submitted to ITC 96

Fault Detection and Diagnosis in Multiprocessor Systems with Point-to-Point Connections*

Krishnendu Chakrabarty, Mark Karpovsky and Lev Levitin

Research Laboratory for Design and Testing of Computer Systems
Department of Electrical, Computer and Systems Engineering
Boston University
44 Cummington Street
Boston, MA 02215

March 1996

Contact author: Krishnendu Chakrabarty
Phone: (617) 353-1235
FAX: (617) 353-6440
Email: kchakrab@hilsa.bu.edu

ABSTRACT

We describe a practical software-implemented system-level testing technique for multiprocessor systems with dedicated connection links. We address the detection and diagnosis of processor, link, and hybrid faults of arbitrary multiplicity, and investigate the testing of several multiprocessor topologies including trees, hypercubes and meshes.

Keywords: Error-correcting codes, fault detection, fault diagnosis, graph theory, monitors, multiprocessors, system-level testing.

Topic area: System test.

---

*This research was supported by the National Science Foundation under grant no. MIP 9208487, by NATO under grant no. 910411, and by a start-up grant from Boston University's College of Engineering.

Fault Detection and Diagnosis in Multiprocessor Systems with Point-to-Point Connections

Krishnendu Chakrabarty, Mark Karpovsky and Lev Levitin

SUMMARY

We describe a practical software-implemented system-level testing technique for multiprocessor systems with dedicated communication links. The testing is carried out using test patterns that are obtained from a test generation program residing in the local memories of a small number of selected processors, called monitors. We address the detection and diagnosis of processor, link, and hybrid (combination of processor and link) faults and investigate the problem of minimizing the number of monitors and determining their placement for several multiprocessor topologies including trees, hypercubes and meshes. A major advantage of this approach is that the detection of all three fault types of arbitrary multiplicity is achieved. While a link fault of any multiplicity is diagnosable, we analyze the diagnosability of processor and hybrid faults and show that an exceptionally large fraction of these faults are also diagnosable.

1 Introduction

Multiprocessor systems must be throughly tested to ensure their reliable operation. The testing process, which includes fault detection and location (diagnosis), should be quick as well as effective. The fault detection and diagnosis problems have been the subject of intensive research in the past and studied under the general area of system-level diagnosis [1, 2, 15, 16, 21, 27]. Traditional system-level diagnosis techniques model the multiprocessor system as a diagraph, termed the test graph, whose vertices denote processors and an edge (p_i, p_j) from processor p_i to p_j indicates that p_i tests p_j. Connection assignment refers to the problem of determining the structure of the test graph required to diagnose all faults in a given fault set. The connection assignment problem and related diagnosability problems have received considerable attention in the past. However, the fault model in system-level diagnosis has been limited to processor faults (link faults have not been considered explicitly) and practical applications of these diagnosis methods are few and limited [5]. Most previous research focusses on processor faults on the assumption that because of circuit complexity, processor faults are more likely than link faults. However, a survey of fault occurrences

in Tandem Computers's fault-tolerant multiprocessors reveals that link faults cause twice as many system failures as processor faults [25].

In this paper, we present a practical system-level fault detection and diagnosis approach that provides a unified methodology for processor, link, and a combination of processor and link faults in mutiprocessor systems with point-to-point, dedicated communication links. We assume that the faults are permanent, i.e., they remain indefinitely when no corrective action is taken. A major advantage of this method is that it guarantees the detection of all faults, diagnosis of all single processor faults and link faults, and the diagnosis of a large fraction of processor and link faults. As the number of processors and links increases in multiprocessor systems, faults of higher multiplicity become more likely. Therefore, any practical and scalable system-level testing technique should explicitly address faults of high multiplicity. Another advantage of this method is that a high degree of diagnosis of faults of large multiplicity is implicitly achieved using a testing strategy aimed at fault detection.

Our proposed approach is based on minimizing the number of tester processors and relies on software-implemented functional testing. We assume that all processors are identical and the links connecting them are bidirectional. We also assume that the processors contain local memory and use message-passing for communication. The testing is carried out using functional test patterns $x_1, x_2, \ldots, x_T$ that are obtained from a test generation program $\mathcal{P}$ residing in the local memories of a small number of selected processors, called monitors. Techniques for generating such functional test patterns have been developed in the past [6, 13, 20, 28, 29]. This approach is motivated by the fact that a functional test for a processor may be very long, requiring excessive local memory for storage. On the other hand, it may be possible to generate these patterns algorithmically using a small program $\mathcal{P}$. This is especially the case if we use pseudorandom patterns for testing [3, 22, 23]. This allows us to trade-off testing time with the amount of local memory that has to be allocated for testing.

We model a multiprocessor system as an (undirected) graph $G = (V, E)$, where V is the set of processors and E is the set of links in the system. Testing is carried out in an off-line mode by exploiting the fact that a subgraph G' of G may be idle during computations, and G' can perform self-testing during such idle periods. Every monitor tests itelf and all its neighboring processors. We address the problem of selecting monitors such that by using "balls" of radius one centered on the monitors, we can detect and diagnose processor and link faults in the system.

Our design objective is to minimize the number of monitor units in any given multiprocessor system. The test program $\mathcal{P}$ has to reside on the local memory of every monitor processor. The amount of memory required to store $\mathcal{P}$ consitutes the main overhead of our system-level testing scheme, therefore we reduce this overhead by minimizing the number of copies of $\mathcal{P}$. By minimizing the number of monitors, we also minimize the hardware/software required to determine the correctness of the test responses. Traditional system-level diagnosis approaches do not guarantee a minimum number of monitors.

We solve the monitor-placement problem for three different faults types:

1. Processor faults, where we assume that k_P processors in the system are faulty.
2. Link faults, in which k_L communication links in the system are assumed to be faulty.
3. Hybrid faults, where k_P processors and k_L links in the system are faulty.

The solution to the monitor-placement problem is different for processor and link faults. Detecting and diagnosing link faults typically requires more monitors. However, we show later that if we place monitors for the detection and diagnosis of link faults, then we are guaranteed the detection of all processor and hybrid faults. In addition, we achieve complete diagnosis of all single processor faults as well as high diagnosability of processor and hybrid faults of arbitrary multiplicity.

We derive several topology-independent bounds on the number of monitors for these problems. We also develop construction methods for the placement of monitors for a number of multiprocessor topologies, including trees, meshes, and hypercubes. Many of these bounds and monitor-placement methods are based on results from graph theory and error-correcting codes. The organization of the paper is as follows. In Section 2, we describe our testing procedure. In Sections 3, we solve the monitor-placement problem for processor faults and analyze the diagnosability of multiple processor faults. Section 4 addresses monitor placement for link faults. In Section 5, we show that monitor placement for link faults provides complete processor fault detection and high degree of processor and hybrid fault diagnosis. Finally, in Section 5, we examine issues related to testing time and outline ongoing research.

