding is driven by implication rules, represented by constraints. They originate from the system structure, the test invalidation model and the actual test outcomes. All the constraints are binary over the fault state of the tester and the tested component, to achieve a greater simplicity: the test results (syndrome bits) are eliminated from them as variables after receiving the test outcome [3].
The constraints originated from the implication rules are the following ($\mathbf{S_{P,R,P'}}$ denotes the result of the test made by **P** on **P'** via router **R**):

- *Forward implication* (from the state of the tester to the state of the tested)
  - $\mathbf{S_{P,R,P'} = 0 \wedge 0_P \Rightarrow 0_{P'}}$ (if the tester processor is fault-free and the test result is "good" then the tested processor is also fault-free);
  - $\mathbf{S_{P,R,P'} = 1 \wedge 0_P \Rightarrow 1_{P'}}$ (if the tester is fault-free and the test is "faulty" then the tested unit is faulty);
  - $\mathbf{S_{P,R,P'} = c \wedge 0_P \Rightarrow L_{P,R} \vee c_R \vee L_{P',R} \vee c_{P'}}$ (if the tester is fault-free and the test is "dead" then the links between them are broken, the router involved is dead or the tested unit is dead).

- *Backward implication* (from the state of the tested to the state of the tester)
  - $\mathbf{S_{P,R,P'} = 0 \wedge 1_{P'} \Rightarrow 1_P}$ (if a faulty unit is tested as good then the tester is faulty);
  - $\mathbf{S_{P,R,P'} = 1 \wedge 0_{P'} \Rightarrow 1_P}$ (if a good unit is tested as faulty then the tester is faulty);
  - $\mathbf{S_{P,R,P'} = c \wedge \overline{c}_{P'} \Rightarrow L_{P,R} \vee c_R \vee L_{P',R} \vee \overline{0}_P}$ (if a unit is tested as dead and it is not dead then the links are broken, the router is dead or the tester is not fault-free).

5.1.2 One Pass and "On-the-Fly" Diagnosis

The diagnostic algorithm has the ability to create diagnosis "on-the-fly", using only partial syndrome information. In this case the diagnostic part is invoked several times at the arrival of partial test results. Implications from new syndrome elements are added to this dynamic constraint network during each call and a path-consistency based preprocessing of the network is performed, successively restricting the solution domain. If few syndromes are available, the solution process is expected to be fast (see Section [7]). Table 3 summarizes this process**:**

`initialization`
`process the list of already known faults`
`while` `there are syndrome messages` **`do`**
`read some syndrome messages`
`process the list of syndrome messages`
`preprocess the constraint network`
`solve the constraint network`
`generate output from the solution set`

Table 3: "On-the-fly" diagnosis

5.2 Distributed Diagnosis

5.2.1 Modelling the Constraints

First of all, implication rules should be created on the basis of the system structure, the applied test invalidation model and the actual syndrome elements [5]. These implication rules can then be represented by constraints. Moreover, some syndrome-independent constraints can be generated before receiving any fault reports:

- Known faulty components **C** of the system — for instance those on the list of a priori known faults obtained by the power-on self-test — are predefined in this model as faulty ($\mathbf{1_C}$).
- The diagnosing processor **P** is fault-free ($\mathbf{0_P}$) in his own local diagnostic image; this assumption is justified by the fact that a diagnosis produced by a faulty processor is worthless anyway.

- The processors on the cluster boundary have unconnected links; these are considered by the algorithm as virtually existing, always fault-free links ($\overline{\mathbf{L}}_{\mathbf{P,0}}$) in order to keep a total symmetry of the structure.
- If a processor is faulty, the fault states of its links cannot be determined in the applied fault model, i.e. processor faults dominate the faults of the associated links. Links are assumed to be fault-free to reduce the number of solutions and speed up the algorithm:

$$\text{Links } \mathbf{L_i} \text{ are connected to } \mathbf{P}\text{: } \mathbf{1_P} \Rightarrow \overline{\mathbf{L}}_{\mathbf{P,1}} \wedge \overline{\mathbf{L}}_{\mathbf{P,2}} \wedge \overline{\mathbf{L}}_{\mathbf{P,3}} \wedge \overline{\mathbf{L}}_{\mathbf{P,4}} \,.$$

The implication rules depend on the actual test results, so they can be produced only at "run-time". In the following paragraphs **P** stands for the processor performing the test**,** **P'** for the processor under test and **L** for the link connecting them.

