International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 3– No.2, July 2012 – www.ijais.org

Adaptable Fault Tolerance Configurations for Multiprocessor Systems

Samia A. Ali
Electrical Engineering Department
Assiut University
Assiut, Egypt

ABSTRACT

The escalating increase in the complexity of multiprocessor systems increases the probability of faults occurring in these systems As a consequence there is a great need for achieving fault-tolerance of processing in multiprocessor systems. Fault-tolerance generally requires some forms of hardware and/or time redundancy. Two fault tolerant configurations are proposed for both single and double transient and permanent faults in any processor of multiprocessor systems. The tolerance for faults takes place in three consecutive steps; fault detection, fault diagnosing and system recovery. The overhead cost for the first (second) configuration is only 100% hardware (time) for fault detection, an extra 100% time for fault diagnoses and system recovery only for those processes running on the faulty processors. The advantages of the proposed configurations are the ease of applicability and the low associated overhead cost over the system without any fault tolerance. An enhancement is developed for both configurations to check upon the system state adequately to detect and recover from faults as soon as they infect the system. Simulations are performed to illustrate the usefulness of the proposed configurations.

General Terms

Fault Tolerance, Multiprocessor Systems.

Keywords

Hardware Redundancy, Time Redundancy, Transient Fault, Permanent Fault, Cold Standby Spare.

1. INTRODUCTION

The advances in technology make processors more powerful but also more vulnerable to hardware errors. Permanent or intermittent hardware faults, caused by defects in the silicon or metallization or process package and wear out over time, lead to “hard faults”. Moreover, approaching the ultimate limits of silicon in terms of channel width, power supply and speed produce circuits increasingly sensitive to noise, which will result in unacceptable rates of soft-errors. Manufacturing testing and periodic testing cannot accommodate soft errors. Recent studies [1] have projected that between a two to nine-orders-of-magnitude increase in logic circuits’ soft error rates. Therefore, fault tolerant techniques are essential for future multiprocessor systems. A typical solution for handling soft errors in high availability systems is to replicate the computation and compare the results to detect an error [2] and then do either backward or forward for error recovery [3]. Some researchers have proposed variants of integrated checking at the processor level [4, 5]. Several studies have evaluated core-level fault detection and containment by running redundant processes, either on a separate thread [6, 7] or on a separate core [8, 9]. Current systems offering very high availability, such as the IBM z-series [10] provide coverage for both permanent and transient hardware faults through a combination of redundant processor hardware and error correcting codes in memories. Redundant processor hardware, employing dual (or triple) modular redundancy – DMR (TMR) – can be applied at different granularities. Redundant hardware not only detects the presence of faults, therefore avoiding costly errors and system failure, but also allows applications to continue executing, without downtime, until faulty component(s) can be replaced.

Normally, there are three basic forms of hardware redundancy: passive, active, and hybrid. Passive techniques mask the fault effects to the next level. The active approach detects the occurrence of faults and takes actions to correct them. The various active redundancy approaches are duplication with comparison, standby sparing, pair-and-a-spare technique, and watchdog timers. Duplication with comparison is often used to detect errors. If two processors disagree on the result found for the same process, obviously there is an error. Standby sparing is another form of active redundancy. In this technique, one module is operational with one or more modules used as spares. Various schemes are used in each module to detect errors. When an error is detected in the operational module, it is removed from operation and replaced by one of the spares.

There are two types of standby spares; hot and cold. In hot standby sparing, spare modules compute the same function as the operational module so that they are ready to take over at any time. In cold standby sparing, spare modules are powerless until needed to replace the faulty operating module. The pair-and-a-spare technique combines the duplication with comparison technique with the standby sparing technique. Two modules work in parallel all the time. Their results are compared to detect discrepancies. Error reports are used to define the faulty unit. Then a spare is used to replace the faulty module.

