Fault Tolerant Subcube Allocation in Hypercubes

Yeimkuan Chang and *Laxmi N. Bhuyan*
Department of Computer Science, Texas A&M University
College Station, Texas 77843-3112

Abstract – *The subcube allocation problem in faulty hypercubes is studied in this paper. An efficient method for forming the set of regular subcubes is proposed. A concept of irregular subcubes is then introduced to take advantage of the advanced switching techniques such as wormhole routing to increase the size of available subcubes. In this paper, a two-phase fault tolerant subcube allocation strategy is proposed. The first phase is the reconfiguration process based on a modified subcube partitioning technique which finds the set of disjoint subcubes in the faulty hypercube. The second phase is to apply an existing fault-free subcube allocation strategy such as Buddy strategy to each disjoint subcube for assigning the fault-free available subcubes to the incoming tasks. The simulation results using Buddy strategy are also given.*

1 Introduction

Hypercube architectures have received much attention due to their attractive properties, such as regularity, scalability, fault tolerance, and multitasking capability. As the size of the system grows, the probability of some processors or links failing in the system becomes larger. In a normal situation, many parallel programs are executed concurrently in different subcubes allocated by the host system. In this paper, we study how to allocate subcubes in a faulty hypercube and thus maintain the multitasking capability of the system.

The main idea in assigning subcubes to incoming tasks is to avoid the interference among different tasks. This allocation problem of hypercubes has been studied in the literature. It was introduced by Chen and Shin [1]. Many other researchers use different approaches to tackle the problem[2, 3, 4]. Parallel algorithms for hypercube allocation have been developed in [5, 6]. The full subcube recognition ability and the efficiency of the allocation algorithms are the two main issues in hypercube allocation. In this paper, we address this allocation problem for a system with faults.

The straightforward method to accomplish fault tolerant subcube allocation is to use the concept of *exclusion*. In other words, the faulty processors are treated as being allocated permanently. To improve on the above approach, Jokanovic et al. [7] propose a two-phase fault tolerant gray code (FTGC) strategy. FTGC is based on the observations that in a fault-free hypercube, the left half of the system becomes more fragmented than the right half since the GC strategy takes a first-fit approach. Therefore in FTGC, the first phase is to find a parameter sequence [1, 7] such that the distribution of the faulty processors is concentrated to the left as much as possible. The second phase is to apply the same allocation and deallocation procedures of the GC strategy.

We consider wormhole routing in addition to store-and-forward routing. The wormhole routing employed in current commercial n-cube systems [8, 9] has the property that communication delay is insensitive to the physical distance provided that there is no interference on the communicating links. We provide results both with and without wormhole routing property. It is assumed that all the faults are static and are detected before the reconfiguration algorithm starts. We assume only node faults and as a result, the links incident on the faulty nodes are also faulty. The faulty nodes can not perform either computation or communication.

We present a modified subcube partitioning strategy and discuss how to construct the subcubes in a hypercube with two faults, since in the worst case, two faults in an n-cube are sufficient to destroy every possible regular $(n-1)$-cubes. The technique first selects a regular $(n-1)$-cube and then the faulty processor in the selected $(n-1)$-cube is replaced with its corresponding healthy processor in the other $(n-1)$-cube. We then extend the results to more than 2 faults and show that at least $\lceil \frac{n}{2} \rceil$ faults can be tolerated while maintaining a fault-free $(n-1)$-cube [10]. We carry out extensive simulation to find the probability of a healthy $(n-1)$-cube in an n-cube system as a function of the number of faults. It is shown that the modified subcube partitioning technique performs better than the method which only searches for the regular subcubes.

Finally, we use the above modified subcube partitioning technique to develop a two-phase fault-tolerant subcube allocation strategy. The first phase is to construct the set of disjoint subcubes in a faulty hypercube according to our modified subcube partitioning technique. The second phase is to apply an existing fault-free subcube allocation strategy such as Buddy strategy to each disjoint subcube. We simulate our fault tolerant allocation strategy based on Buddy strategy. The simulation results show that the fault tolerant subcube allocation strategy based on the modified subcube partitioning is always better than the others.

The rest of the paper is organized as follows. An efficient construction method of regular subcubes is proposed in section 2. In section 3, the concept of irregular subcube is introduced. The modified subcube partitioning technique is proposed to take advantage of wormhole routing technique. In section 4, a two-phase fault tolerant subcube allocation strategy is presented. Simulation results are given in section 5. Finally, the concluding remarks are given in the last section.