Test result	Implication rules	Constraint(s)
0	$\mathbf{0_P} \Rightarrow \overline{\mathbf{L}}_{\mathbf{P,P'}} \wedge \mathbf{0_{P'}}$; $\mathbf{L_{P,P'}} \vee \mathbf{1_{P'}} \Rightarrow \mathbf{1_P}$	$\neg(\mathbf{0_P} \wedge \mathbf{L_{P,P'}})$; $\neg(\mathbf{0_P} \wedge \mathbf{1_{P'}})$
1	$\mathbf{0_P} \Rightarrow \mathbf{L_{P,P'}} \vee \mathbf{1_{P'}}$; $\overline{\mathbf{L}}_{\mathbf{P,P'}} \wedge \mathbf{0_{P'}} \Rightarrow \mathbf{1_P}$	$\neg(\mathbf{0_P} \wedge \overline{\mathbf{L}}_{\mathbf{P,P'}} \wedge \mathbf{0_{P'}})$

Table 4: Implication rules generated from test results

A compact representation has a low number of variables, with a domain kept small, and a simple structure of the constraint network. These are contradictory aspects, so a compromise had to be made: a variable is assigned to a processor and the links to its eastern and northern neighbors. The variable does not include the fault states of the western and southern links in order to avoid modelling links twice, together with each of the two processors connected by the link.

This representation ensures that the constraint $\neg(\mathbf{0_P} \wedge \overline{\mathbf{L}}_{\mathbf{P,P'}} \wedge \mathbf{0_{P'}})$ becomes binary. Now, as **P** and **P'** are neighboring processors, the link **L** between them is assigned to either the variable representing **P** or to the one representing **P'**.

Even the implication expressing the dominance of a processor fault over the link faults in the form of $\mathbf{1_P} \Rightarrow \overline{\mathbf{L}}_{\mathbf{P,1}} \wedge \overline{\mathbf{L}}_{\mathbf{P,2}} \wedge \overline{\mathbf{L}}_{\mathbf{P,3}} \wedge \overline{\mathbf{L}}_{\mathbf{P,4}}$ is now binary. Additionally, a part of it becomes unary, i.e. the values **101**, **110** and **111** are not allowed, reducing the domain size from $2^3 = 8$ to 5.

It did not seem worthwhile to integrate more components into a single variable, as the domain size would have become intolerably large (growing exponentially with the number of components).

5.2.2 A Priori Known Faults

A list of already known faults could be generated by the power-on self-test, or by a previous diagnosis. This list is processed on the first invocation of the diagnostic part: the initialization of the constraint network includes restricting the domains of variables by node consistency, conforming to a faulty component of the list. This simplifies the constraint network, thus speeding up the diagnosis algorithm.

5.2.3 Detection of Message Losses

The constraint network built on the basis of the received syndrome messages has – in most practical cases – a huge number of solutions. This prevents the algorithm from generating any useful diagnosis. The reason for this is that, despite of the relatively low fault rate, no limitation on the number of faulty elements is used in an intrinsic way by the algorithm. If only a few of the components fail, most processors test all of their neighbors as fault-free and broadcast no syndrome messages. Thus, there are too few syndrome messages, the constraints and the solution space remains large. To overcome this problem, an inference engine reconstructs the results of tests performed by "silent" processors.

Silent processors can be classified the following way:

- *Reachable* processors have a live connection to the diagnosing processor. They are silent because there is nothing to broadcast (their neighbors are fault-free).
- *Non-reachable* processors cannot communicate with the diagnosing processor; they are isolated from it by faulty components.

The algorithm has to find the processors, which are reachable but were silent in the previous round. Numerous new constraints can be formulated by using this additional information and consequently the solution space shrinks.

	`determine the "union" of possible faults`
	`find some reachable silent processors`
	`produce new constraint from new information`
`do while` `there is new information`	

Table 5: Detection of message losses

The identification of these processors and addition of new constraints can be solved by means of reachability constraints. The transitivity of the relation "reachable" can be described by simple implication rules such as "if *x* is reachable and *y*, a neighbor of *x* is fault-free, then *y* is reachable". However, this would make the constraint network much more complex, by introducing ternary constraints.