Time redundancy is the extra time needed to detect and correct faults. Time redundancy is particularly useful in applications in which time is not a critical issue. Conversely, it is not appropriate for hard real-time applications (i.e., stringent deadlines). It provides a viable solution for space missions because weight, size, and power consumption are critical aspects of spacecraft design. Time redundancy is extremely useful to detect transient faults. Simple retries allow for transient fault detection as well as their correction. Therefore, the form of incorporated redundancy should be carefully determined. Many fault-tolerant systems that are deployed today are not reusable because the fault-tolerance mechanisms used in these systems are intimately connected to the specific applications that run on them [11]. If the applications are changed, then the fault-tolerance mechanisms must be changed as well.

In this paper, we propose two configurations for fault-tolerant scheduling of parallel programs in multiprocessor systems. With the proposed configurations, fault-tolerance can be achieved at a small cost of either time or hardware redundancy. The proposed schemes of this paper are pair-and-a-spare type technique combined both the time and hardware redundancy. The time redundancy is employed to detect and recover from transient faults. While the hardware redundancy, in the form of pair-and-spare, is introduced to detect and recover from permanent faults.

This paper is organized as follows. Section2 provides preliminary assumptions for the proposed fault tolerant configurations. Section 3 presents a description for the first proposed configuration. In section 4, the second proposed configuration is presented. An enhancement for both proposed configurations is shown in Section 5. Simulation results are given in Section 6 to illustrate the usefulness of the proposed configurations. Finally, Section 7 draws conclusions from this work.

2. PRELIMINARY ASSUMPTIONS

It will be assumed throughout this paper that the occurrence of transient faults is more likely than the occurrence of permanent faults. Also, the probability of single faults either transient or permanent is much higher than the probability of double transient or permanent faults. Moreover, the probability of two adjacent processors being permanently faulty is more likely than the probability of two nonadjacent processors being transiently or permanently faulty. Transient faults will be active for a single running and the processor will self-recover by the time of the following run. A permanent faulty processor reports matched faulty results for a given process executing any number of times on it. However, if a given processor experienced a transient fault its reported result for the same process will not necessarily match.

The multiprocessor system under consideration consists of autonomous processor modules connected by an interconnection system (bus, direct links or switching networks). Each processor is equipped with a local memory which can be accessed by a local bus. The local memory of one processor module is accessible by other processor modules. A fault in any processor is assumed to manifest itself as a failure of that processor, while a fault in an interconnection facility is attributed to the failure of one or more processors which make use of that facility.

To ease the explanation of the proposed configurations, the microprocessor system is assumed to be consisting of a master processor, and even number of working processors and a single cold spare processor. The master processor is a distinguished processor responsible for distributing data to the working processors, receiving processor's results, comparing those results to detect faulty processors, diagnose those faults and initiating the recovery actions. Moreover, the master processor is assumed to be fault-free and a self-checking processor. It performs the recovery from permanent faults by replacing the faulty processor with the cold spare processor and orders replacement and/or repair for the faulty processor to keep a cold spare available in the system for future replacements. In case of multiple permanent faults, the master processor replaces the faulty processors one after another until all faulty processors have been replaced and schedules rerun for the processes with mismatched results.

3. THE FIRST CONFIGURATION

The multiprocessor system with the incorporated first configuration for fault tolerance with n processor nodes, a single spare processor, and a master processor is able to execute $n/2$ processes simultaneously as follows:

1- The master processor assigns $n/2$ independent processes, P_j's and $0 \le j \le n/2-1$ to be executed on the n working processor nodes, N_i's and $0 \le i \le n-1$. More specifically, process P_j is assigned to be executed simultaneously on nodes N_j and $N_{j+n/2}$ (see Table. 1).

2- Each processor node Ni, $0 \le i \le n-1$, executes its assigned process and reports its result to the master processor.

3- Upon receiving the two results for each of the running process, the master processor performs $n/2$ comparisons for the $n/2$ sets of the reported results; one set for each executed process.

4- According to the results of the $n/2$ comparisons, the master processor determines the state of the multiprocessor system. Consequently, there are three distinguishable cases

Case (I): There is a complete match for all the $n/2$ set of results reported to the master by the n executing processors. The master processor concludes that the multiprocessor system is free of faults and proceeds to schedule $n/2$ new processes to be executed on the multiprocessor system. This case is the most probable to occur in normal system condition.

Table 1. Node assignment for the first configuration.

