Efficient Fault Tolerance: an Approach to Deal with Transient Faults in Multiprocessor Architectures

Andrea Bondavalli*, Silvano Chiaradonna* and Felicita Di Giandomenico**

* CNUCE/CNR, Via S. Maria, 36, 56126 Pisa, Italy
** IEI/CNR, Via S. Maria, 46, 56126 Pisa, Italy

Abstract

Dynamic error processing approaches are an important mechanism to increase the reliability in a multiprocessor system, while making efficient use of the available resources. To this end, dynamic error processing must be integrated with a fault treatment approach aiming at optimising resource utilisation. In this paper we propose a diagnosis approach that, accounting for transient faults, tries to remove units very cautiously and to balance between two conflicting requirements. The first is to avoid the removal of units that have experienced transient faults and can be still useful for the system and the other is to avoid to keep failed units whose usage may lead to a premature failure of the system. The proposed fault treatment approach is integrated with a mechanism for dynamic error processing in a complete fault tolerance strategy. Reliability analyses based on the Markov approach and an efficiency evaluation performed by simulation are carried out.

1. Introduction

Dependence on complex computing systems is growing. Examples are systems to control satellites, aircraft or nuclear plants. Dependability is defined as that property of a system that allows reliance to be justifiably placed on the service it delivers [6]. The development of dependable systems consists in the combined utilisation of a wide range of techniques, including fault tolerance techniques, intended to cope with the effects of faults at run-time. Fault tolerance is carried out by error processing and by fault treatment [6]. Error processing aims at removing errors from a computational state, if possible before any occurrences of failure; fault treatment aims at preventing faults from being activated again.

Common practice for increasing the reliability of a system consists in identifying some classes of faults and in developing several types of redundancy and techniques for tolerating them (e.g., [9]). Redundancy involves increasing costs (and therefore a degradation in the performance-costs ratio, i.e., in the efficiency of the system) which are often considered unaffordable. Therefore we use systems with less than the desired level of dependability. Reducing costs can make it possible to afford developing dependable systems.

In this work we propose and analyse a solution for efficiently increasing the reliability of multiprocessor architectures. This problem has already been addressed in the literature, for example in [5, 8]. Multiprocessors are particularly suitable for realising techniques to tolerate processors' operational faults, which occur during system execution owing to adverse phenomenological causes. In this context, fault tolerance must be introduced into the management of the processors, which are seen as redundant resources. Our approach, beside improving reliability by tolerating processors' operational faults, aims at reducing the associated costs focusing on operational transient faults whose appropriate management may lead to significant improvements in efficiency. The error processing and fault treatment strategies we develop are integrated in a complete fault tolerance strategy to exploit each other's benefits. They must be carefully analysed to find the best tuning of their respective parameters to avoid weak points that could defeat the strategy itself.

The rest of the paper is organised as follows. In Section 2 we describe our architectural model and fault assumptions. In Section 3 the adopted error processing technique is presented. In Section 4 we deal with transient faults from the point of view of diagnosis introducing the procedures for fault treatment. We model and quantify the loss of reliability due to the presence of latent faults, improve the capabilities of the error processing mechanism and discuss the issue of an efficient management of the available resources. In Section 5 we present a preliminary evaluation of our proposal performed by simulation. We compare the effectiveness of ours and more classical schemes and analyse the behaviour of the proposed fault treatment approach. Our conclusions are in Section 6.

2 System and fault assumptions

The system can be thought of as composed of n identical computing units (processors), $u_0, ..., u_{n-1}$, considered

as atomic failure units, and two subsystems, SCHED_IN and MNG_OUT, as shown in Figure 1.

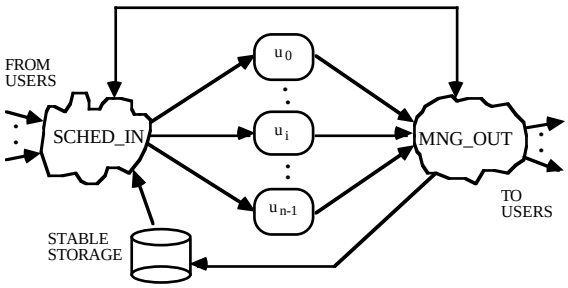

Figure 1. Logical architecture of the system.

