NASA GRANT REPORT

Miroslaw Malek

Department of Electrical and Computer Engineering,
The University of Texas at Austin,
Austin, Texas 78712.

December 14, 1990

The proliferation of increasingly powerful and complex multiprocessor systems has made fault-tolerant design a necessity. Optimizing fault tolerance in multiprocessor systems is a very difficult task because it involves multi-dimensional tradeoffs. The system architecture, the computation structure, the implementation technology, the frequency, duration and location of faults, and many other factors all have certain impact on the effectiveness of a particular fault tolerant approach. In our research, we have attempted to look at different areas in fault tolerance and have tried to integrate them under one umbrella. A unified approach to fault-tolerance is perhaps the only solution that may succeed for the difficult task of redundancy management. Such an approach covers design for fault tolerance, testing, reconfiguration and recovery.

Our main focus in this research has been on efficient reconfiguration procedures. Previous works in this area are somewhat diverse and ad hoc, and there is a lack of analytical studies to evaluate the existing fault tolerance techniques and to guide future research. We have attempted to solve this difficult problem by a graph theoretic approach. In our research, we have introduced this approach and concentrated on the analysis and optimization of fault tolerance in multiprocessor systems.

Specifically, a reconfiguration model that allows a faulted job to be recovered with minimum space and time overhead and without performance degradation has been formally introduced [1]. This allows the execution of jobs on a multiprocessor system with predictable behavior. Additionally, eleven parameters have been precisely defined to facilitate the evaluation of the fault tolerance of different multiprocessor systems for executing a given set of target applications [1]–[3]. These parameters also allow the quantitative comparison of various fault reconfiguration techniques so that efficient algorithms can be developed. The graph theoretic approach used is widely applicable to multiprocessor systems and applications with various topologies. We have concentrated on two well-known systems, namely, the mesh and the hypercube, and two frequently used computational structures, namely, the path and the complete binary tree. Solutions and algorithms for determining various optimization parameters have also been presented. Algorithms that allow optimized job reconfiguration are also developed. More importantly, we have studied the applications of the analytical approach to the fault-tolerant design of multiprocessor systems. Our approach explores the inherent fault tolerance of multiprocessor systems and exploits the topological relationship between the system architecture and the target applications. Hence our approach not only leads to good reconfiguration

procedures but also helps one in designing and selecting a good architecture for fault tolerance based on the requirements of the target application.

Besides studying fault reconfiguration schemes we have also directed our efforts to fault recovery approaches. We have attempted to characterize the utilization of fault recovery in such a way as to introduce minimum performance degradation. In particular we have studied the conditions for applicability of efficient low-cost fault recovery techniques, such as forward recovery, to deterministic problems [4]. The approach we used for that was to determine specific classes of applications that met those conditions. More specifically, we have explored the natural redundancy existing in the variables of these programs in order to recover from faults without the need of extensive recomputation. Rather than roll back, due to the existence of this naturally embedded redundancy, the state of a computation plagued by faults from a defined set of faults can be utilized to reconstruct a correct state from which the execution of the program can continue.

Any fault-tolerant approach cannot be complete without efficient testing procedures to weed out faulty components after manufacture or during the operation of the system. To accomplish this aim, a new concept called *topological testing* has been introduced [5]. Topological testing uses graph theoretic optimization methods such as the Traveling Salesman Problem, the Chinese Postman Problem, coloring, covering, matching, partitioning and path covering to minimize test time. The topological testing techniques can be applied to test a system's behavior and its organization at each level of the system's hierarchy, namely, circuit, logic, register transfer, instruction and processor-memory-switch levels. Specifically, the topological testing approach is demonstrated by developing tests for the multistage interconnection network, the hypercube network and the mesh network [5]–[8]. Time optimization for the testing of these networks gives very promising results by taking advantage of inherent parallelism and removing test redundancy. Considerable improvement in testing time is achieved by applying topological testing techniques to the testing of these networks.