Node Number	Running Process
N0	P0
N1	P1
N2	P2
N3	P3
N4	P0
N5	P1
N6	P2
N7	P3

Case (II): There is only a single mismatch in the reported results for a single running process. Let us assume that only the two results of P_j are mismatched. The master processor diagnoses that the multiprocessor system has incurred either a single or double transient or permanent faults. The master processor must keep the set of reported results of the mismatch process, P_j, for further analysis in the next run to enable the master to recover from (tolerate) the incurred fault(s). The master processor schedules a new set of $n/2-2$ processes and the process with the mismatch results, P_j to be executed on the multiprocessor system. The master assignes process, P_j to be executed simultaneously on N_j , N_{j+1}, $N_{j+n/2}$, and $N_{j+n/2+1}$ (see Table 2). This is done in order to distinguish between the existence of transient and permanent faults in the system's processors.

Following the second run the master processor diagnoses and recovers from the incurred fault(s) at the previous run as follows:

1- Single or Double Transient fault(s) in the first run at either or both N_j and $N_{j+}{}^{n}\!/_{2}$ if all processors in the second run have reported matched results for all the processes running on them. In this case no action is necessary.
2- Single Permanent fault at processor Nj (or Nj+n/2) if the result reported by processor Nj+n/2 (or Nj) in the first run is matched with the results reported by the processors Nj+$_1$, $N_{j+n/2}$, and $N_{j+n/2+1}$ (or N_j , N_{j+1}, and $N_{j+n/2+1}$) in the second run. The master replaces the faulty processor with the spare processor.
3- Double Transient fault; a single in the first run at N_j (or $N_{j+}{}^{n}\!/_{2}$) and another single in the second run at $N_{j+}{}^{n}\!/_{2}$ (or N_j) if $N_{j+}{}^{n}\!/_{2}$ (or N_j) in the first run as well as N_j, N_{j+1}, and $N_{j+n/2+1}$ (or $N_{j+}{}^{n}\!/_{2}$ and $N_{j+n/2+1}$) in the second run have reported matched results. Also, no action is necessary.
4- Double transient fault in the first run at N_j and $N_{j+}{}^{n}\!/_{2}$ and double transient fault in the second run at any two processors of the four processors, N_j , N_{j+1}, $N_{j+n/2}$, and $N_{j+n/2+1}$, if only two processors in the second run have reported matched results for P_j. No action is necessary.
5- Single permanent fault at N_j (or $N_{j+}{}^{n}\!/_{2}$) and a single transient fault at any of the processors, $N_{j+}{}^{n}\!/_{2}$ (or N_j), N_{j+1}, and $N_{j+n/2+1}$ in the second run if only two processors from $N_{j+}{}^{n}\!/_{2}$ (or N_j), N_{j+1}, and $N_{j+n/2+1}$have reported matched results for P_j in the second run. The master processor replaces N_j (or $N_{j+}{}^{n}\!/_{2}$) with the spare processor.
6- Single permanent at N_j (or $N_{j+}{}^{n}\!/_{2}$) and double transient at the two processors, N_{j+1}, and $N_{j+n/2+1}$ in the second run if the processor $N_{j+}{}^{n}\!/_{2}$ (or N_j) has reported matched results for P_j in the first and second run. The master processor replaces N_j (or $N_{j+}{}^{n}\!/_{2}$) with the spare processor.

Table 2. Node assignments for the first configuration after a mismatch for P0.

Node Number	Running Process	Processes of the Second Run
N0	P0	P0
N1	P1	P0
N2	P2	P4
N3	P3	P5
N4	P0	P0
N5	P1	P0
N6	P2	P4
N7	P3	P5

7- Single permanent at N_j (or $N_{j+}{}^{n}\!/_{2}$) and double transient; one at either $N_{j+}{}^{n}\!/_{2}$ (or N_j) in the first or the second run and the other at one of the two processors, N_{j+1}, and $N_{j+n/2+1}$ in the second run . if $N_{j+}{}^{n}\!/_{2}$ (or N_j) in the second or first run has reported matched results as one of the two processors N_{j+1}, and $N_{j+n/2+1}$ in the second run. The master processor replaces N_j (or $N_{j+}{}^{n}\!/_{2}$) with the spare processor.
8- Double permanent fault at N_j and $N_{j+}{}^{n}\!/_{2}$ if only the two processor, N_{j+1}, and $N_{j+n/2+1}$, have reported matched results for P_j at the second run. The master processor should replace the two faulty processors N_j and $N_{j+}{}^{n}\!/_{2}$ one after the other with the available spare processor and order repair for the faulty processors as soon as they are replaced.