2 Testing procedure

In this section, we describe our procedure for testing and diagnosis of processor, link, and hybrid faults. We assume that the multiprocessor system is homogeneous, i.e., the processors in the system are identical.

Testing procedure:

1. The monitors, selected using methods describe in Sections 3 and 4, execute the test program $\mathcal{P}$ and generate test patterns $X = x_1, x_2, \ldots, x_T$, where T is the length of the test set.

2. Every monitor executes a self-test routine using X. As part of the self-test routine, it executes a function $f(X)$ using the test patterns. The function f is implemented in software and stored in the local memory of every processor.

3. Every monitor regenerates X (the test set is not stored because it may be too long) and broadcasts the tests to its neighboring processors.

4. Every idle processor executes a self-test routine by computing $f(X)$.

5. The monitors collect the test responses $f(X)$ from their neighboring processors and compare them one by one to their own copy of $f(X)$. They then forward the result of the comparisons to a host computer.

We assume that the test set is sufficiently long and effective such that a fault in the system leads to an observable error. We also assume that the errors are not masked (aliased) either in the processors or on the (faulty) links.

Using this testing procedure, we can detect all processor, link, and hybrid faults. This is useful in a number of multiprocessor systems, for example Tandem's fault-tolerant computers, which employ built-in hardware redundancy [25]. Such systems typically contain replicated processors which are reconfigured for normal operation on as soon as a fault is detected. An effective low-cost fault detection strategy is especially desirable in these applications. In addition to fault detection, we can diagnose all link faults and all single processor faults with our testing procedure. A small number of processor and link hybrid faults are not diagnosable, but we show later this number is extremely small.

3 Processor faults

In this section, we show that we can detect and diagnose processor faults using a small number of monitors. Monitor placement for processor faults also detects a considerable number of link faults. (However, additional monitors are required to detect and locate all link faults; this is discussed in Section 4.)

The monitor-placement problem for processor faults is stated as follows: find the minimum number of monitor processors and their location in the multiprocessor system such that every processor is contained in at least one of the balls of radius one around the monitor. Let T_P be the number of monitors required to detect processor faults in the system. The *diameter* D of a graph G is the length of the maximum of the shortest paths between any pair of vertices in G [9]. A graph G is *d-regular* if every vertex in G has degree d.

Theorem 1 *The following lower bounds on T_P can be easily derived:*

1) $T_P \geq D/3$, where D is the diameter of the graph G.

2) $T_P \geq K$ such that $\sum_{i=1}^{K}(d_i + 1) \geq N$, where N and d_i are the number of nodes and the degree (number of neighbors) of node v_i in G, respectively, and $d_1 \geq d_2 \geq \cdots \geq d_N$.

3) For a d-regular graph, $T_P \geq N/(d+1)$.

Proof: We first prove 1). Let the diameter of G correspond to the shortest path p between vertices u and v and let $T_P = m < D/3$. If all these monitors lie on the path p then they cannot cover all the processors in the system. On the other hand, if at least one of these monitors does not lie on p then there exists a path between u and v of length less than D. This is a contradiction because D is the length of the shortest path between u and v.

The proof of 2) follows from the fact that the greedy method of placing monitors first on nodes of higher degrees provides a covering of all the processors in the system only when the balls of radius one do not overlap. The bound 3) is derived from 2) since each ball now includes $d + 1$ processors. □

The bound $N/(d + 1)$ corresponds to a perfect node cover (perfect processor testing), and

is achieved when every processor is covered by exactly one monitor in the system. A necessary condition for a perfect cover is that the minimum distance between any two monitors is three. We show later that this lower bound is achieved for a number of topologies.

We next consider the monitor-placement problem for processor faults in various specific topologies.

3.1 Trees

A number of hierarchical computing systems such as dictionaries and search machines can be modeled as a tree [4, 26]. Many parallel algorithms can be mapped on to p-ary tree, and the architecture of a general-purpose multiprocessor can often be modeled by a tree structure [18]. Another application of a tree structure is the data network of the Thinking Machines CM-5 [12, 17].

We consider a p-ary tree with l levels. Let $T_P(l)$ be the number of monitors required for processor testing. The following recurrence relation, illustrated in Figure 1, provides a method to compute the number of monitors required for processor testing.

$$T_P(l) = (p-1)T_P(l-1) + p^2 T_P(l-3) + 1 \quad (1)$$

The base cases of (1) are $T_P(1) = T_P(2) = 1$, $T_P(3) = p$, and $T_P(4) = p^2 + 1$. We do not yet have a closed-form solution for (1). An alternative recurrence relation for $W_N^T(l)$ is given below:

$$T_P(l) = 1 + p^2 T_P(l-2) \quad (2)$$

The recurrence relation (2) can be easily solved to obtain the following closed-form expression:

$$T_P = \begin{cases} \dfrac{p^{l-3}-1}{p^2-1} + p^{l-2} & \text{, if } l \text{ is odd;} \\ \dfrac{p^l-1}{p^2-1} & \text{, if } l \text{ is even.} \end{cases}$$

A lower bound on the number of monitors, assuming a perfect cover with no root or leaf node as a monitor is given by $T_P \geq \dfrac{p^l-1}{(p-1)(p+2)}$. If the number of levels is a multiple of three, then it is possible to approach this lower bound asymptotically, i.e., for large values of p. In such cases, we place monitors on levels $l-1$ (the leaves are at level l and the root is at level 1), $l-4, \ldots, 2$. The number of monitors is given by

$$T_p = p^{l-1} + p^{l-4} + \cdots p$$

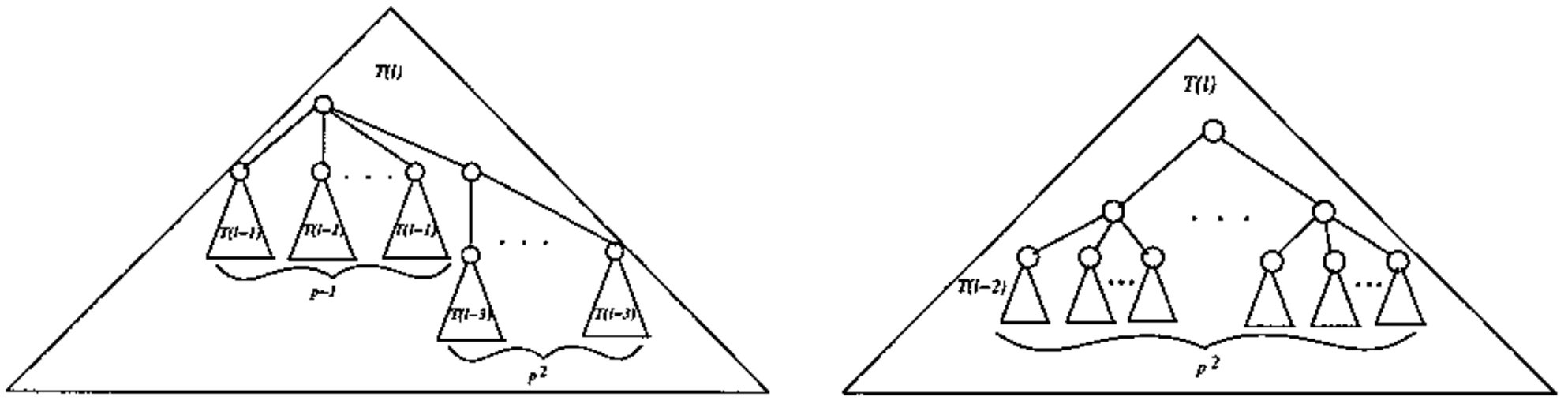


Figure 1: Recurrence relation formulations for determining the number of monitors for processor testing of trees: (a) recurrence (1) and recurrence (2).