It has been observed that a lot of problems require efficient heuristics for obtaining good solutions since they fall in the category of NP-complete problems. Taking this fact in consideration we have developed a new hybrid algorithm technique for improved performance based on the idea of mixing two or more algorithms [9]. This hybrid algorithm has an inherent potential for parallelization and has been observed to obtain good results for the Traveling Salesman Problem for which two heuristics namely Simulated Annealing and Tabu Search were hybridized [10]. Comparison of the algo-

rithm with each individual algorithm indicates consistently better results. A similar approach towards parallelizing a single algorithm, where multiple instances of execution of the algorithm with different parameters are mixed, has also shown very encouraging results.

Our current research has been directed towards introducing fault tolerance in real-time systems. These fault-tolerant real-time systems, called *responsive systems* [11] are used for very critical applications, and are applicable to NASA's future Space Station. Redundancy Management to obtain fault tolerance in such system is a challenging task due to the additional constraints of real-time and criticality of application. Our approach is towards a comprehensive design of such systems including specification, modeling and design for redundancy management and recoverability. Our research in 1989/90 resulted in nine publications, three research reports and two book chapters. We hope that our research will contribute to facilitating redundancy management in NASA's distributed computing systems.

References

[1] Harary F. and M. Malek, "Quantifying Fault Recovery in Multiprocessor Systems", Technical Report, Department of Electrical and Computer Engineering, The University of Texas at Austin.

[2] Yau, K., "The Analysis and Optimization of Fault Tolerance in Multiprocessor Systems: A Graph Theoretic Approach", Ph.D. Dissertation, Department of Electrical and Computer Engineering, The University of Texas at Austin.

[3] Malek, M. and K. Yau, "The Resiliency Triple", Proceedings of the 1988 International Conference on Parallel Processing, 351-358, Chicago, August 1988.

[4] Laranjeira, L. and M. Malek, "Modeling Forward Recovery in Responsive Systems", Technical Report, Department of Electrical and Computer Engineering, The University of Texas at Austin.

[5] Malek, M., A. Mourad and M. Pandya, "Topological Testing, Proceedings of the 1989 International Test Conference, 103-110, August 1989.

[6] Malek, M., A. Mourad, B. Ozden and M. Pandya, "Topological Testing", Working Paper, submitted to IEEE Transactions on CAD.

[7] Malek, M. and B. Ozden, "Optimized Testing of Meshes", Proceedings of the 1990 International Test Conference, 627-637, September 1990.

[8] Mourad, A., B. Ozden and M. Malek, "Comprehensive Testing of Meshes", To appear in IEEE Transactions on Computers.

[9] Malek, M., M. Guruswamy, H. Owens and M. Pandya, "Serial and Parallel Search Techniques for the Traveling Salesman Problem", Annals of Operations Research, 21, 59-84, 1989.

[10] Malek, M., M. Guruswamy, H. Owens and M. Pandya, "A Hybrid Algorithm Technique", Working Paper, also Technical Report, Dept. of Computer Sciences, The University of Texas at Austin, TR-89-06, 1989.

[11] Malek, M., "Responsive Systems, A Challenge for the Nineties", Proceedings of Euromicro '90, Sixteenth Symposium on Microprocessing

and Microprogramming , Keynote Speaker, Amsterdam, The Netherlands, North-Holland, Microprocessing and Microprogramming 30, 9-16, August 1990.

[12] Malek, M. and E. Opper, "The Cylindrical Banyan Multicomputer: A Reconfigurable Systolic Architecture", Journal of Parallel Computing, 10, 319-327, 1989.

[13] Fussell, D., S. Rangarajan and M. Malek, "Built-in Testing of Integrated Circuit Wafers", IEEE Trans. on Computers, C-39 (2), 195-205, February 1990.

[14] Fussell, D., S. Rangarajan and M. Malek, "Wafer-Scale Testing/Design for Testability", in the book titled *Wafer Scale Integration,* E. E. Swartzlander (ed.), Chapter 9, 413-472, Kluwer, Boston, 1989.

[15] Fussell, D., S. Rangarajan and M. Malek, "Fault Diagnosis of Linear Processor Arrays", *Defect and Fault Tolerance in VLSI Systems,* I. Koren (ed.), Chapter 6, 149-160, Plenum Press, New York, 1989.