SCHED_IN and MNG_OUT have the task of managing the processors, including the provisions taken for fault tolerance purposes. The application programs, which may have an internal state maintained between successive executions, reside on some stable storage. Upon service request, SCHED_IN selects the processors and makes available to them both the input data and the application software to run. Every selected processor executes then this application. The correct execution by other processors (or by the same one at different times) of the same application on the same input, always yields the same output. In a non fault tolerant setting, the subsystem MNG_OUT collects the output from the processors and either redirects it to the users or stores it in the stable storage. In the systems we consider, where fault tolerance provisions are taken, MNG_OUT has to take care also of adjudicating the collected results, and to diagnose the faulty processors. These are then removed by SCHED_IN which stops sending them any further service request.

The assumptions that define the fault model are:

- the faults considered are permanent and transient operational faults affecting the units;
- all the links, the stable storage, SCHED_IN and MNG_OUT are considered reliable;
- the activation of a fault in a unit u_i results in the failure of the service u_i is carrying out; if the fault is permanent, each future service provided by u_i will be a failure;
- each unit fails independently from the others in the probabilistic sense;
- two or more faults appear to be simultaneous if their activations occur within the time interval necessary to provide a service, sequential faults occur during separate service executions;
- reliable timers are used such that failures in the time domain (or omission failures) are converted into value failures by raising exceptional value errors upon the expiration of the deadlines. Therefore, we will consider only value failures;
- any unit considered permanently faulty is removed; repairs and replacing are not considered.

Although this last assumption of not considering repairs and replacements represents a restriction, this limited environment is still sufficiently large for including many interesting and important applications. Many systems must operate for all their life or at least for the duration of a mission in circumstances where repairs and replacements are not possible or considered too expensive (e.g., space-exploring vehicles, highly automated manned vehicles, satellites). We intend to explore in next studies if this restriction could be released without significant changes in the model, but simply adding new features.

3 The proposed error processing scheme

The error processing strategy we intend to adopt, called PEP in the following, originates from the SCOP (Self-Configuring Optimistic Programming) scheme [1], which was developed to tolerate design faults at the software level. In SCOP, a subset of the available variants is initially executed which would be enough to satisfy a *delivery condition* (e.g., that the result be correct given that no more than two variants fail, or that the result be correct with a minimum stated probability) if no errors occurred; then, if errors do occur, additional variants may be executed. The adjudicator checks for the satisfaction of this delivery condition, in terms of agreement among the results produced, and then, if necessary, more variants are executed until either the variants are exhausted or so many errors have occurred that the delivery condition can no longer be satisfied. The scheme is thus configured by assigning the delivery condition, the number of variants available and, in addition, a maximum allowable number of execution rounds (to represent real-time constraints)

For tolerating operational faults, variants are replaced with replicas, i.e., with identical copies of the same component. As a first approximation, our choice is to have three replicas and the 2-out-of-3 majority delivery condition, with only two rounds allowable. PEP can be regarded also as a modification of TMR (triple modular redundancy) to improve efficiency. TMR is a particular case of the classic error-masking scheme NMR (N-modular redundancy) used in many fault tolerant architectures, such as SIFT and FTMP [12]. NMR consists in the parallel execution of N replicas of a component (3 in the case of TMR), whose results are then compared by a majority voting algorithm. Figure 2 shows an iteration of the PEP scheme.

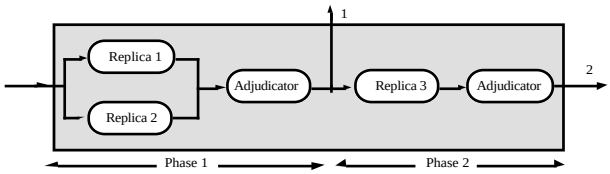

Figure 2. PEP operation.

In the first phase Replicas 1 and 2 begin to execute (possibly at the same time). After both have completed their executions, the adjudicator compares their results. If they are consistent, it accepts them (it delivers one of them as the result of the redundant component). Path 1 in Figure 2 represents termination at the end of the first phase. Otherwise, the second phase begins, Replica 3 executes, and then the adjudicator decides on the basis of all three results, seeking a 2-out-of-3 majority. Path 2 in Figure 2 represents this second case.