p	Number of levels l	No. of monitors T_p	Total no. of processors N	T_P/N
2	5	10	31	0.32
	6	18	63	0.29
	7	40	127	0.31
	8	81	255	0.32
3	5	28	121	0.23
	6	84	364	0.23
	7	253	1093	0.23
	8	820	3280	0.25

Table 1: Number of monitors required for processor testing of trees.

$$= \frac{p(p^l - 1)}{p^3 - 1}$$

$$= \frac{p^l - 1}{(p-1)(p+1+1/p)} \tag{3}$$

It follows from (3) above that T_p is very close to the lower bound $\frac{p^l - 1}{(p-1)(p+2)}$ and approaches it as p increases.

Table 1 shows the number of monitors required for processor faults in a tree topology. These results demonstrate that processor faults can be detected and diagnosed in trees using a small number of monitors.

A *star* is a rooted tree with two levels. It can be easily seen that a star can be tested for all processor faults with the root node as the only monitor. Processor faults of arbitrary multiplicity are detected and faults involving the leaf nodes are diagnosed.

3.2 Hypercubes

A hypercube or binary d-cube computer is a multiprocessor system with $N = 2^d$ processors interconnected as a d-dimensional binary cube. Each processor P_i constitutes a node of the cube and is a self-contained computer with its own CPU and local memory. Each P_i also has direct communication paths to d other neighbor processors through the edges of the cube. An example of a commercial hypercube computer is the NCUBE/ten, which is a 10-dimensional system developed by NCUBE Corporation [10, 14].

Since every processor of a d-dimensional hypercube can be assigned a d-bit vector, the monitor-placement problem for third topology can be solved by finding a Hamming code with covering radius one that covers all d-bit vectors. It follows from coding theory that for the binary d-dimensional hypercube, perfect processor testing is achieved with $T_P = 2^n/(n+1) = 2^{n-m}$ monitors if and only if $n = 2^m - 1$, i.e., a perfect Hamming code exists. Figure 2(a) shows monitors (shaded) for the 3-dimensional hypercube, where the monitors are the processors labeled 000 and 111. If $n \neq 2^{m-1}$, the best solution to the monitor-placement problem is obtained using tables of the best (minimal) error-correcting codes with a covering radius of one [7]. Table 2 shows the number of monitors T_P (size of the minimal Hamming code) required for processor faults in hypercubes.

We next give an example of the monitor selection procedure for the 7-dimensional hypercube.

d	$N = 2^d$	Lower bound on T_P	T_P (best known)
3	8	2	2
4	16	4	4
5	32	7	7
6	64	12	12
7	128	16	16
8	256	32	32
9	512	55	62
10	1024	105	120
11	2048	177	192
12	4096	342	380
15	32768	2047	2047

Table 2: The number of monitors T_P for hypercubes.

Since $7 = 2^3 - 1$, it is possible to find a perfect Hamming code with 7 bits, and hence a set of monitors that provide perfect processor testing. We generate a perfect (7,4) Hamming code as follows. The generator matrix G for this code is

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

The 16 codewords (monitors) are generated by performing the operation $x \cdot G$ over $GF(2)$ using all 16 4-bit vectors x. With $x \in \{0000, 0001, \ldots, 1110, 1111\}$, we obtain the following labels for the 16 monitors in the system: 0000000, 0001101, 0010110, 0011011, 0100011, 1001010 11000, 1000111, 1001010, 1010001, 1011100, 1100100, 1101001, 1110010, 1111111.

3.3 Meshes

A mesh is a multiprocessor topology that finds extensive applications in parallel processing. A p-ary n-dimensional mesh has p^n processors and each processor is connected to its $2n$ neighbors. (every processor has two neighbors in each dimension.) Practical mesh architectures include 2-dimensional rectangular meshes such Intel's Paragon architecture [11], 3-dimensional meshes such as the MIT-Intel J-machine [8], and hexagonal meshes [24].

Theorem 2 *(a) For a p-ary (p prime) n-dimensional mesh, perfect processor testing can be achieved if and only if* $n = (p^m - 1)/2$, *where* m *is an integer greater than zero. (In this case,* $T_P = p^n/(2n+1) = p^{n-m}$*); (b) Perfect processor testing can be achieved for a hexagonal mesh* $(d = 3)$ *using* $T_P = N/4$ *monitors; (c) Perfect processor testing can be achieved for a triangular mesh using* $T_P = N/7$ *monitors.*

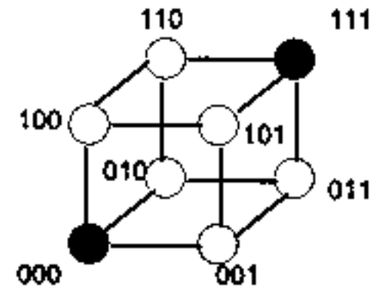
Proof: We first prove (a). Perfect processor testing for a p-ary n-dimensional mesh is achieved only if $T_P = N/(2n+1) = p^n/(2n+1)$. If p is prime, then $2n+1$ must equal p^m where $m > 0$. (If $m = 0$ then every processor is a monitor.) This proves necessity of (a). To prove sufficiency, we use a p-ary single-error correcting (SEC) Hamming code with check matrix H having m rows and n columns. The codewords of this Hamming code correspond to the monitors in the n-dimensional p-ary mesh, and because it is an SEC code, no noncodeword (monmonitor) is covered by (at distance one from) more than one codeword (monitor), i.e., the balls of radius one around the monitors are nonoverlapping. The number of codewords, and hence monitors, is p^{n-m}. These p^{n-m} monitors cover a total $p^{n-m}(2n+1) = p^n$ processors, and since the balls around the monitor are nonoverlapping, every processor in the system is covered.