Case (III): There are two mismatched set of results for two of the running processes in the multiprocessor system. Let us assume that P_i and P_j are the two processes with mismatched results. The master processor concludes that the system has experienced either a quadruple, triplet, double transient or double, single permanent faults or any combination of transient and permanent faults in any of the four processors; N_i , $N_{i+n/2}$, N_j and $N_{j+n/2}$. In order to figure out which processor experienced what type of fault the master processor schedules *n/2 – 2* new processes and reschedules the two processes with mismatched results to be executed in the system processors. However, the two processes P_i and P_j with mismatched results in the first run should switch processors with each other. In other words, P_i should run on N_j and $N_{j+n/2}$ and P_j should run on N_i and $N_{i+n/2}$ in the second run (see Table 3).

Following the second run the master processor diagnoses the state of the multiprocessor system as follows:

1- Either Quadruple, Triple, or Double Transient faults in the first run at N_i, $N_{i+n/2}$, N_j and $N_{j+n/2}$ if all processors in the second run have reported matched results for P_i and P_j. In this case no action is necessary.
2- Double Permanent faults; one at N_i (or $N_{i+n/2}$) and another at N_j (or $N_{j+n/2}$) if $N_{i+n/2}$ (or N_i) in the first run reported the same result for P_i as $N_{j+n/2}$ (or N_j) in the second run. Also, $N_{j+n/2}$ (or N_j) in the first run has reported the same result as $N_{i+n/2}$ (or N_i) in the second run. The master replaces one of the faulty processors with the system spare processor and orders repair in order to be able to replace the other faulty processor. The occurrence of this case is rare.
3- Single permanent at N_i (or $N_{i+n/2}$) and another single transient in the first run at N_j (or $N_{j+n/2}$) or vice versa for the permanent and transient faults. This happens if $N_{i+n/2}$ (or N_i) in the first run has reported matched results as N_j and $N_{j+n/2}$ in the second run for P_i , and $N_{j+n/2}$ (or N_j) in the first run has reported matched results as N_i and $N_{i+n/2}$ in the second run for P_j. The master processor replaces the permanent faulty processor with the system spare and orders repair for the faulty processor to be available as system spare.
4- Double transient in the first run at any two processors of the four processors; N_i , $N_{i+n/2}$, N_j, and $N_{j+n/2}$ and double transient in the second run at the two fault-free processors in the first run. No action is necessary.

In the two cases II and III, the master processor is able to recover from the fault(s) occurred in the previous run. However, it is possible that the multiprocessor system may incur further faults either transient or permanent in the current (i.e., the recover run for the faults occurred in the previous) run. The master processor detects those faults in this current run and recovers in the following run. Thus the first proposed configuration detects and tolerates the existence of any number of transient faults up to four, double and/or single permanent faults or combination of transient and permanent faults. The cost of tolerance is doubling the hardware required for processes executions without providing faults detection and tolerance, and a time overhead of 100% only for those processes running on faulty processors. Moreover, the first proposed configuration recovers from double permanent faults by replacing one of the two faulty processors with the spare processor provided in the system, orders repair for the two faulty processors and when repaired replaces the other faulty processor and keeps the other as a system spare. In conclusion the total overhead cost for the first configuration is 100% of hardware and no time overhead for the normal case; case (I) when the multiprocessor system is free of all faults. The overhead cost is 100% of hardware and 100% time only for the processes running on faulty processor; cases II and III. It is worth noting that processes running on fault-free processors do not experience any delay.