Therefore the problem of processor reachability is handled by a non-CSP based iterative process, shown on Table 5. In the first step, the existing constraints are satisfied. Components, which were faulty in all the solutions of the network are marked as faulty. Then the silent and reachable processors are identified by a simple backtracking algorithm starting from the diagnosing processor. If such processors were found, new constraints are formulated and the first step is repeated; the subsequent processing of the extended constraint network will generate fewer solutions. The iteration terminates when there are no more silent and reachable processor candidates. In this case no further reduction of the solution space is possible. Note, that this process is strictly *monotone*, i.e. no constraint is removed from the network, only new ones are added, and the solution space is cut by the new constraints. Test runs show that the algorithm terminates after at most 5 steps, even in the case of the most complicated fault patterns.

This strategy contains the implicit assumption that links have only permanent faults. Without this assumption no diagnostic correctness is ensured. An example is

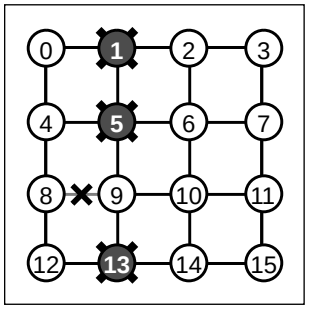

Fig. 4. Fault pattern in a system; failure of link 8-9 is transient

shown in Fig. 4; if the faulty link behaves fault-free when processor 9 tests processor 8 but blocks communication during the distribution of messages, the diagnosis program comes to the wrong conclusion that every processor is reachable. Therefore the observation seen by the program (on processor 0) that processors 1, 5, and 13 are tested faulty from the left and fault-free from the right leads to a global contradiction.

5.2.4 Implementation Characteristics

In its present form the distributed algorithm runs on a single processor. A tool simulating a cluster of the Parsytec machine and a random fault injector were developed. The diagnostic part itself could be used on a multiprocessor without any modification.

5.3 The Applied CSP Solver

A diagnostic purpose CSP solver should be written in a low-level, effectively compilable language, as it is expected to be run fast and require relatively few resources. Due to the "benign" nature of the employed special class of CSPs, there is no need to apply a full-featured, general purpose CSP solver system. The majority of known constraint-oriented languages and systems (CLPR, CONSTRAINTS, etc.) are strongly related to resource-hungry, interpreter-based languages like Prolog or LISP; a solver for special CSP classes, written in the more effective C programming language was much more promising. The applied CSP solver is based on a public domain CSP library [14], intended for solving static binary CSPs. Fortunately, its built-in preprocessing methods made it applicable for the class of dynamic CSP appearing in the developed model as well [15].

The solver was added the ability to prune obviously useless branches of the decision tree. This involves checking of an explicit upper bound on the number of faults (diagnostic *t*-limit), based on a priori knowledge about the system. Typically, uncorrelated faults affect only a few processors. The assumed testing frequency is by several orders of magnitude higher than the fault rate, thus the accumulation of many faulty components is highly improbable. If, in spite of this, the number of faults exceeds the

predefined upper bound, the constraint network becomes unsatisfiable: a global contradiction occurs. This way the incorrect assumption about the behavior of the system can be detected.

6 Reconfiguration Control

The goal of *reconfiguration* is to exclude the faulty components from the system. Usually the architecture of a multiprocessor does not allow the exclusion of an arbitrary set of components; if multiple neighboring components fail, the exclusion of a larger group of components might be necessary. Therefore, the list of components to be excluded may differ from the list of the faulty components identified by the diagnosis algorithm.

The implemented distributed diagnosis algorithm is able to generate the list of components to be excluded, based on the following modification of the system model:

- groups of 2×2 neighboring processors form so-called *reconfiguration entities*;
- a reconfiguration entity must be excluded if it has less than 3 processors that are
  - fault-free
  - connected by fault-free links
  - reachable via fault-free links and processors (see Section 5.2.3).

Moreover, all faulty components — links or processors — not contained in the excluded reconfiguration entities must be excluded one by one. Generating the list of components to be excluded is a simple task and is performed by means of conventional methods. It may be worthwhile to implement it using constraint-based methods in the case of a more complicated reconfiguration policy.

The program extends the list of the faulty components with the reconfiguration measures that remove the corresponding fault. These may be the exclusion of the component or the exclusion of the whole enclosing reconfiguration entity.

7 Measurement Results

The CSP-based diagnosis algorithm was tested by means of a logic fault injector: the host machine generated a random fault pattern for the Parsytec processors and downloaded it as part of the test 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 performed the test but reported a random result and “dead” processors remained totally inactive. 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 the final model based on T9000 was unusable due to the difference in the hardware structure.