2 Construction of Regular Subcubes

We first present an efficient method to construct the set of disjoint subcubes in a faulty hypercube. The construction of the set of disjoint regular subcubes (SDS) is

Proceedings of the International Conference on Parallel Processing (ICPP'93)
0-8493-8983-6/0-7 $20.00 © 1993 IEEE

described in the procedure Form_SDS as follows.

```
Procedure Form_SDS($n, F$)
/* $n$: size of hypercube and $F$: set of faults */
begin
    $d$ := Form_Regular_n_1_cube($n, F, F_0, F_1$);
    if $|F_0| = 0$ then
        SDS := SDS $\cup *^{n-i-1}0*^i$;
    else Form_SDS($n-1, F_0$);
    if $|F_1| = 0$ then
        SDS := SDS $\cup *^{n-i-1}1*^i$;
    else Form_SDS($n-1, F_1$);
end
```

The procedure Form_Regular_n_1_cube finds a dimension d at which the n-cube is divided into two $(n-1)$-cubes such that the difference of the numbers of the faulty nodes in these two $(n-1)$-cubes is the maximum. If there is a tie, we will solve it as follows. Let Q_d be the $(n-1)$-cube in which the number of faults is less than that in the other $(n-1)$-cube. We define the *weight* of a faulty processor in a hypercube as the number of neighbors which are also faulty and the *system weight* of the hypercube as the sum of the weights of all the faulty processors in the same hypercube. Let the system weight of Q_i be W_i. The dimension i is selected if $W_i \geq W_j$ for all $j \in \{0,..,n-1\}$. If there is a tie for W_i again, we arbitrarily select one dimension.

Example 1: Fig.1 shows a 4-cube containing 5 faulty nodes marked as shaded circles. Fig.1(a) and (b) illustrates that the 4-cube is split into two 3-cubes at dimension 3 and 1, respectively. Thus dimension 3 is selected for split instead of dimension 1 since the system weight of Q_3 is 2 which is greater than the system weight, 0, of Q_1. After the dimension d is selected, the procedure Form_SDS is continued for each $(n-1)$-cube until the subcube contains no faults or all the processors in the subcube are faulty.

The time complexity of Form_SDS is $O(|F| \times n^3)$ since Form_Regular_n_1_cube takes $O(|F| \times n^2)$ time units for each dimension and there are n dimensions. We shall see that the split and select process does not guarantee to find the *maximum set of subcubes* (MSS) introduced in [2]. However, our method shown in procedure Form_SDS is efficient whereas finding the MSS is an NP-hard problem.

Let us analyze the worst case number of faults that procedure Form_SDS can tolerate in order to maintain a fault-free $(n-m-1)$-cube. After calling Form_SDS once, there exists an $(n-1)$-cube containing at most $\lfloor \frac{|F|}{2} \rfloor$ faulty nodes. Thus Form_SDS can continue m times until the finally selected $(n-m)$-cube contains only one fault, when $|F| \leq 2^m$. Obviously, a fault-free $(n-m-1)$-cube is available in an n-cube containing $|F| \leq 2^m$ faults, for $n \geq m+1$.

3 Construction of irregular subcubes

We introduce the concept of *irregular subcubes* to take advantage of the advanced routing technology such as wormhole routing currently used in the commercial hypercube multiprocessors [8, 9]. We define an ***irregular*** subcube as a hypercube in which there exists one or more pairs of logically adjacent nodes such that the physical distance between the adjacent nodes is more than one.

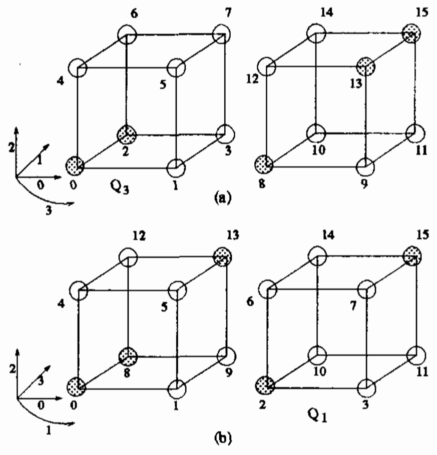

Figure 1: Construction of regular subcubes.

Due to wormhole routing, the communication speed between the logically adjacent nodes is the same as that between a set of physically adjacent nodes [8, 9]. A regular subcube can be embedded on an irregular subcube such that the load of the embedding is one and dilation of the embedding is equal to the maximum physical distance between the adjacent nodes.