Table 3. Node assignments after the mismatch for Both P1 and P3

Node Number	Processes of the First Run	Processes of the Second Run
N0	P0	P4
N1	P1	P3
N2	P2	P5
N3	P3	P1
N4	P0	P4
N5	P1	P3
N6	P2	P5
N7	P3	P1

4. THE SECOND CONFIGURATION

As for the first proposed configuration the microprocessor system is assumed to be consisting of a master processor and n working processor nodes called Ni, $(0 \le i \le n-1)$ and a single cold spare to replace any of the working processor nodes in case of permanent faults striking the working processors. The multiprocessor system with the incorporated second configuration for fault tolerance proceeds as follows:

1- The master processor assigns n independent processes; Pj $(0 \le j \le n-1)$ to be executed on the n working processor nodes, Ni's $(0 \le i \le n-1)$ respectively (see Table 4).

2- Each Ni $(0 \le i \le n-1)$ executes its assigned process and reports its execution result to the master processor for comparison.
3- The master processor reassigns the same processes Pj's $(0 \le j \le n-1)$ for a second run on the working processors Ni's $(0 \le i \le n-1)$. However, this time each process must be executed on a different working processor node in order to detect the existence of faults within the working processors. More specifically, in the second run Ni executes process P(i+1) MOD n , where $(0 \le i \le n-1)$ (see Table 4). Similar to the first run, each Ni executes its assigned process and reports its execution result to the master processor.
4- The master processor performs *n* comparisons simultaneously for the 2 *n* reported results one comparison for each of the *n* processes over the two runs.

Accordingly, there are three distinguishable cases presented below in a decreasing order of the probability of their occurrence in normal system operation.

Case (I): The *n* sets of results reported to the master processor from the two runs for the *n* processes Pj's $(0 \le j \le n-1)$ are exactly matched. The master processor concludes that the system is free of all types of faults. This case is most probable case.

Case (II): Only one set of all the reported sets to the master processor from the two runs for the *n* processes, Pj's $(0 \le j \le n-1)$ has reported mismatched results. The master processor concludes that the system has incurred either single or double transient and/or permanent fault in one of the two processors reporting the set of two mismatched results. The master processor reschedules the process, Pi; with mismatch set of results for another run with *n-1* new processes to be executed by the system (see Table 5 where Pi is P1). In this third (current) run, process Pi, should be schedule to execute in processor N(i+2) MOD n.

After the third (current) run the master processor distinguishes between the different incurred faults as follows:

1- Single transient or permanent fault in the first or second run at Ni or N(i+1) MOD n if there are two matched results reported for Pi by either N(i+1) MOD n or Ni, and N(i+2) MOD n in this run. The distinction between transient and permanent is performed by the master according to the result of the next system run. More specifically, if in the next run the suspected faulty processor, Ni or N(i+1)MOD n has been involved in another mismatch for another process then the fault is permanent otherwise it is transient.

Table 4. Node assignments for the second configuration

Node Number	Processes of the First Run	Processes of the Second Run
N0	P0	P7
N1	P1	P0
N2	P2	P1
N3	P3	P2
N4	P4	P3
N5	P5	P4
N6	P6	P5
N7	P7	P6

2- Double transient fault at any two processors of the three processors; Ni, N(i+1) MOD n and N(i+2) MOD n if there is no matched results for the process Pi. The master schedules process Pi to run in processor P(i+3) MOD n to obtain the correct result for Pi.

Table 5. Node assignments for the second Configuration for case II after the mismatch of P1

Node Number	Processes of the First Run	Processes of the Second Run	Processes of the Third Run
N0	P0	P7	P8
N1	P1	P0	P9
N2	P2	P1	P10
N3	P3	P2	P1
N4	P4	P3	P11
N5	P5	P4	P12
N6	P6	P5	P13
N7	P7	P6	P14

Case (III)): Two sets of processes; Pi and Pk of the n processes have mismatched results. The master processor reschedules these two processes Pi and Pk for another run on processors P(k+1) MOD n and P(i+1) MOD n respectively with n-2 new processes to run in the multiprocessor system. For illustration purposes, see Table 6 and let Pi and Pk be P1 and P3 respectively.