The results of a typical test run for the centralized algorithm are shown at first on Fig. 5. In this case the fault pattern contained a single faulty processor. Fig. 6 displays the number of the candidate solutions found by the CSP solver and the number of pro-

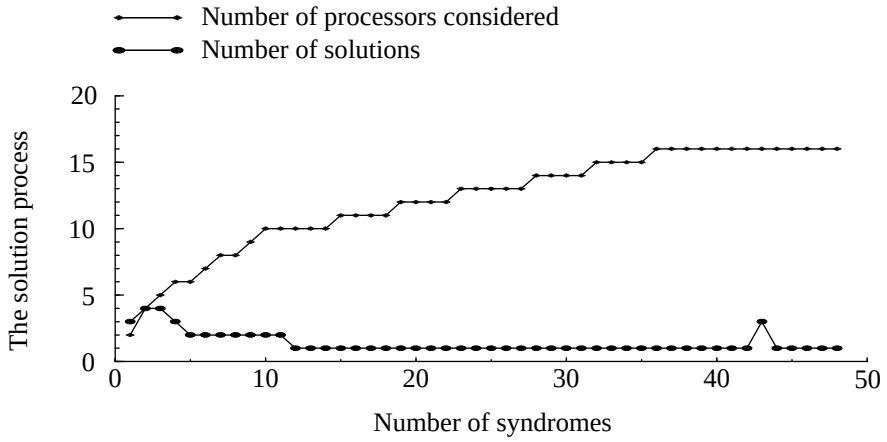

Fig. 5. Centralized diagnosis: the solution process

cessors considered at various stages of the solution process. Fig. 7 displays the number of consistency checks made, as a measure for the computational efficiency.

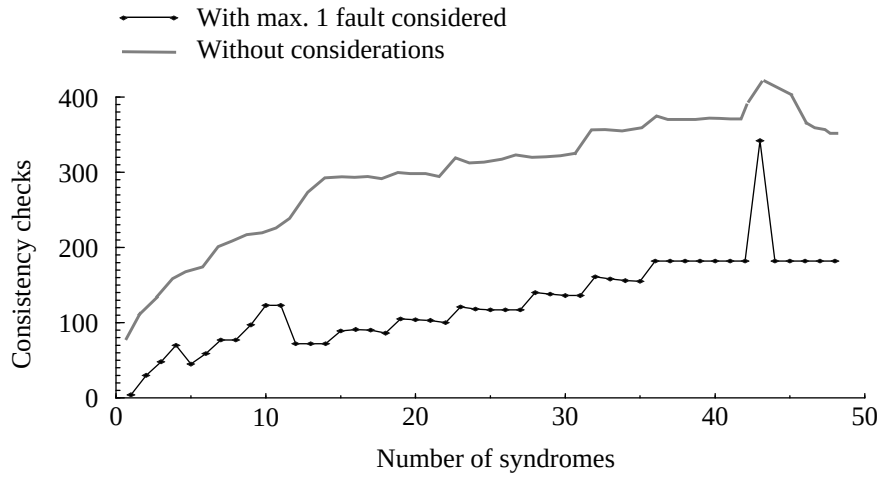

Fig. 6. Centralized diagnosis: computational efficiency

It was examined how a *t*-limit-like upper bound on the number of faults affects the efficiency of the distributed algorithm by diagnosing the same fault pattern with a variable fault limit (Fig. 8).

The efficiency of the algorithm is strongly influenced by the *t*-limit. The proper selection depends on the failure characteristics of the system: a high *t*-limit unnecessarily increases the execution time, while an excessively low one can lead to a contradiction in the constraint network and thus to no diagnosis, if there are more faults than the preset *t*-limit.