We will briefly review the *modified*[1] subcube partitioning technique which constructs the irregular subcubes.

Modified Subcube Partitioning

The idea of modified subcube partitioning comes from how to utilize the unused links to achieve fault tolerance in the faulty hypercube. We shall see that the modified subcube partitioning technique tolerates more faults than only allocating regular subcubes.

The procedure of constructing an irregular subcube based on modified subcube partitioning is as follows. An $(n-1)$-cube is first selected in the faulty hypercube. Then the faulty processor in the selected $(n-1)$-cube is replaced by the corresponding processor in the adjacent $(n-1)$-cube. The replacing processor and its neighbors must work since the links between it and its neighbors are utilized in the reconfiguration process. We shall see that this reconfiguration process works as long as the nodes which are two or fewer hops from any faulty node of the selected $(n-1)$-cube, are healthy. The worst case of two faults being at antipodal positions can be solved by the modified subcube partitioning technique. The following example shows how the technique is applied to the 2-fault case.

Example 2: Assume there are two faulty nodes at antipodal positions of a 4-cube, i.e. (0000) and (1111) as shown in Fig.2. First, the node (1111) is replaced by node (0111). An irregular 3-cube is formed by the links shown as bold lines and the nodes marked with label 3 in parentheses. We call nodes 0011, 0101, and 0110 as the intermediate nodes of the irregular 3-cube since they reside in between node 0111 and its logical neighbors of the irregular 3-cube. If the intermediate nodes can be used for forming other subcubes, then one 2-cube consisting of nodes 0011, 0100, 0101, and 0110

[1] called *modified* in order to distinguish the subcube partitioning technique proposed in [11]

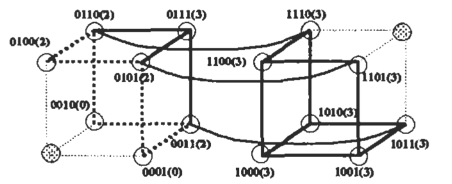

Figure 2: Subcube allocations of 4-cube with 2 faults.

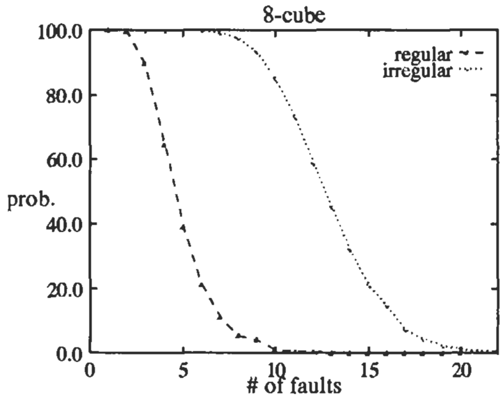

Figure 3: Probabilities of fault-free 7-cubes in an 8-cube with faults.

and two 0-cubes, 0001 and 0010, can be constructed as shown in the figure. However, if the intermediate nodes can not be used, the nodes 0001, 0010, and 0100 and the intermediate nodes can be used only as 0-cubes.

We call the assumption that the intermediate processors can not be used to form other disjoint subcubes to be *pessimistic*. However, in all the commercial hypercubes[8, 9], there are two kinds of processing units in a processor node, namely, a communication unit and a computation unit. Inside the communication unit, there is an $(n+1) \times (n+1)$ crossbar switch. For a message, there will be no delay in the switch if no other message competes for the same output. Thus the intermediate nodes can still be used for other tasks as long as there are no shared links with other tasks.

The time complexity of the modified subcube partitioning technique to find an $(n-1)$-cube is given as follows. Let f be number of faults. For any dimension i, the faulty nodes are divided into two groups. The faulty node whose i^{th} bit value is zero is assigned to group 1, otherwise it is assigned to group 2. Next, the Hamming distances of any fault in group 1 and any fault in group 2 are computed. Each Hamming distance takes n time units. There are at most $O(f^2)$ Hamming distances. Thus the time complexity leads to $O(f^2 \times n)$.

Simulation Results

Finding a fault-free $(n-1)$-cube in an n-cube containing a certain number of faulty nodes is the objective of our experiments. We carry out the simulation experiments by randomly generating faults in an n-cube. The probabilities of recognizing regular $(n-1)$-cubes (section 2) are compared with the probabilities of recognizing irregular and regular $(n-1)$-cubes based on the modified subcube partitioning technique.