After the third (current) run the master processor distinguishes between the different incurred faults as follows:

1- Double and/or Single transient faults at Ni and/or Nk in the first run if there are two matched results for processes Pi and/or Pk in the last two runs. No action is necessary.
2- Double and/or Single transient faults at N(i+1) MOD n and/or N(k+1) MOD n in the second run if there are two matched results for processes Pi and/or Pk in the first and last run. No action is necessary.
3- Double and/or Single permanent faults at N(i+1) MOD n and/or N(k+1) MOD n started in the second run if there is no match results for Pi and/or Pk in the three consecutive runs. The master processor assures the existence of permanent faults if and only if the two processors N(i+1) MOD n and/or N(k+1) MOD n have reported mismatch results for the processes assigned to them in the two subsequent runs. Otherwise, the master decides that the incurred faults are transient and no further action is required. In case of permanent faults the master processor replaces the faulty processors N(i+1) MOD n and/or N(k+1) MOD n one after the other with the spare processor and orders repair for the faulty ones.

Table 6. Node assignments for the second Configuration for case II after the mismatch of P1and P3

Node Number	Processes of the First Run	Processes of the Second Run	Processes of the Third Run
N0	P0	P7	P8
N1	P1	P0	P9
N2	P2	P1	P3
N3	P3	P2	P10
N4	P4	P3	P1
N5	P5	P4	P11
N6	P6	P5	P12
N7	P7	P6	P13

In both two cases, II and III, the master processor tolerates the fault(s) occurred in the previous two runs. However, it is possible that the multiprocessor system may incur further faults in recovery runs for faults occurred in previous runs. Those faults incurred in recovery runs are detected and tolerated by the master processor recursively as done above. More specifically, the master keeps the results of every two successive runs, compares these reported results for all the processes executed in these two runs; detects the incurred faults and then reschedules one more run for those processes incurred the faults with other new processes to recover from the incurred faults. Thus the second proposed configuration detects and tolerates the existence of double and/or single transient faults, and double and/or single permanent faults or combination of those transient and permanent faults. The cost of this is doubling the time required for executions without providing any fault detection or tolerance plus an extra time to rerun the processes executed on faulty processors. The overhead cost for the second configuration is 100% of time and no hardware overhead for the normal case; case (I) when the multiprocessor system is free of all faults. The overhead cost is 100% of time and an extra time for one more run only for process(s) experienced some type of faults during its two runs on the multiprocessor system (i.e., cases II and III).

5. ENHANCING THE PROPOSED CONFIGURATIONS

With the current scale of massively parallel systems, occurrence of faults is no longer an exception but it is the norm. As more of such systems are being deployed in practice, issues of fault-tolerance and self-healing are becoming tremendously important. The reliability of a fault tolerant system depends upon a reasonably fast detection of faults to ensure that no more than the number of tolerated faults is active at the same time. While hardware fault-tolerance can be achieved by deploying redundant hardware components and re-allocating alternate hardware resources to the applications at run-time, this approach is not cost-effective. The age-old checkpoint/restart mechanism still seems most attractive due to its simplicity and low-cost. The high failure rate of these systems puts additional pressure on checkpoint mechanisms. Checkpoints should now be taken more frequently [11, 12] relative to the failure rate of the system which, in turn, directly impacts the application running time and disk-storage requirements. One of the ways to reduce the checkpoint file size is incremental check-pointing technique which is proposed and implemented by several researchers [13, 14].

The sooner the multiprocessor system detects and recovers from the infected faults the more reliable the system will be. The performance of the two proposed fault tolerant configurations for multiprocessor systems can be increased substantially through periodical checking upon the system processors; every τ time interval. Therefore, in order to increase the performance of the two presented fault tolerance configurations we propose dividing each running process, similar to dividing memory into pages, into a number of sub-processes each with execution time τ. The exception is only may be for the last sub-process which could have an execution time less than τ. The master processor conducts a periodical checkup upon the state of the system processors every τ time interval. More specifically, at the end of every τ time interval each of the executing sub-processes reports its result to the master processor. Upon receiving two results for each sub-process, the master processor conducts the comparison as described in Sections 3 and 4 to learn the state of the system processors and performs the necessary actions. It is worth noting that the periodical checkup for the multiprocessor system in the fault-free state occurs at the end of every τ time interval for the first proposed configuration and at every other time interval (i.e., 2τ) for the second proposed configuration. Otherwise, when the system experiences any type of fault, the master processor conducts its check for the multiprocessor system every τ time interval for both the proposed configurations until the system recovers completely from all faults and returns to its fault-free state.