QUANTIFYING FAULT RECOVERY IN MULTIPROCESSOR SYSTEMS

Frank Harary* and Miroslaw Malek**

*Computing Research Laboratory
New Mexico State University
Las Cruces, New Mexico 88003
e-mail address: harary@nmsu.edu

**Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712
e-mail address: malek@ngp.UTEXAS.EDU

ABSTRACT: We formalize and quantify various aspects of reliable computing with emphasis on efficient fault recovery. The mathematical model which proves to be most appropriate is provided by the theory of graphs. We have developed new measures for fault recovery and observe that the value of elements of the fault recovery vector depend not only on the computation graph H and the architecture graph G, but also on the specific location of a fault. In our examples we choose a hypercube as a representative of parallel computer architecture, and a pipeline as a typical configuration for program execution. We define dependability qualities of such a system with or without a fault. These qualities are determined by the resiliency triple defined by three parameters: multiplicity, robustness and configurability. We also introduce parameters for measuring the recovery effectiveness in terms of distance, time, and the number of new, used, and moved nodes and edges.

Key Words: Fault recovery, resiliency, multiprocessor systems, hypercube, pipeline

1 INTRODUCTION

Reliable and fault-tolerant design is becoming increasingly important with the ever-growing demand for more sophisticated and complex computer systems. Simple reliability evaluations of several research and commercial parallel systems indicate that fault tolerance is becoming a *sine qua non* condition in any multiprocessor. In fact, parallel and distributed computers offer a unique opportunity of trading performance for reliability. In order to accomplish that, sophisticated fault detection, fault location, and fault recovery algorithms are required. Our objective is to introduce a formal approach based on graph theory to fault recovery problems. Most previous works concentrated on fault diagnosis (detection and location) [1, 2, 3, 4]. The study of fault recovery has been relatively neglected. Very little has been accomplished with the notable exception of the work by Yanney and Hayes [5]. Our objective is to formalize and quantify various aspects of reliable computing and give examples of fault recovery of pipelines imbedded in hypercube architectures.

2 FAULT RECOVERY MODELS

In general, two problems occur: analysis and synthesis. In the first problem area, analysis, a prescribed graph architecture such as a grid (mesh) or hypercube is given and all fault considerations must be handled within that framework. On the other hand for a given set of conditions to be realized, a synthesis problem necessitates the construction of an appropriate architecture.

In an excellent pioneering work, Hayes [6] proposed a fundamental synthesis problem for graphs: Given a job H, construct a supergraph G with the minimum possible number of additional nodes and edges such that after the removal of any k nodes from G, the resulting subgraph still contains H. The cases studied take H as a path, a cycle, or binary tree. A graph G with this property is called *k-fault-tolerant* with respect to the job H.

Another problem of synthesis that we propose for both architecture and computation graph models could be stated as follows: Find a graph G with a minimum number of additional nodes and

edges, such that after the removal of any k nodes, the reconfiguration time is required to restore the original subgraph corresponding to a job is minimized.

In analysis the problem is different and can be described as follows. Let G be a given graph and let H be a subgraph of G. When a node u is removed from H, there are two possibilities:

Either G – u contains another subgraph H′ isomorphic to H or it does not. If it does not, then the system G is called *non-recoverable* with respect to H and the node u. On the other hand, when G - u does contain a subgraph H′ isomorphic to H, we require the minimum cost (such as some function of distance, time, or utilization of new resources) to produce such a subgraph H′. There is often more than one way to map H onto H′. One or more of these ways will yield the required minimum distance by the following procedure. The *distance* d (H,H′,f) *with respect to an isomorphism* f from H onto H′, where f(v) = v′, is equal to the sum of the distances in G of d(v,v′) for all the nodes v in H. Now we can precisely define the *distance* d (H,H′), namely, it is the minimum of the distances d(H,H′,f) taken over all isomorphisms f between H and H′.

Another interesting analysis problem pertinent to fault-tolerant computing can be stated as follows. Given a graph G and a subgraph H, determine the maximum number of node-disjoint embeddings of H onto G. We will call this parameter the *multiplicity* of H in G and denote it by m=m(G,H). In graph theory [7], this is known as the node-disjoint packing number pac_0 (G,H) as introduced in [8]. In [9] both pac_0 (Q_n, Q_m) and the corresponding edge-disjoint packing number pac_1 (Q_n,Q_m) of subcubes in hypercubes are studied.