The proofs for (b) and (c) follow from the monitor-placement shown in Figure 2. □

In Figure 2(b), we show monitor placement for perfect processor testing of hexagonal meshes. Figure 2(d) shows perfect processor testing of a triangular mesh.

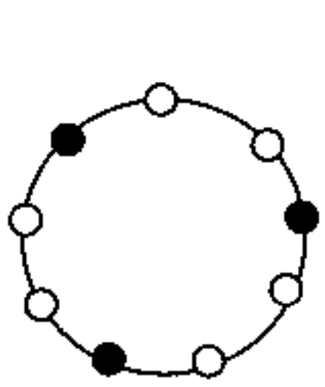
The theorem implies that for a 2-dimensional mesh such as Intel's Paragon, perfect processor testing can only be achieved if $p = 5$. This is shown in Figure 2(c) for a mesh with 25 processors. (We assume, as is generally the case, a toroidal mesh, i.e., the ends of the mesh wrap around.) Figure 2(a) shows perfect processor testing for the ring toplogy. In general, if we represent the processors in a p-ary n-dimensional mesh by coordinates $(x_1, x_2, \ldots, x_n)$, where $0 \leq x_i < p$, then the location of the monitors can be obtained by solving the equation $\sum_{i=1}^{n} x_i = 0 \pmod{p}$.

Another topology for which perfect processor testing is achieved is the ring; see Figure 2(b). A ring with N processors requires $\lceil N/3 \rceil$ monitors.

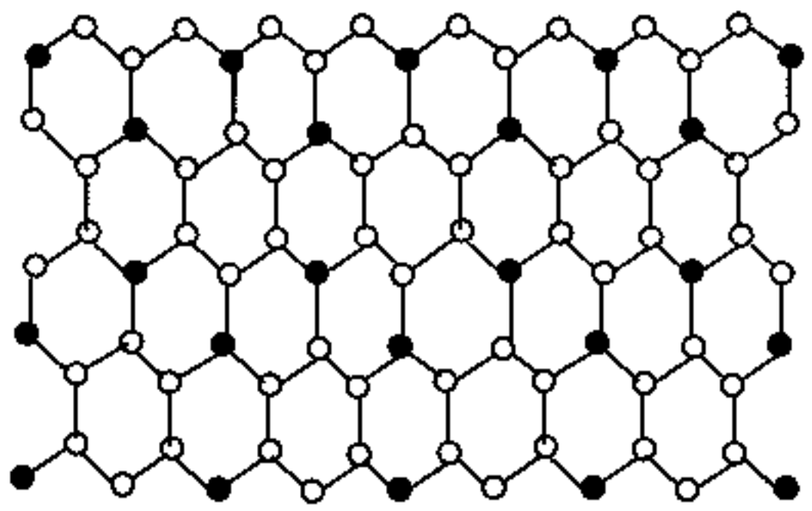


(a)

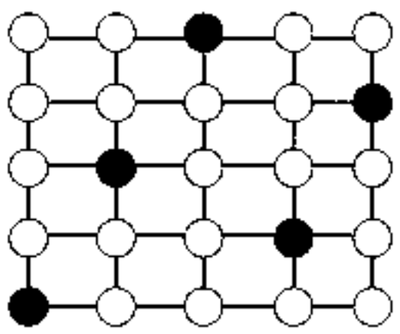
● Monitor



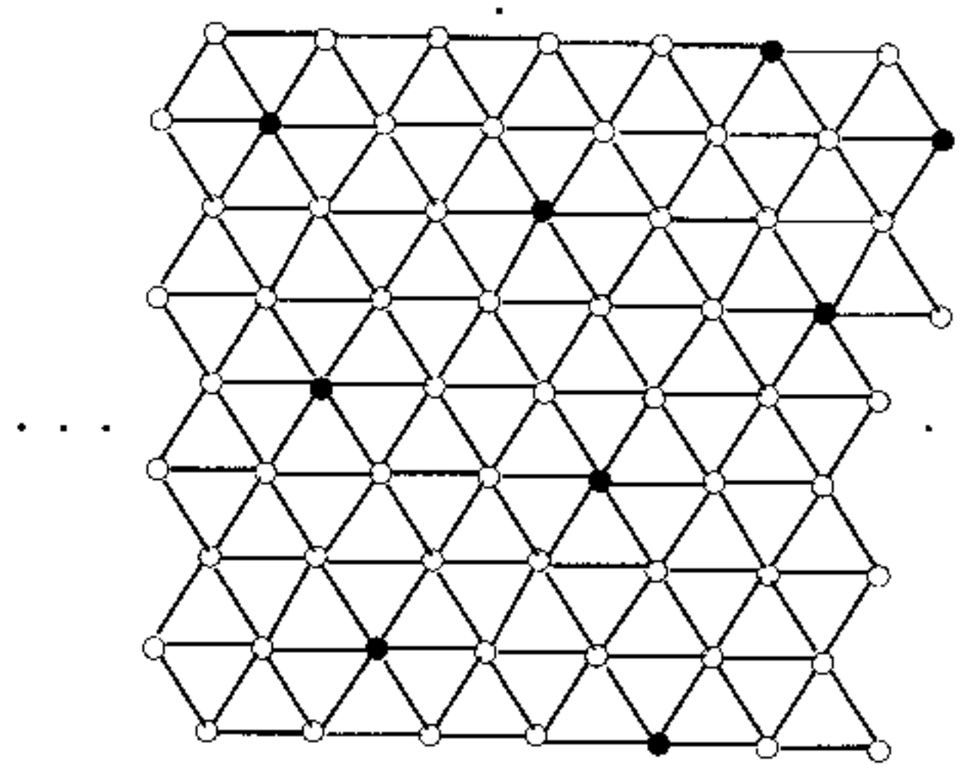
(b)



(c)



(d)



(e)

Figure 2: Perfect processor testing for (a) 3-dimensional cube, (b) ring, (c) hexagonal mesh, (d) 2-dimensional mesh with 25 processors, (e) triangular mesh.

3.4 Diagnosis of multiple processor faults

We next determine the fraction of processor faults of multiplicity $k_P > 1$ that are diagnosed by our testing procedure. A processor fault involving k_P processors is not diagnosable if and only it includes at least one monitor and a neighboring nonmonitor processor. Hence the number of nondiagnosable faults is $T_P d\binom{N-2}{k_P-2}$. Therefore, the fraction f of processor faults of multiplicity k_P that are not diagnosable is given by

$$f = 1 - \frac{T_P d\binom{N-2}{k_P-2}}{\binom{N}{k_P}}$$

If we achieve perfect processor testing, then $T_P = N/(d+1)$, therefore

$$\begin{aligned} f &= 1 - \frac{\frac{Nd}{d+1}\binom{N-2}{k_P-2}}{\binom{N}{k_P}} \\ &\approx 1 - \frac{Nk_P(k_P-1)}{N(N-1)} \\ &\approx 1 - \frac{k_P(k_P-1)}{N} \end{aligned} \tag{4}$$

for large values of N and d.