There is number of issues concerning the duration of the time slices τ; how long should it be to achieve the best enhancement for the reliability of the multiprocessor system. If the time slice τ, is too short the master processor will be performing checking upon the system processors more frequently, thus wasting valuable processing time. On the other hand, if the time slice τ is too long the system may experience more faults than it can tolerate efficiently and thus decreasing the multiprocessor system performance drastically. Deciding upon the period of the time slice is similar to the page size of the memory system. Therefore, the time slice τ should be moderate; not too short nor too long. There are some factors should be taken into consideration to figure out the suitable time slice τ for a given multiprocessor system deploying either of the proposed configurations. Some of the foremost factors are the system exposure to faults, the frequency of fault occurrence, the nature of the executing applications and the environment where the system is deployed. The most suitable time slice for a given system and executing applications could be found through some heuristic.

Applying the proposed enhancement, the master processor is able to detect any exposed single transient, permanent or double transient, permanent faults at any working processor node within at most the chosen time interval τ from the instance of the system exposure to the fault(s). Using the proposed enhancement, diagnosing the system faults is within at most 2 τ, and recovering in at most 3 τ. Thus employing the proposed enhancement saves precious system execution time. However, without the proposed enhancement the master processor will diagnose the fault state of the system after unspecified amount of time depending on the execution time of the executing processes during the system fault exposure. The worst case for the wasted time for the system without the proposed enhancement would be the longest execution time for any of the executing processes during the fault attack.

6. SIMULATION RESULTS

For this research we have developed a discrete event simulator, which has been designed to deal with the two proposed fault tolerant configurations for multiprocessor system described in sections 3, 4, and 5. The simulation program is written using C# in which we have simulated a multiprocessor environment consisting of eight processors and a cold spare. The application chosen to perform the simulation is multiplication of two matrices, one of size 8 x16 and the other is 16x1. Each processor computes one element of the resultant matrix. The simulation for the first proposed configuration verified the different fault categories (cases II and III of Section 3). Similarly, the simulation for the second proposed configuration has verified the different fault categories for cases II and III of Section 4.

Simulation for the enhanced configuration has been performed with the assumption that the context switch time is τ. The chosen context switch time is high however; it is the most suitable for matrix multiplication. The matrix multiplication process is divided to 2, 4, 8, and 16 sub-processes. The fault(s) has been introduced randomly at the beginning of a slice and the simulation performed for 1, 2, 4, 8, and 16 slices for no fault, single transient, and single permanent, double transient, and double permanent faults. The result of the simulation is shown in Fig.1. The time consumed to detect and recover from faults is about 3τ for double transient, single transient and permanent faults and about 9τ for double permanent faults.

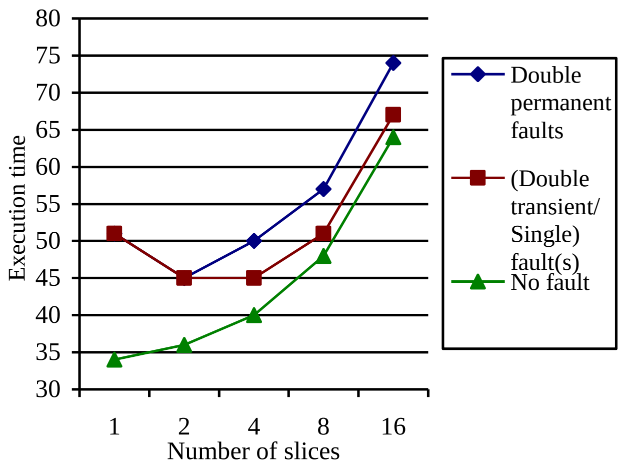

Fig 1: Illustration of Slicing on the performance of Multiprocessor System for the proposed Configurations.