Our fault-tolerant computing philosophy in multiprocessors is that if the redundant resources are available we should use them for the execution of a given job. When m = 2, we can effectively detect any single fault by comparing the outputs of two jobs while for $m \geq 3$ we can mask a single fault by voting on the outputs. This dynamic duplication or triplication could also be used for single fault recovery by exploiting redundancy in time [10] where stages of a systolic FFT (Fast Fourier Transform) array, say, can be executed on different processors each time. In this case the problem

can be stated as follows.

Given a labeled graph G and a subgraph H, find multiple embeddings of H onto G with different labels of G for each node of H. The number of such executions of a given job H in a graph G we call the *robustness* r = r(H,G) of an execution of a job H.

The same parameters m and r can be defined for a system in the presence of faults, and then m becomes the number of embeddings of H onto G – u (when u corresponds to a faulty node) exist, while r becomes the number of embeddings of H onto G – u having different labels of G – u for each element of H. We denote these parameters m′ and r′. Thus m′ = m(H,G – u) and r′= r(H,G – u).

Finally, the *configurability* c corresponds to the number of ways that a particular job H can be configured or embedded in system G or on a subgraph of the system after the removal of a faulty node u, i.e., c = c(G,H) is the number of different types of subgraphs of G, with respect to the automorphism group of G. In other words, c(G,H) is the number of orbits in the group of G modified to act on all subgraphs of G isomorphic to H as in Harary and Palmer [11]. After the removal of the faulty node, we have c′ = c(G – u,H). We will refer to (m, r, c) as the *resiliency triple* of system (G,H) because it characterizes such fault-tolerant qualities of the system.

3 FAULT RECOVERY VECTOR (FRV)

We have just defined the resiliency triple for quantifying fault tolerance in systems with and without faults. We have considered only a single fault, call it node u, in a job H embedded in a system graph G. This will extend readily to multiple faults. We now introduce several parameters in addition to the multiplicity m, the robustness r, and the configurability c which furnish various optimization criteria for efficient fault recovery.

These parameters for a given recovery procedure involve distance, time, the number of new nodes, the number of used nodes, the number of new edges, the number of used edges, and what will be called the eviction order and eviction size.

d = *distance*

d = d(H,u,G) has already been defined above. It is the minimum of d(H,H′), which was defined as $\Sigma d(v,v')$ among all subgraphs H′ of G isomorphic to H but not containing the known fault, u.

t = *time*

t = t(H,u,G) is the maximum value of the terms d(v,v′) in the above sum.

By *recovery effectiveness* we mean the distance, time pair (d,t).

ν_0 = *number of new nodes*

ν_0 is the number of new nodes that were added in order to configure the job H′ equivalent to H, the one that was at fault.

μ_0 = *number of used nodes*

μ_0 is the number of nodes of G which were traversed while moving from faulty job H to faultless job H′ but do not appear in $H \cup H'$. These nodes are not incorporated in the resulting subgraph of the recovered job.

ν_1 = *number of new edges*

ν_1 is to edges as ν_0 is to nodes; it is the smallest number of new edges in the new recovered job that were not used in the original faulty job.

μ_1 = *number of used edges*

This number μ_1 is simply the number of edges that are required for mapping the recovered job. These used edges enable the transfer of the data from H to H′ and do not occur in

$H \cup H'$.

The quadruplet of parameters $(\nu_0, \mu_0, \nu_1, \mu_1)$ is called the *recovery overhead.*

e_0 = *eviction order*

During a recovery procedure there are certain nonfaulty nodes which can remain in their original locations in G. These are called the *stationary nodes* of this procedure in G – u. A node which must be moved in order to reconfigure is called *evicted* as it is not stationary. We define e_0 as the number of evicted nodes in any reconfiguration procedure of H in G – u.

e_1 = *eviction size*

This parameter is defined just by changing every word "node" above to "edge". We now have eight invariants which we wish to minimize:

$$d, t;\ \nu_0, \mu_0, \nu_1, \mu_1;\ e_0, e_1$$

It is convenient to separate these into three groups: the (d,t) pair determines the recovery effectiveness, the node-edge vector $(\nu_0, \mu_0, \nu_1, \mu_1)$ represents recovery overhead, and now the eviction pair (e_0, e_1) is called the *relocation measure.* The remaining invariants form the resiliency triple (m, r, c) and can be used to evaluate the level of system fault-tolerance in the presence or absence of faults.