For a 15-dimensional hypercube, Figure 3 shows the fraction of diagnosable processor faults. The coverage drops with an increase in k_P, nevertheless a coverage of over 95% is achieved for upto 30 processor faults.

4 Link faults

In this section, we examine the problem of detecting and diagnosing link faults. If monitors are placed in the multiprocessor system targetting only processor faults, then the number of links covered by the monitors is at most $T_p d$. Assuming perfect processor testing, this implies that the fraction of links covered by the monitors is only $\frac{Nd/(d+1)}{Nd/2} \approx \frac{2}{d}$. For large d, this is a significantly small fraction, hence to guarantee high coverage of link faults, we need to solve the monitor-placement problem explicitly for links.

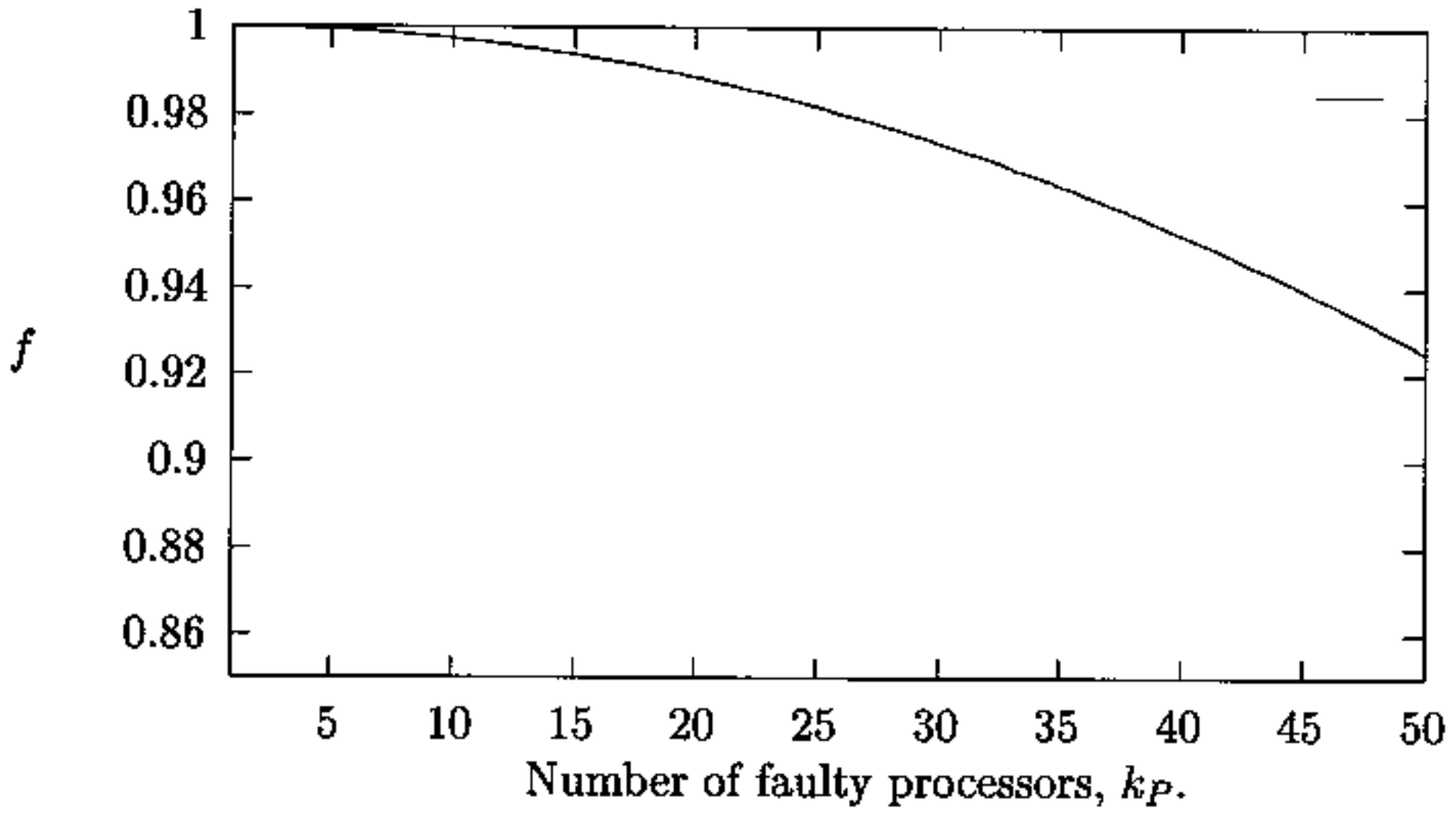


Figure 3: Fraction of processor faults of multiplicity k_P that are diagnosable for a 15-dimensional hypercube.

The monitor-placement problem for link faults is stated as follows: find the minimum number of monitor processors and their location in the multiprocessor system such that every link is contained in at least one of the balls of radius one around the monitors. Let T_L be the number of monitors required for link faults in the multiprocessor system. In a d-regular graph, a ball of radius one around a monitor covers d links. Therefore, for a multiprocessor system with E links, *perfect link testing* is achieved with E/d monitors for a d-regular graph if every link is tested by exactly one monitor. This implies that $T_L \geq E/d$.

Theorem 3 *Perfect edge testing requires $N/2$ monitors.*

Proof: The number of edges E in a d-regular graph with N vertices is $Nd/2$. Therefore perfect link testing requires $E/d = (Nd/2)/d = N/2$ monitors. In other words, $T_L \geq N/2$. □

The monitor placement problem for link testing is related to the problem of determining the point covering number of the graph G. A vertex and an edge in G *cover* each other if they are incident, and the smallest set of vertices that covers all the edges in G is called the point covering number $\alpha(G)$ [9]. Clearly, $T_L = \alpha(G)$ and the processors in the smallest cover are selected as monitors.

A set of vertices in G is *independent* if no two of them are adjacent, and the number of vertices in the largest independent set is called the line independence number $\beta(G)$. The maximum independent set for $G = (V, E)$ can be constructed using the following procedure:

begin
S:= ϕ;
S':= V;
while S' is not empty **do**
 begin
  Pick an element u of S';
  $S' := S' - \{u\}$;
  if $(u, v) \notin E$ for all $v \in S$ **then**
    $S := S \bigcup \{u\}$;
end

The following theorem shows the relationship between $\alpha(G)$ and $\beta(G)$. It can be used to determine $\alpha(G)$, and therefore T_L, from $\beta(G)$.