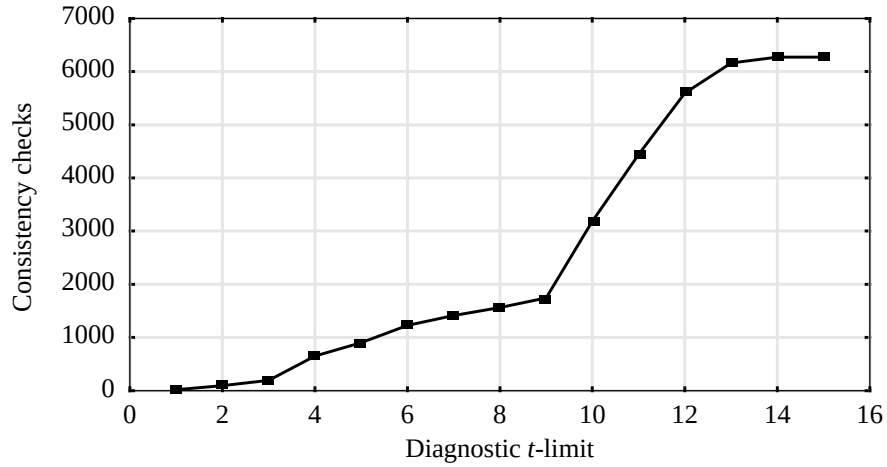

Fig. 7. Distributed diagnosis: effect of a diagnostic *t*-limit on computational efficiency

Moreover, detection of a global inconsistency – in the case when the number of faults exceeds the *t*-limit – is by some orders of magnitudes faster than finding the correct diagnosis. At values slightly greater than the actual number of faults the gradient of the curve is high. This property suggests an adaptive fault number minimization oriented diagnosis algorithm: the testing is started with the assumption of a single fault; the limit can be increased if global contradiction was detected. This strategy will result in low computational cost for the most frequent cases, while the diagnostic capabilities of the algorithm are preserved.

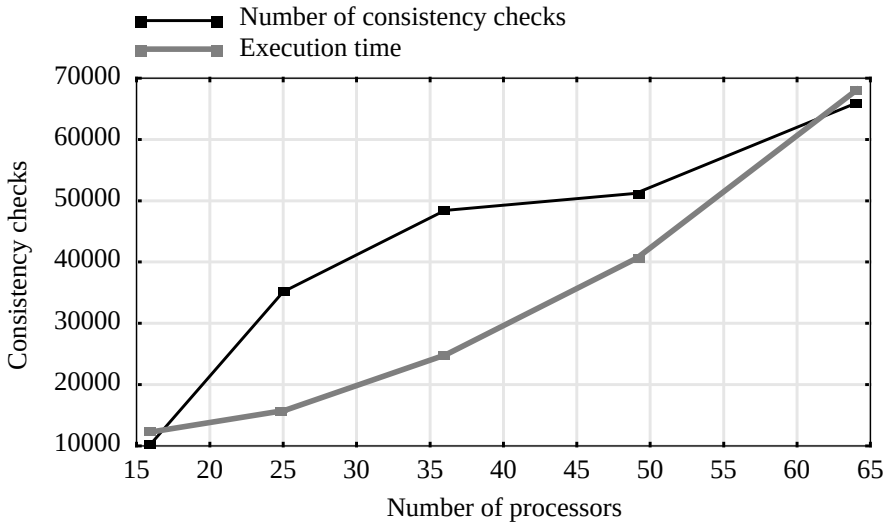

Fig. 8. Distributed diagnosis: performance versus the scale of the system

It was examined how the scale of the system affects the number of consistency checks performed by the diagnosis algorithm. The number of faults and its upper bound was fixed; the fault pattern was generated randomly. As the number of consistency checks strongly varies with the actual fault pattern, the averages of 40 test runs were taken.

The statistically fitted curves suggest that the algorithm is of a complexity $O(n^{2.52})$ (execution time) and the complexity of the backtracking part is $O(n^{1.61})$ (number of consistency checks), where n is the number of processors. In fact, the majority of the time is spent with the preprocessing of the network (this task is of complexity $O(n^3)$ for general binary CSPs). The reason for this is the CSP library, which contains only inoptimal preprocessing methods (**PC-2**, see Section 2.2). A significant improvement is expected from implementing faster preprocessing techniques. The number of consistency checks, measuring the efficiency of the backtracking part has exponential worst-case complexity. In fact, it varied strongly with the actual fault pattern, hence the 5 highest and the 5 lowest values were omitted from the analysis. The most important task in the future is to find a good heuristic function to decrease this fluctuation. Also, the CSP library is written for general binary networks and cannot make use of the special properties of the constraint network used for diagnosis. Theoretical investigations could help to find a more efficient constraint solver for this special class of CSPs.

Another possibility is to employ an approximate diagnosis method as a heuristic for the reduction of the search space. The algorithm described in [8] introduces the number of implications supporting a fault state hypothesis as a decision factor. Additional assumptions about the fault model can be included in the diagnosis by using a weighted sum instead of simply counting the implications. This heuristic could contribute to decrease the number of fruitless backtracks during the search process.