PEP makes a trade-off between space and time redundancy and, since the redundancy actually used while providing a service depends on fault occurrence, it results in a low overhead in the absence of faults and provides an efficient use of the available resources. The reliability analysis of PEP is simplified with respect to SCOP thanks to the hypothesis of independent failures between the components. Denoting with q_{ir} the error probability of a replica and with q_d that of the adjudicator, in [2] we derived, through a detailed dependability analysis performed adopting Markov chains, that PEP and TMR have the same failure probability, reported below:

$$p_{f_{TMR}} = p_{f_{PEP}} = 3q_{ir}^2(1-q_{ir}) + q_{ir}^3 + [(1-q_{ir})^3 + 3(1-q_{ir})^2 q_{ir}]q_d$$

4 The complete fault tolerance strategy

4.1 Fault treatment

Fault treatment consists of fault diagnosis and fault passivation. Fault diagnosis has the purpose of locating the source of the fault, i.e. the (hardware or software) component(s) affected, and understanding the nature of the fault (persistent, transient or intermittent [6]). Many diagnosis strategies have been developed in the literature. [10] presented the basic framework for system-level fault diagnosis (PCM model). In this model each unit in the system is able to test each other unit by sending stimuli and analysing the response. The derived information is collected by an ultra-reliable central unit which is then able to decide for each unit if it is fault-free or failed. Much work has been derived from it [3, 4, 11]. In particular, [3] proposed a more practical and interesting approach. The diagnostic tests' programs, which may have a low coverage and are often very expensive to run, are substituted by the application programs. Each application program is executed on pairs of processors and the results produced are compared. Two goals may so be reached: a syndrome is made available for diagnosis purposes and results obtained by fault-free processors may be released. In applications adopting fault masking methods based on voting, the diagnosis based on application programs seems particularly interesting. From the comparisons of the results of the redundant executions, performed during error processing, the syndrome *w* useful for obtaining information on the processors' status is derived.

The diagnosis model we adopt relies on the error processing mechanism and simply follows it. We define for each execution L of an application task in the system a syndrome $w(L) = [w_i]$, with i = 0,..., n-1 and

$$w(L)_i = \begin{cases} 1 & \text{if } u_i \text{ is in the majority,} \\ 0 & \text{if } u_i \text{ is not in the majority,} \\ \text{undefined} & \text{if } u_i \text{ is not involved in execution L.} \end{cases}$$

With only permanent faults, the diagnosis is completed by producing the syndrome w(L): as soon as $w(L)_i$ becomes 0 the processor u_i can be removed. Experience shows that the majority of faults are transient [13], so in systems where repairing or replacing is not allowed, removing a unit after just one failure is inappropriate. In such systems, an appropriate diagnosis strategy should aim to a trade-off between the cost in terms of general performance of the system and the cost in terms of reliability. The first is influenced by immediately removing the failed units (which could be affected by a transient fault), while the second suffers from maintaining in service units affected by permanent faults, leading to a premature failure of the error processing mechanisms.

In our solution, information on processors' behaviour is gathered at each execution. This information is then used to take a decision on whether processors are affected by permanent faults. A function α_i is associated to each not-yet-removed processor u_i to record the previous failures experienced by u_i. α_i is initially set to 0, i.e., $\alpha_i(0) = 0$, and accounts for the L-th service provided by the system as follows:

$$\alpha_i(L) = \begin{cases} \alpha_i(L-1) * K & \text{if } w(L)_i = 1, \\ \alpha_i(L-1) + H & \text{if } w(L)_i = 0, \\ \alpha_i(L-1) & \text{if } w(L)_i = \text{undefined.} \end{cases}$$

When the value of $\alpha_i(L)$ grows bigger than a given threshold α', u_i is diagnosed as permanently faulty and is then removed. The accuracy of the diagnosis depends on α_i, on its parameters H and K, on the threshold α' and on the expected ratio between permanent and transient faults. For example, if only permanent faults are expected and H=1, α' should be set to 1. If both permanent and transient faults are expected, the same unit is used many times and its failures must be detected and masked until it is diagnosed as being affected by a permanent fault.

This approach to fault diagnosis avoids removing processors affected by transient faults, but introduces a delay

in removing processors affected by permanent faults. This delay impacts the error processing mechanism and contributes to lower the overall reliability of the system.

4.2 Improving the error processing mechanism

Continuing to use a failed unit increases the probability of having two failures while providing the same service, introducing the problem of the so-called 'latent faults' [12]: two sequential faults in two distinct units are perceived as simultaneous. This phenomenon reduces system reliability. The PEP strategy may become too weak for coping successfully with the latent faults' rate. Figure 3 shows the Markov chain of one execution of PEP, considering a non null probability to start the execution with some permanently faulty processors not yet recognised by the diagnosis mechanism. $q_h(j)$ represents the probability that an execution starts with j already failed processors, j ranges from 0 to the maximum number of processors employed by the scheme (in our case 3). About the use of already faulty processors, we make the conservative hypothesis that, if employed, they are used just from the first phase of the execution. In Table 1 the states of the Markov chain and their meaning are defined.