Theorem 4 [9] *For any nontrivial connected graph G with N vertices, $\alpha(G) + \beta(G) = N$.*

Figure 4 shows monitor placement for link faults in various multiprocessor topologies. Perfect link testing is achieved for hypercubes, rings, and hexagonal meshes. For a p-ary n-dimensional mesh and the n-dimensional binary cube, the perfect monitor placement for link faults corresponds to a "'checkerboard" pattern and $T_L = N/2$. For a p-ary tree with n levels, an optimal monitor placement for link testing is achieved by selecting processors on alternate levels as monitors (Figure 4(a).

Theorem 5 *The optimal number of monitors for link faults in a p-ary tree with l levels is given by*

$$T_L = \begin{cases} \dfrac{p(p^{l-1}-1)}{p^2-1} & , \textit{if } l \textit{ is odd;} \\ \dfrac{p^l-1}{p^2-1} & , \textit{if } l \textit{ is even.} \end{cases}$$

Table 3 shows the number of monitors required for link testing of p-ary trees.

For a triangular mesh, it is not possible to achieve perfect link testing. Consider a subgraph of G with vertices $v_1, v_2.v_3$ such that $(v_1, v_2), (v_2, v_3), (v_1, v_3) \in E$. In order to cover all these three

p	Number of levels l	No. of monitors T_p	Total no. of processors N	T_P/N
2	5	10	31	0.32
	6	21	63	0.33
	7	42	127	0.33
	8	85	255	0.33
3	5	30	121	0.25
	6	91	364	0.25
	7	273	1093	0.25
	8	820	3280	0.25

Table 3: Number of monitors required for link faults in trees.

edges, we must select two of the vertices in the subgraph as monitors. This implies that at least one of the edges is covered by two monitors, hence perfect link testing is not achieved. This can be generalized to the following statement: perfect link testing is not possible in graphs that contain triangles.

An optimal monitor placement for link faults in a triangular mesh is shown in Figure 4(f). This placement, which requires $3N/4$ monitors can be proven to be optimal in the following way. Consider any two row of vertices in Figure 4(f) where alternate processors are monitors. This is the minimum number of monitors required for that row to cover all the links. It can now be easily seen that all processors on the adjacent rows must be monitors, otherwise, not all links between these rows are covered.

We next address the problem of placing monitors to detect and diagnose both processor and link faults.

5 Hybrid faults

A hybrid fault $\{k_P, k_L\}$ includes k_P faulty processors and k_L faulty links. Before investigating hybrid faults, we examine the detection and diagnosis of processor faults using monitors placed for link faults.

Theorem 6 *A set of monitors that covers every link provides complete detection of all single processor faults and diagnosis of all single processor faults.*

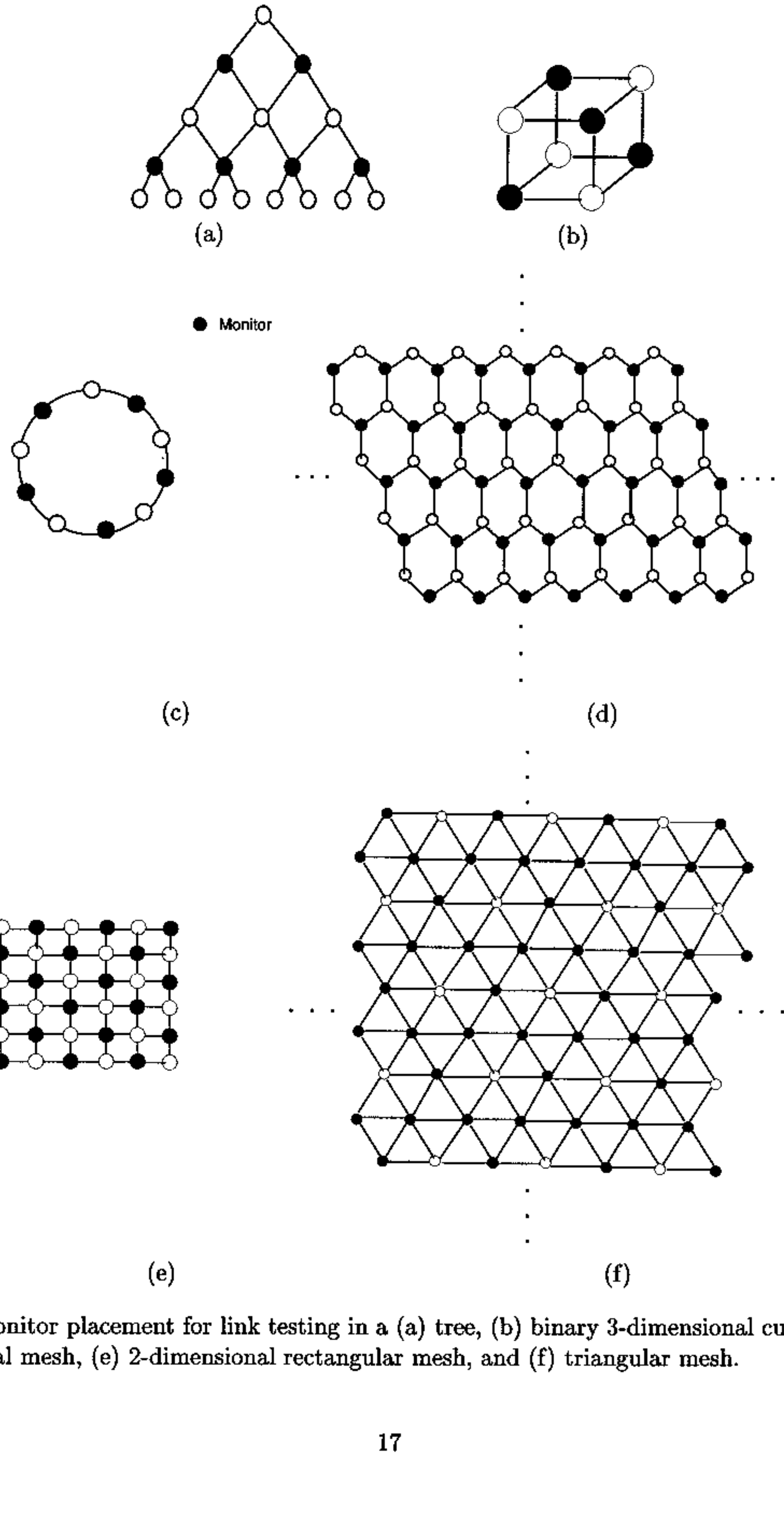


Figure 4: Monitor placement for link testing in a (a) tree, (b) binary 3-dimensional cube, (c) ring, (d) hexagonal mesh, (e) 2-dimensional rectangular mesh, and (f) triangular mesh.