8 Conclusions and Further Work

The CSP-based diagnosis approach proved the correctness of the basic concepts, in both a centralized and a distributed environment. It is capable of handling situations which cause problems in traditional diagnosis algorithms; in particular, complex fault models and situations when a part of the system is isolated from other parts by faulty components. It was demonstrated that a t-limit affects the speed of the algorithm significantly. Adjusting this upper bound adaptively during the diagnosis enables to exploit this fact. In the distributed model the inter-processor communication was reduced radically, at the expense of a moderate performance degradation.

Further work involves selecting an optimal constraint solver focusing on memory use, fast preprocessing and exploiting the special properties (e.g. sparseness) of diagnostic constraint networks. Another task is studying the applicability to large-scale systems. Hierarchical diagnosis and stepwise fault model refinement could help reducing the time complexity.

References

1. E. Selényi, “Generalization of System-Level Diagnosis Theory,” D.Sc. Thesis, Budapest, Hungarian Academy of Sciences, 1985.
2. A. Pataricza, K. Tilly, E. Selényi, M. Dal Cin, “A Constraint Based Approach to System-Level Diagnosis,” Internal report 4/1994, University of Erlangen-Nürnberg, 1994.
3. A. Petri, “A Constraint Based Algorithm for System Level Diagnosis,” Diploma Thesis, Technical University of Budapest, 1994.
4. A. Pataricza, K. Tilly, E. Selényi, M. Dal Cin, A. Petri, “Constraint-based System Level Diagnosis of Multiprocessor Architectures,” Proc. of *8th Symp. on Microprocessor and Microcomputer Applications*, vol. 1, pp. 75-84, 1994.
5. P. Urbán, “A Distributed Constraint Based Diagnosis Algorithm for Multiprocessors,” *Scientific Conference of the Students*, Technical University of Budapest, Faculty of Electrical Engineering and Computer Science, 1995.
6. J. Altmann, T. Bartha, A. Pataricza, “An Event-Driven Approach to Multiprocessor Diagnosis,” Proc. of *8th Symp. on Microprocessor and Microcomputer Applications*, vol. 1, pp. 109-118, 1994.
7. J. Altmann, T. Bartha, A. Pataricza, “On Integrating Error Detection into a Fault Diagnosis Algorithm For Massively Parallel Computers,” Proc. of *IEEE IPDS ‘95 Symposium*, pp. 154-164, 1995.
8. T. Bartha, “Effective Approximate Fault Diagnosis of System with Inhomogeneous Test Invalidation,” submitted to the *Euromicro ‘96 Conference*, 1996.
9. K. Tilly, “Constraint Based Logic Test Generation,” Ph.D. Thesis, Hungarian Academy of Sciences, 1994.
10. U. Montanari, “Networks of Constraints: Fundamental Properties and Applications to Picture Processing,” *Information Sciences*, vol. 7, pp. 95-132, 1974.
11. R. Mohr, T. C. Henderson, “Arc and Path Consistency Revisited,” *Artificial Intelligence*, vol. 28, pp. 225-233, 1986.
12. A. Mackworth, E. C. Freuder, “The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems,” *Artificial Intelligence*, vol 25, pp. 65-74, 1985.
13. R. Seidel, “A New Method for Solving Constraint Satisfaction Problems”, *IJCAI ‘81*, pp. 338-342, 1981.
14. P. van Beek, “A Binary CSP Solution Library,” available by FTP from ftp.cs.alberta.ca.
15. G. Kondrak, “A Theoretical Evaluation of Selected Backtracking Algorithms,” M.Sc. Thesis, University of Alberta, Edmonton, 1994.
16. M. Barborak, M. Malek, A. Dahbura, “The Consensus Problem in Fault-Tolerant Computing,” *ACM Computing Surveys*, vol. 25, no. 2, pp. 171-220, June 1993.
17. F. Preparata; G. Metze; R. Chien, “On the Connection Assignment Problem of Diagnosable Systems,” *IEEE Trans. Comput.*, vol. EC-16, no. 6, pp. 848-854, Dec. 1967.
18. C. Kime, “System Diagnosis,” in *Fault-Tolerant Computing: Theory and Techniques*, D. Pradhan ed., Prentice-Hall, New York, pp. 577-623, 1985.