7. CONCLUSIONS

High performance multiprocessor systems offer promising and powerful mechanisms for large scale computation. The chances that those massively parallel systems will experience component failures from time to time are high. Designers of massively parallel systems cannot just demand that no parts of the system should fail; they have to be aware that this or that part of the system may eventually fail and have to take the steps to add redundancy and reconfigure. In this paper, we have presented two fault tolerant configurations with variable overhead cost to detect, diagnose and tolerate single and double transient and/or permanent faults in multiprocessor systems. The overhead cost depends upon the fault state of the multiprocessor system. More specifically, for fault detection only (i.e., the system is free of all faults which is the norm for most systems) the overhead cost for the first configuration is 100% hardware and no time overhead. While, the second configuration has 100% time overhead and no hardware overhead cost. For fault diagnoses both the proposed configurations require an extra overhead of 100% time but only for those processes executed on faulty processors.

Furthermore, at most another 100% time overhead only for those processes executed on permanent faulty processor for both configurations is required to recover from incurred permanent faults.

An enhancement is presented for both fault tolerant configurations to reduce the fault existence duration in the system before detection to a minimum value of τ. This is accomplished through dividing the execution time of each executing process into slices; each with a time interval τ. The most suitable time interval τ for a given multiprocessor system depends upon the nature of the executing processes, the system susceptibility to faults, and the environment where the system is deployed. A good estimation for τ could be found through some heuristics.

8. REFERENCES

[1] Shivakumar, P. Keckler, S.W., Moore, C.R., Burger, D., "Exploiting Microarchitectural Redundancy for Defect Tolerance", the 21st International Conference on Computer Design (ICCD), October, 2003.

[2] Bernick, D., Bruckert, B., Vigna, P. D., Garcia, D., Jardine, R., Klecka,J., Smullen, J., "NonStop® Advanced Architecture", DSN, 2005.

[3] Anderson, T., Lee, A., "Fault-tolerance - Principles and Practice", Prentice Hall, Eaglewood Cliffs, 1981.

[4] Qureshi, M. K. et al. Microarchitecture-based introspection: A technique for transient-fault tolerance in microprocessors. In Proc. of 32nd Intl. Symp. on Comp. Arch. (ISCA-32), June 2005.

[5] Ray, J. et al. Dual use of superscalar datapath for transient-fault detection and recovery. In Proceedings of the 34th International Symposium on Microarchitecture, December 2001.

[6] Rotenberg, E.. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In Proceedings of the 29th International Symposium on Fault-Tolerant Computing, June 1999.

[7] Vijaykumar, T. N. et al. Transient-fault recovery using simultaneous multithreading. In Proceedings of the 29th International Symposium on Computer Architecture, May 2002

[8] Gomaa, M. et al. Transient-fault recovery for chip multiprocessors. In Proceedings of the 30th International Symposium on Computer Architecture, June 2003.

[9] Mukherjee, S. S. et al. Detailed design and evaluation of redundant multithreading alternatives. In Proceedings of the 29th International Symposium on Computer Architecture, May 2002, 99–110.

[10] Fair, M.L., Conklin, C.R., Swaney, S. B., Meaney, P. J., Clarke, W. J., Alves, L. C., Modi, I. N., Freier, F. , Fischer, W. ,and Weber, N. E. Reliability, Availability, and Serviceability (RAS) of the IBM eServer z990. IBM Journal of Research and Development, Nov, 2004.

[11] J. S. Plank and W. R. Elwasif, “Experimental assessment of workstation failures and their impact on checkpointing systems,” in 28th International Symposium on Fault-Tolerant Computing, June 1998.

[12] N. H. Vaidya, “Impact of checkpoint latency on overhead ratio of a checkpointing scheme,” IEEE Transactions on Computers, vol. 46 ,Aug. 1997.

[13] K. Li, J. F. Naughton, and J. S. Plank, “Low-latency, concurrent checkpointing for parallel programs,” IEEE Transactions on Parallel and Distributed Systems, vol. 5, Aug. 1994.

[14] J. S. Plank, J. Xu, and R. H. Netzer, “Compressed differences: An algorithm for fast incremental checkpointing,” Tech. Rep. CS-95-302, University of Tennessee at Knoxville, Aug. 1995.