Proof: Since the monitors all the processors in the system, it follows immediately that all processor faults are detected and all single processor faults are diagnosed. □

Monitor placement for link faults does not however provide diagnosis of all multiple processor and hybrid faults. For processor faults of multiplicity $k_P > 1$, we determine the fraction of diagnosable faults in Section 2. We next address hybrid faults. Let $C(k_P, k_L)$ be the fraction of faults involving k_P processors and k_L links that are diagnosable. We first note that $C(0, k_L) = 1$ because all link faults are diagnosable, and $C(k_P, 0) = 1 - \frac{k_P(k_P-1)}{N}$ for large values of N and d. We next address the problem of determining $C(1, k_L)$, which is a measure of the diagnosability of hybrid faults involving one processor and K_L links. The analysis is motivated by the fact that in many cases, link faults are more likely than processor faults [25].

To determine $C(1, k_L)$, we first count the number of hybrid faults $\{1, k_L\}$ involving one processor and k_L links that are diagnosable. A hybrid fault $\{1, k_L\}$ is diagnosable if no faulty link is incident on the faulty processor. Hence for regular graphs ($d_i = d$ for any i), there are $N\binom{L-d}{k_L}$ faults that are not diagnosable. The total number of $\{1, k_L\}$ faults is $N\binom{L}{k_L}$. Therefore, the fraction of these faults that are diagnosable is given by

$$\begin{aligned} C(1, k_L) &= \frac{N\binom{L-d}{k_L}}{N\binom{L}{k_L}} \\ &= \frac{\binom{L-d}{k_L}}{\binom{L}{k_L}} \end{aligned} \tag{5}$$

If $k_L = 0$, we get $C(1, 0) = 1$, which also follows from the fact that all single processor faults are diagnosable. For $k_L = 2$, we have $C(1, 1) = 1 - d/L = 1 - 2/N$ since $L = Nd/2$. For a 10-dimensional hypercube, $C(1, 1) > 0.99$, which implies that over 99% of faults involving a single processor and a single link are diagnosable. If $k_L << L$, then we simplify (5) as follows:

$$\begin{aligned} C(1, k_L) &= \frac{\binom{L-d}{k_L}}{\binom{L}{k_L}} \\ &= (1 - \frac{d}{L})^{k_L} \\ &= (1 - \frac{2}{N})^{k_L} \end{aligned} \tag{6}$$

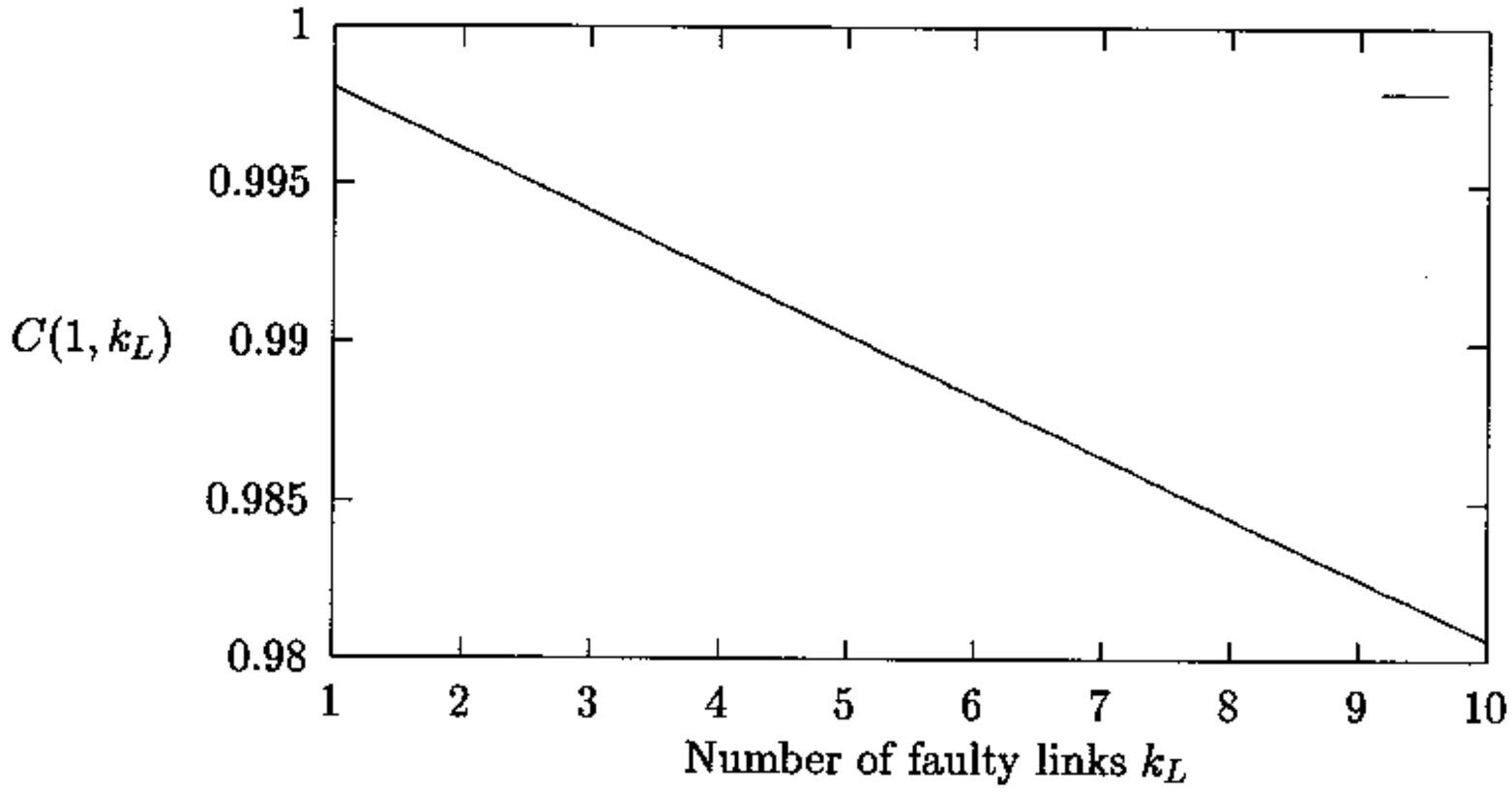


Figure 5: Diagnosability $C(1, k_L)$ of hybrid faults involving one processor and k_L links for a 10-dimensional hypercube.

Figure 5 shows the variation of $C(1, k_L)$ with k_L for a 10-dimensional hypercube machine such as the NCUBE/ten. Over 98% of faults involving one processor and as many as ten links are diagnosable. The range of k_L ($k_L \leq 10$) in the graph has been chosen has to satisfy the condition $k_L << L$. Note that the graph appears to be a straight line because $1 - 2/N$ is very close to one for this system.