The vector consisting of eight parameters obtained by concatenating the recovery effectiveness (d,t), the recovery overhead $(\nu_0, \mu_0, \nu_1, \mu_1)$, and the relocation measure (e_0, e_1) is called the *fault recovery vector* (FRV). (As in any graph theoretical extremal problem, it is difficult to optimize simultaneously with respect to all parameters, but it is conceivable that depending on the system user requirements we can find optimal solutions with respect to some of the eleven parameters listed above).

Although we have restricted the above formulation to the case of a single fault node u, all the

parameters extend at once to more general faults. For example, there might be a single fault which is an edge, or multiple faults which correspond to a set of nodes, a set of edges, or a mixed set of nodes and edges. We illustrate fault recovery vectors for the special case when G is a hypercube Q_n, H is a path P_k with k nodes, and u is one of the nodes of H.

4 FAULTS IN PIPELINES IN HYPERCUBES: EXAMPLES AND SIMPLE RESULTS

We now illustrate our concepts by numerous examples. We will show how the choice of multiprocessor architecture, a job graph, the location of a fault, and a recovery procedure affect the recovery parameters.

For purposes of illustration, we will use an 8-processor hypercube, Q_3, as our multiprocessor system and we will imbed in that system the pipelines represented by path P_4 and path P_5.

In Figure 1 we demonstrate how to determine a resiliency triple, starting with the multiplicity m. As there are eight processors in Q_3, only two jobs requiring 4-processor paths (pipelines) can be packed and hence executed simultaneously provided there are no faults. Therefore, the multiplicity $m = 2$.

The next parameter of P_4 in Q_3 is robustness. Here $r = 8$ as can be verified from the definition of r by sliding P_4 along a hamiltonian cycle C_8 containing this path. This necessarily gives different labels to each node. The last parameter in the resiliency triple is the configurability c which is the number of different ways that P_4 can be mapped onto Q_3. In the hypercube Q_3 it is apparent that there are only two distinct ways of mapping P_4. Therefore, $c = 2$. In general finding c is a difficult problem and requires separate treatment [12]. In the following examples, we will illustrate all eight parameters in the fault recovery vector, FRV.

In Figures 2 through 6 we show a path P_4 imbedded in Q_3 in different ways with different faults and different recovery strategies. In the example in Figure 2 we show a path P_4 with a faulty

node A at the beginning of the path. The recovery procedure shown in three stages in Figure 2 minimizes the overhead on additional resource utilization, where the distance d = 2, the time t = 2, only one additional node and edge are needed, and only a single node and a single edge were evicted.

Another way to recover this pipeline P_4 with a faulty beginning node is shown in Figure 3. Here the fault recovery vector requires a high number of four additional nodes and four additional edges. Also the number of used edges and evicted nodes and edges is equal to 4. What is optimized though is the time which is minimum, t = 1.

In Figures 4 and 5 we show the same example P_4 with B as the faulty node. The differences here are significant. In Figure 4 the recovery overhead is minimized while in the second case illustrated in Figure 5 it is the recovery effectiveness which is optimized.

Next, in Figures 6, 7, and 8, we extend the path order to 5 and show the reconfiguration of P_5 in the presence of faulty nodes A, B, C, respectively. Notice that for each case, the recovery vectors are quite different except for time.

We have demonstrated how a given architecture graph, pipeline graph, fault location, and recovery procedure may impact recovery and fault tolerance parameters.

Having defined the resiliency triple and the fault recovery vector, we now proceed with a generalized analysis of resiliency parameters in a hypercube Q_n with embedded pipeline P_k.

Given a binary n-cube Q_n with $p = 2^n$ nodes and $q = n2^{n-1}$ edges, and a pipeline of length k represented by a path P_k, the following straightforward results were obtained.

4.1 Multiplicity

(1) $m(Q_n, P_k) = \lfloor 2^n/k \rfloor$

Proof. Since Q_n contains a hamiltonian path containing all $p = 2^n$ nodes, the number of paths of order k which are node-disjoint and hence correspond to the multiplicity m is as stated in (1).