Fig.3 shows the comparison of probabilities of recognizing fault-free 7-cubes in an 8-cube with 2 to 22 faults. We can see that when the number of faulty nodes exceeds 5, the probability of successful recognition of regular 7-cubes drops under 50%. The probability of successful recognition of regular 7-cubes drops to 0 when the number of faulty nodes reaches 10. However, the probability of successful recognition of fault-free irregular 7-cubes drops to 0 when the number of faulty nodes reaches 20.

4 Subcube Allocation

We have seen that $\lceil \frac{n}{2} \rceil$ faults can be tolerated in the worst case by the modified subcube partitioning technique while maintaining a fault-free $(n-1)$-cube. However, the simulation results show that in the average case, more faults can be tolerated in larger systems. In this section, we propose a two-phase fault tolerant subcube allocation strategy which allocates the regular and irregular subcubes. Since the faults do not occur as frequently as the processors are allocated in the system, we assume that the faults are detected and located before the construction starts.

The first phase is the construction of the set of disjoint subcubes by the modified subcube partitioning technique. The second phase is to sort the disjoint subcubes by their sizes in an increasing order and apply the existing fault-free subcube allocation strategy such as Buddy strategy etc. to each disjoint subcube. When an incoming task requesting a d-cube arrives in the system, starting from d-cube of the sorted disjoint subcubes, an available d-cube is searched and assigned to the incoming task.

Subcube Construction Phase

The algorithm Find_Disjoint_Subcubes for constructing the set of disjoint regular and irregular subcubes in a faulty hypercube is presented as follows. If the number of faults, $|F|$, is equal to the number of processors in the system then the process stops. Otherwise the procedure Find_n_1_cube described later is called to find an available $(n-1)$-cube. If an $(n-1)$-cube exists, the set F of faulty processors is updated. The number of faults remains the same and the fault-free $(n-1)$-cube is put in Disjoint_Set. Finally, Find_Disjoint_Subcubes is called recursively with the dimension decreased by one. If an $(n-1)$-cube does not exist , then Split_n_cube is called to find a dimension at which the n-cube is split into two $(n-1)$-cubes such that the difference between the numbers of faulty processors in the two $(n-1)$-cubes is maximum. If the sets of faults in the two $(n-1)$-cubes are in F_0 and F_1, then two Find_Disjoint_Subcubes are called recursively for F_0 and F_1.

```
Procedure Find_Disjoint_Subcubes($n$, $F$)
/* $n$: size of hypercube and $F$: set of faults */
begin
  if $|F| = 2^n$ then return;
  $Q_{n-1}$ := Find_n_1_cube($n$, $F$);
  if an $(n-1)$-cube $Q_{n-1}$ exists then
    Disjoint_Set := Disjoint_Set $\cup$ $Q_{n-1}$;
    Find_Disjoint_Subcubes($n-1$, $F$);
  else
    Split_n_cube($n$, $F$, $F_0$, $F_1$);
    Find_Disjoint_Subcubes($n-1$, $F_0$);
    Find_Disjoint_Subcubes($n-1$, $F_1$);
  endif
end
```

Find_n_1_cube first checks if there exists a regular $(n-1)$-cube, i.e. if there exists a dimension i such that the i^{th} bit values of the addresses of faulty nodes are all 0's or all 1's. If there exists an $(n-1)$-cube, the dimension i is put in P which is the set of dimensions that have been processed. The found $(n-1)$-cube is then returned. Otherwise, the irregular $(n-1)$-cube is searched according to the modified subcube partitioning technique. Hamming distances of addresses of any two faulty nodes are computed. For each pair of faulty nodes f_i and f_j whose Hamming distance is 1 or 2, we computes the dimensions spanning f_i and f_j and puts them into Dim_Set. If there exists a dimension i which does not belong to P and *Dim_Set*, then i is put in P and the $(n-1)$-cube is constructed by the procedure construct_n_1_cube(n, F, i) as follows. The n-cube is split into two $(n-1)$-cubes along the dimension i. One $(n-1)$-cube is selected and the faults in the selected $(n-1)$-cube is replaced with the healthy processors in the other $(n-1)$-cube. To update the set of faults, F, the i^{th} bits of the addresses of faults in the selected $(n-1)$-cube is complemented. The following example illustrates the construction process of procedure Find_n_1_cube.

Example 3: Based on Fig.2, there are two faults, 0 and 15 in the 4-cube. There is no regular 3-cube available since the two faults are at antipodal positions. Since the Hamming distance between the two faults is 4, any dimension from 0 to 3 can be selected for constructing the irregular 3-cube by procedure construct_n_1_cube.