6 Discussion

Finally, we estimate the time t required to detect and diagnose faults using the monitor placement strategy. First we assume a perfect cover, i.e., nonoverlapping balls of radius one. This corresponds to minimum testing time because of maximum parallelism in the test process. We assume that (i) T is the test length, (ii) it takes one cycle to generate each test pattern, (iii) the generation of the ith test pattern is done in parallel with the broadcast of the $(i-1)$th pattern, and (iv) it takes time L to compute $f(X)$. The generation of X at the monitors takes time T and the self-test of the monitors takes time L. Next, the execution of $\mathcal{P}$ on the monitors and the broadcast of the tests to the nonmonitor processors can be done in $T + 1$ cycles. The number of cycles required to compute $f(X)$ and send back the result to the monitor is $L + 1$. For a d-regular graph, it takes

an additional d cycles to compare the test responses at the monitors. Therefore, the total testing time is $t_{min} = T + L + T + L + 2 + d \approx 2(T + L)$.

An upper bound on the testing time t_{max} is $(Td + Ld) + d \approx 2(Td + Ld)$ because when the balls around the monitor overlap, a nonmonitor processor can receive test data X and compute $f(X)$ from only one monitor at a time. Therefore, $2(T + L) \leq t \leq 2(Td + Ld)$. The testing time in this system-level approach therefore depends only on the length of the test set and is independent of the number of processors in the multiprocessor system. Moreover, if a perfect cover is achieved and d is small, then the testing time is also independent of the number of links in the system.

We have described a technique to test multiprocessor systems for processor and link faults using a small number of tester (monitor) processors. We have presented optimal solutions to the monitor-placement problem for a number of practical multiprocessor toplogies. The monitors provide detection and diagnosis of single and multiple processor, link, and hybrid faults. In order to evaluate the coverage of low-level faults obtained with our approach, we are investigating the use of software tools for fault injection and coverage measurement. We are also examining distributed diagnosis algorithms and methods to determine the diagnosability of hybrid faults involving an arbitrary number of processors and links. One promising and practical decentralized diagnosis approach for dynamic reconfiguration is to use $N/2$ monitors in a "checkerboard" pattern as discussed in Section 4, and adopt a policy where the monitors disconnect links from which they do not receive correct test responses. Faulty processors are therefore effectively disconnected from the system. This approach allows correct system operation in the presence of processor, link, and hybrid faults of arbitrary multiplicity.

References

[1] D. R. Avresky et al. An approach to fault diagnosis of multimicrocomputer systems: algorithm and simulation. In *Proc. 1987 Int. Symp. Fault-Tolerant Computing*, pp. 305–310, 1987.

[2] D. R. Avresky and D. K. Pradhan (eds.) *Fault-Tolerant Parallel and Distributed Systems*, Computer Society Press, 1995.

[3] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-in Test for VLSI: Pseudorandom Techniques*. John Wiley, New York, 1987.

[4] J. Bentley and H. T. Kung. A tree machine for searching problems. In *Proc. 1979 Int. Conf. Parallel processing*, pp. 257–266, 1979.

[5] R. Bianchini, K. Goodwin and D. S. Nydick. Practical application and implementation of system-level diagnosis theory. In *Proc. 1990 Int. Symp. Fault-Tolerant Computing*, pp. 332–339, 1990.

[6] D. Brahme and J. A. Abraham. Functional testing of microprocessors. *IEEE Transactions on Computers*, vol. 33, pp 475–485, June 1984.

[7] G. D. Cohen et al. Covering radius 1985–1994. Tech. report, Department Informatique, Ecole Nationale Superieure des Telecommunications, France, 1994.

[8] W. J. Dally et al. The message-driven processor: a multicomputer processing node with efficient mechanisms. *IEEE Micro*, vol. 12, pp. 23–39, April 1992.

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

[10] J. P. Hayes et al. A microprocessor-based hypercube supercomputer. *IEEE Micro*, vol. 6, pp. 6–17, October 1986.

[11] R. M. Hord. *Parallel Supercomputing in MIMD Architectures.* CRC Press, Boca Raton, FL, 1993.

[12] W. D. Hillis and L. W. Tucker. The CM-5 connection machine: a scalable supercomputer. *Communications of the ACM*, vol. 36, pp. 31–40, 1993.

[13] M. G. Karpovsky and R. G. Van Meter. A practical approach to testing microprocessors. In *Proc. 1984 Design Automation Conference*, pp. 186–202, 1984.

[14] D. Jurasek, W. Richardson and D. Wilde. A multiprocessor design in custom VLSI. *VLSI Systems Design*, pp. 26–30, June 1986.

[15] C. R. Kime. System diagnosis. In *Fault-Tolerant Computing: Theory and Techniques*, vol. 2, D. K. Pradhan (ed.), Prentice-Hall, New Jersey, 1986.

[16] J. Kuhl and S. M. Reddy. Distributed fault tolerance for large multiprocessor systems. In *Proc. 1980 Int. Symp. Computer Architecture*, pp. 23–30, 1980.

[17] C. E. Leiserson. Fat trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, vol. 34, pp 892–901, 1985.

[18] C. A. Mead and L. A. Conway. *Introduction to VLSI Systems.* Addison-Wesley, Reading, Mass., 1980.

[19] S. F. Nugent. The iPSC/2 direct-connect communications topology. In *Proc. 1988 Conf. Hypercube Concurrent Computers and Applications*, pp. 51–60, 1988.

[20] R. Parthasarathy, S. M. Reddy and J. G. Kuhl. A testable design of general purpose microprocessors. In *Proc. 1982 Int. Symp. Fault-Tolerant Computing*, pp. 117–124 , 1982.

[21] F. P. Preparata, G. Metze and R. T. Chien. On the connection assignment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, vol. EC-16, pp. 848–854, December 1967.

[22] J. Savir and P. H. Bardell. On random pattern test length. *IEEE Transactions on Computers*, vol. C-33, pp. 467–474, June 1984.

[23] J. Savir, G. S. Ditlow, and P. H. Bardell. Random pattern testability. *IEEE Transactions on Computers*, vol. C-33, pp. 79–80, January 1984.

[24] K. G. Shin. HARTS: A distributed real-time architecture. *IEEE Computer*, pp. 25–35, May 1991.

[25] D. P. Siewiorek and R. S. Swarz. *Reliable Computer Systems–Design and Evaluation*, 2nd ed. Digital Press, Bedford, Mass., 1992.

[26] A. K. Somani and V. K. Agarwal. An efficient unsorted VLSI dictionary machine. *IEEE Transactions on Computers*, vol. 34, pp. 841–852, September 1985.

[27] A. K. Somani, D. Avis and V. K. Agarwal. A generalized theory for system-level diagnosis. *IEEE Transactions on Computers*, vol. 36, pp. 538–546, May 1987.

[28] T. Sridhar and J. P. Hayes. A functional approach to testing bit-sliced microprocessors. *IEEE Transactions on Computers*, vol. 30, pp 563–571, August 1981.

[29] S. M. Thatte and J. A. Abraham. Test generation for microprocessors. *IEEE Transactions on Computers*, vol. 29, pp 429–441, June 1980.