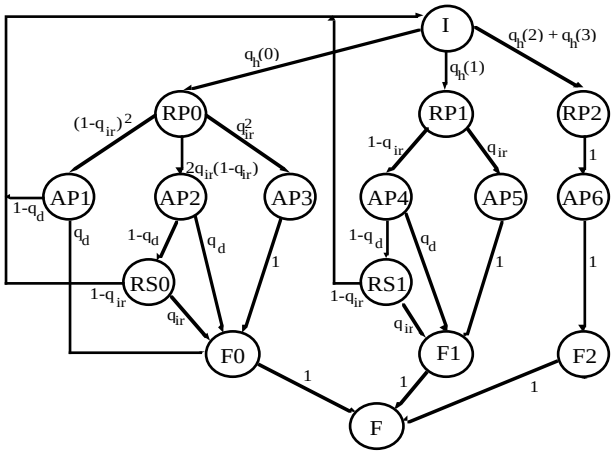

Figure 3. Reliability model for PEP starting with already failed processors.

State	Definition
I	initial state of an iteration
$\{RP_i \mid i=0,1,2\}$	start of the first phase with 'i' already failed processors
$\{AP_i \mid i=1,...,6\}$	execution of the adjudicator after RP_i
$\{RS_i \mid i=0,1\}$	execution of the third replica
$\{F_i \mid i=0,1,2\}$	failure of the execution (and of the system) after RP_i
F	dummy state to collect failures

Table 1. Definition of the states of Figure 3.

Solving the Markov model we obtain:

$$p_{f_{PEP}} = 3q_{ir}^2(1-q_{ir}) + q_{ir}^3 + [(1-q_{ir})^3 + 3(1-q_{ir})^2 q_{ir}]q_d + \\ +(1-q_{ir})^2(1-q_d)(2q_{ir}q_h(1) + (1+2q_{ir})(q_h(2)+q_h(3)))$$

By comparing this expression with the one reported earlier for PEP, it derives that, due to the latent faults' phenomenon, the probability of failure is increased by $(1-q_{ir})^2(1-q_d)(2q_{ir}q_h(1)+(1+2q_{ir})(q_h(2)+q_h(3)))$. This positive quantity may become relatively large thus making the reliability of the scheme too low for being acceptable, as described in [2]. To address this problem, we extend the proposed error processing strategy defining a variation called APEP in the following. APEP is composed of four replicas, two of which executed in the first phase. When two disagreeing results are observed, APEP re-executes the same service on two other units and not just on one, thus obtaining 4 results. A result value will be considered correct if agreed on by at least two units. So, unless two equal wrong results are obtained at the end of the first phase, APEP may have success in many cases when two processors are failed, either before starting execution or during execution. These are cases in which PEP fails, so using APEP in place of PEP improves reliability and helps in addressing the problem of latent faults.

The precise probabilities of starting an execution with some processors already failed are quite difficult to model. For a precise evaluation of the gain in reliability that APEP offers this modelling effort is necessary. Still, from the simulation that will be described in the next section, one can qualitatively appreciate that APEP is more resilient than PEP. From the point of view of the costs we point out that APEP is not significantly more expensive than PEP since the first phase of APEP, the only one performed in the large majority of cases, has the same cost as the first phase of PEP.

4.3 Processor management

A dynamic and flexible management of the processors of a multiprocessor system, aiming at maximising their use, is generally always useful to increase efficiency, and is also fundamental to PEP and APEP. In fact, if we do not have a dynamic management of the processors, we lose the possible advantages in terms of either efficiency or reliability. The scheduling policy should also avoid delays due to synchronisation, immediately employing the free processors. In this way, there may be some results already produced by some processor, called *pending* results, that cannot be judged until the corresponding results are produced by some other processor. Of course the asynchrony between processors has to be limited taking into account the possibility that a processor may be removed. Two solutions seem to be practical. The first limits the asynchrony by keeping a processor u_i idle after it has provided a number of pending results which, if all erroneous, will make the function α_i to grow bigger than α' (as in this case u_i must be removed). In the second approach, processors are executed without condition and

when a processor u_i is removed, MNG_OUT must remove also the pending results provided by u_i and re-execute the same tasks on other processors.