4.2 Robustness

(2) $r(Q_n, P_k) = 2^n$

Proof. Since Q_n contains a hamiltonian cycle, every node of P_k can be mapped on every node of Q_n by sliding a path P_k along the hamiltonian cycle.

4.3 Configurability

In general the problem of finding c is difficult and special algorithms are required [12]. Some special cases can be solved. For example for paths in Q_3 we have

$c(Q_3,P_3)=1$ $\quad$ $c(Q_3,P_6)=4$

$c(Q_3,P_4)=2$ $\quad$ $c(Q_3,P_7)=3$

$c(Q_3,P_5)=3$ $\quad$ $c(Q_3,P_8)=2$

It is interesting to observe that even in such a simple example the configurability grows monotonically to 4 and then monotonically decreases to 2. For paths P_3, P_4, and P_5 in Q_4, c=1, 1, and 2 respectively.

We have shown some results on the resiliency triple and observe that the value of elements of the FRV vector depend not only on H and G, but also on the specific location of a fault. (We need only analyze faults in P_k in nodes numbered from 1 to $\lceil k/2 \rceil$ because all the remaining cases in the pipeline are symmetrical and have a buddy among the nodes from 1 to $\lceil k/2 \rceil$ except for $\lceil k/2 \rceil$ when k is odd).

5 CONCLUSIONS

We formalize and quantify various aspects of reliable computing with emphasis on efficient fault recovery, a relatively neglected area of fault-tolerant computing. We define dependability qualities of parallel systems with or without a fault. These qualities are determined by the resiliency triple defined by three parameters: multiplicity, robustness and configurability. We also introduce parameters measuring the recovery effectiveness in terms of distance, time, the number of new, used, and moved nodes and edges. In our examples we choose a hypercube as a representative of

parallel computer architecture and a pipeline as a typical configuration for program execution. We confine the examples to faulty nodes, but, of course, faulty edges should also be considered.

This comprehensive framework for fault recovery should aid computer designers and users in comparing quality of parallel architectures, job configurations, and job allocations with respect to the efficiency of fault recovery. We have also obtained some basic results for finding optimal multiplicity and robustness parameters for a pipeline in a hypercube. The main contribution of this paper is the fact that the term "efficient fault recovery" has been formally quantified and this work establishes quantifiable goals in search for efficient recovery algorithms.

REFERENCES

[1] F.P. Preparata, G. Metze, and R.T. Chien, "On the connection assignment problem of diagnosable systems," *IEEE Trans. on Electronic Computers*, EC16, 1967, pp. 848-854.

[2] C.R. Kime, "An analysis model for digital system diagnosis," *IEEE Trans. on Comp.*, C-19, 1970, pp. 1063-1070.

[3] A.D. Friedman and L. Simoncini, "System-level fault diagnosis," *IEEE Trans. on Comp.*, C-13, 1980, pp. 47-53.

[4] G.J. Lipovski and M. Malek, *Parallel Computing: Theory and Comparisons*, Wiley, New York, 1987.

[5] R.M. Yanney and J.P. Hayes, "Distributed recovery in fault-tolerant multiprocessor networks," *IEEE Trans. on Computers*, 871-879, vol. C-35, Oct. 1986.

[6] J.P. Hayes, "A graph model for fault-tolerant computing systems," *IEEE Trans. on Computers*, 875-883, vol. C-25, Sept. 1976.

[7] F. Harary, *Graph Theory*, Addison-Wesley, Reading, 1969.

[8] F. Harary, "Covering and packing in graphs, I." *Annals N.Y. Acad. Sci.* 175, pp. 195-208, 1970.

[9] N. Graham and F. Harary, "Packing, mispacking and covering hypercubes with subcubes," *Computers and Mathematics with Applications*, to appear.

[10] Y.-H. Choi and M. Malek, "A fault-tolerant FFT processor," *Digest of Papers of the 15th*

Fault-Tolerant Computing Symposium, 266-271, June 1985.

[11] F. Harary and E.M. Palmer, *Graphical Enumeration*, Academic Press, New York, 1973.

[12] M. Malek and K. Yau, "Configurability in multiprocessor systems," Technical Report, Dept. of Elect. and Comp. Engr., University of Texas, 1987.