5 Evaluation

In this section, a preliminary evaluation is carried out via simulation, with the sole purpose of comparing the different strategies considered. The results of the reliability and performability analyses, based on analytical models, of the main error processing schemes are largely available in the literature [1, 7, 14]. Analytical models tend to become extremely complex if one wishes to include many parameters, and often the approximations introduced for obtaining models amenable to solution are such that the resulting solutions lose precision and usefulness. In this respect we consider simulation a useful complementary approach to evaluation since it permits more flexibly to investigate the different combinations of techniques and parameters.

Error processing and fault treatment strategies have complex relationships which depend, among other things, on their respective parameters. It becomes interesting to analyse different combinations of these techniques and their parameters in order to determine, if possible, suitable settings. With our simulation we looked for three quantities: 1) the average number of services the system correctly executes before its failure, 2) the average number of removed processors at system failure, and 3) the average discrepancy between how many processors the fault treatment strategy removed and how many were actually faulty. The first quantity gives a rough measure of combined dependability, performance and costs. The second is useful for checking the balance between the error processing and fault treatment strategies, consequently suggesting the subject of possible improvements. The third measure concerns only the fault treatment strategy and is useful to assess its quality: large discrepancies indicate the need for further refinements.

In the simulated system, processors are used without any synchronisation or scheduling delays, to reach the highest parallelism permitted by the employed error processing strategy assuming an 'infinite' load. Under our hypotheses, the system can fail due to the exhaustion of available processors (when less than a prefixed number of processors is working) or to the production of an incorrect value. Different kinds of services are accomplished by the system, each service requiring a constant execution time that is uniformly distributed in the interval 6-12. Due to our assumption of independence, the probability that independent faults result in coincident errors depends on the cardinality of the result type that we assume to be Boolean, Char or Integer. Each processor has the same probability of experiencing a fault, which is divided into probabilities of permanent and transient fault. The complete set of parameter values is reported in Table 2.

We compare PEP and APEP with NMR in combination with three fault treatment strategies: 1) a processor which fails is always immediately removed (optimal if only permanent faults are admitted); 2) processors are never removed (optimal if only transient faults are admitted); 3) the function α described in Section 4 is used with parameters as in Table 2. The first two fault treatment strategies are at the extremes in the spectrum of possible fault treatment strategies and have been used just to get an idea of the behaviour of the system, although they are inappropriate in contexts where both transient and permanent faults are admitted. They have been combined only with PEP, while the third one is combined with both PEP and APEP.

Parameters	Value
Number of processors in the system	16
Probability of permanent fault per execution	.0005
Probability of transient fault per execution	.0045
Probability of success per execution	.995
Necessary units for the system to be operational	9
Probability that the result of a service is boolean	.01
Probability that the result of a service is a character	.09
Probability that the result of a service is an integer	.9
α' (for considering a unit permanently faulty)	2
Increment H in α_i when unit i fails	1
Decremental factor K of α_i when unit i is correct	.9

Table 2. Parameters and values used in the simulation.

The results of our simulation, determined by executing 120 independent runs of the system, are collected in the next two tables. Table 3 shows the average number of services correctly executed by the time of system failure.

	NMR	APEP/PEP
4 replicas and strategy α	3642.95	7304.09
3 replicas and strategy α	4133.02	5509.07
3 replicas and failed processors always removed	511.19	770.57
3 replicas and failed processors never removed	946.74	1355.12

Table 3. Average number of successful services executed before the system failure.

It can be observed that the best results are obtained combining APEP and the α fault treatment strategy. APEP shows significant improvements over PEP and much better results than NMR: the number of services executed by APEP is about twice that executed by 4MR. PEP improves over 3MR of about 35-50%. The number of services executed by 4MR does not improve with respect to 3MR; on the contrary, it decreases. This is due to the fact that executing an additional replica for each service is a very high price in our context of infinite load, but it could be reasonable in other contexts. As expected,

in such setting, the two extreme fault treatment strategies are inferior to the strategy α, yielding a low number of services. These results, although quite raw, confirm our conjecture: error processing schemes using their redundancy according to the observed faults allow a more effective and efficient usage of the resources in a system. Table 4 reports the average number of processors removed by the strategy α, and the discrepancy between the removed processors and those which were truly permanently faulty. It shows that the strategy α is satisfactory, as it removes from the system almost all the permanently failed processors, and almost no processor that is not faulty, both in the case of APEP/PEP and NMR. The strategy α is thus very precise: it treats as faulty almost exactly those processors that are indeed faulty.

	Removed Processors	Permanently Faulty Processors Removed	Permanently Faulty Processors
APEP	7.94	7.68	7.71
4MR	7.92	7.51	7.52
PEP	5.31	5.10	5.28
3MR	6.37	6.12	6.26

Table 4. Average removed and permanently faulty processors at system failure.

From Table 4 it can also be observed that combining the strategy α with PEP and 3MR leads the system to fail with a number of removed processors significantly smaller than the maximum allowed: this is due to the problem of latent faults. The usage of APEP and 4MR improves the utilisation of the processors, as the system only fails when the number of required operative processors is very close to the minimum allowed.

6 Conclusions

In this paper we have proposed a fault tolerance strategy for multiprocessor architectures, composed by error processing and fault treatment, that shows interesting characteristics regarding both reliability and the associated costs. Under the faults hypotheses made, the adopted error processing scheme proves to be as reliable as TMR allowing also a lower degradation in performance. The proposed fault treatment policy α is flexible and efficient, resulting in the quick removal of permanently faulty processors. Further, it seeks to keep those units affected by transient faults in the system, so as to avoid an unnecessary reduction in performance. This approach introduces a delay in removing processors affected by permanent faults which causes the phenomenon of latent faults and lowers the overall reliability of the system. We addressed this problem by improving the capabilities of the error processing mechanism.

Our fault tolerance strategy was then compared, by simulation, with TMR in combination with the α fault treatment strategy. Different combinations of the parameters of the two error processing and fault treatment strategies were analysed, yielding results that, although preliminary, confirm the benefits reachable by our proposed approach to fault tolerance.

Acknowledgement

This research was supported by the CEC in the framework of the ESPRIT Basic Research Action 6362 "Predictably Dependable Computing Systems-2". We would like to thank the referees and our colleagues Fabrizio Grandoni and Lorenzo Strigini for their useful comments.

References

[1] A. Bondavalli, F. Di Giandomenico and J. Xu, "A Cost-Effective and Flexible Scheme for Software Fault Tolerance," Journal of Computer Systems Science and Engineering, Vol. 8, pp. 234-244, 1993.

[2] S. Chiaradonna, A. Bondavalli, F. Di Giandomenico, "A Fault Treatment Approach to Support Dynamic Redundancy in Multiprocessor Architectures," Esprit BRA 6362 PDCS2 2nd year deliverables, 1994.

[3] K. Y. Chwa and S. L. Hakimi, "Schemes for Fault-Tolerant Computing: a Comparison of Modularly Redundant and t-Diagnosable Systems," Information and Control, Vol. 49, pp. 212-238, 1981.

[4] A. T. Dahbura, K. K. Sabnani and L. L. King, "The Comparison Approach to Multiprocessor Fault Diagnosis," IEEE TC, Vol. C-36, pp. 373-378, 1987.

[5] R. E. Harper, J. H. Lala and J. J. Deyst, "Fault Tolerant Parallel Processor Architecture Overview," in Proc. FTCS-18, Tokyo, Japan, 1988, pp. 252-257.

[6] J. C. Laprie, "Dependability: a Unifying Concept for Reliable Computing and Fault Tolerance," in "Dependability of Resilient Computers", T. Anderson Ed., BSP Professional Books, 1989, pp. 1-28.

[7] J. C. Laprie, J. Arlat, C. Beounes and K. Kanoun, "Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures," IEEE Computer, Vol. 23, pp. 39-51, 1990.

[8] F. Lombardi, "Optimal Redundancy Management of Multiprocessor Systems for Supercomputing Applications," in Proc. 1st. Int. Conf. on Supercomputing Systems SCS-85, St. Petersbourgh, Fl., 1985, pp. 414-422.

[9] D. Powell, "Failure Mode Assumptions and Assumption Coverage," in Proc. FTCS-22, 1992, pp. 386-395.

[10] F. P. Preparata, G. Metze and R. T. Chien, "On the Connection Assignment Problem of Diagnosable System," IEEE TEC, Vol. EC-16, pp. 848-854, 1967.

[11] S. Rangarajan and D. Fussel, "A Probabilistic Method for Fault Diagnosis of Multiprocessor Systems," in Proc. FTCS-18, Tokyo, Japan, 1988, pp. 278-283.

[12] D. Rennels, "Fault Tolerant Computing: Concept and Examples," IEEE TC, Vol. c-33, pp. 1116-1129, 1984.
[13] D. P. Siewiorek and R. S. Swarz, "Reliable Computer System - Design and Evaluation," Digital Press, 1992.
[14] A. T. Tai, A. Avizienis and J. F. Meyer, "Performability Enhancement of Fault-Tolerant Software," IEEE TR, Vol. R-42, pp. 227-237, 1993.