When Find_Disjoint_Subcubes finishes, Disjoint_Set contains all the fault-free disjoint subcubes. The last step is to sort the Disjoint_Set by the size of the subcubes.

Subcube Allocation Phase
We gives the procedure FT_Subcube_Allocation, for fault tolerant subcube allocation as follows. The procedure checks each subcube in the set of disjoint subcubes, Disjoint_Set, from the smallest subcube to the largest subcube and uses the existing fault-free subcube allocation strategy to allocate an available subcube from the set of disjoint subcubes. The deallocation procedure does not change except that the subcubes are released to the disjoint subcube from which they were allocated.

```
Procedure FT_Subcube_Allocation($n$, $d$)
/* $d$: size of requested subcube*/
begin
    for all $Q_i \in$ Disjoint_Set, from $i = 1$ to
            $|Disjoint\_Set|$ do
        if $|Q_i| \geq d$ then
            $Q_d$ := Fault_Free_Subcube_Alloc($Q_i$, $d$);
            if $Q_d \neq$ Null then return ($Q_d$);
        endif
    endfor
    return (Null);
end
```

Example 4: Assume there are two faults in a 4-cube as in Fig.2. Procedure Find_Disjoint_Subcubes returns Disjoint_Set = $\{Q_0,Q_0,Q_2,Q_3\}$ or $\{Q_0,Q_0,Q_0,Q_0,Q_1,Q_3\}$ if the intermediate nodes such as nodes 0011, 0101, and 0110 can not be used for forming subcubes except being used as 0-cubes. Take Buddy strategy as the dedicated allocation strategy. Assume that the incoming task sequence is $\{1,1,0,3\}$. For former case, two 1-cubes in Q_2, a 0-cube in Q_0, and a 3-cube in Q_3 will be granted. However, for the later case, a 1-cube in Q_2, a 1-cube in Q_3, and a 0-cube in Q_0 are granted. The incoming task requesting a 3-cube needs to wait for the other tasks to finish.

5 Subcube Allocation Simulation

Performance based on the Buddy and single GC strategies is studied. We select the Buddy and single GC strategies because Buddy strategy is currently employed in the commercial hypercube multiprocessors and the single GC strategy is simple to implement. The following allocation strategies are adopted in the simulation.

1. the straightforward *regular* allocation strategy which treats the faulty processors as being allocated permanently,
2. the *modified regular* subcube allocation strategy which is based on the procedure Form_SDS proposed in section 2.
3. *irregular* subcube allocation strategy which is based on our two-phase fault tolerant subcube allocation strategy in section 4,
4. *pessimistic* irregular subcube allocation strategy which is same as the *irregular* except that the intermediate nodes can not be used to form other disjoint subcubes. However, the intermediate nodes can be assigned as individual 0-cubes.

The simulations on the two systems without any fault are also conducted for comparison. One follows the existing allocation strategy (called *no fault*) and the other is the two-phase fault-free allocation strategy called *no fault split*. The latter is motivated by the observation [7] that the left half of the system is more fragmented than the right half in the first-fit allocation approaches. The first phase is to split the n-cube into a set of subcubes, one k-cube for $1 \leq k \leq n-1$ and two 0-cubes. The second phase is the same as the one for irregular subcubes. Since there is no way we can service an incoming task which requests an n-cube or even an $(n-1)$-cube in a faulty hypercube, we will double (or quadruple) the residence time of the incoming task and reduce the subcube size by one dimension (or by two). This gives a fair comparison and indicates the impact of faults on the performance of the subcube allocation strategy.

In the experiments, K faults are generated 20 times with $K \geq 2$. For each K faults, 100 incoming tasks are generated and queued. The dimensions of the requested size by the incoming tasks are assumed to follow a given distribution such as uniform and normal distributions. The residence time of the allocated subcube is assumed to be uniformly distributed. In our experiments, the residence time is kept as $uniform(5, 11)$. Let p_i be the probability that an incoming task requests a subcube of size i, for $0 \leq i \leq D$ where D is the size of the system. Thus we have $\sum p_i = 1$. The p_i's($p_0..p_8$) for normal distribution used in our experiments are (.0228, .044, .919, .1498, .383, .1498, .919, .044, .0228).

The service discipline of the system is assumed to be first come first served (FCFS). At each time unit, the system attempts to find a fault-free subcube of the requested size to the first task in the queue and the assigned task is removed from the queue. After an incoming task in the system finishes, the subcube assigned to it is released. The process continues until all the 100 tasks

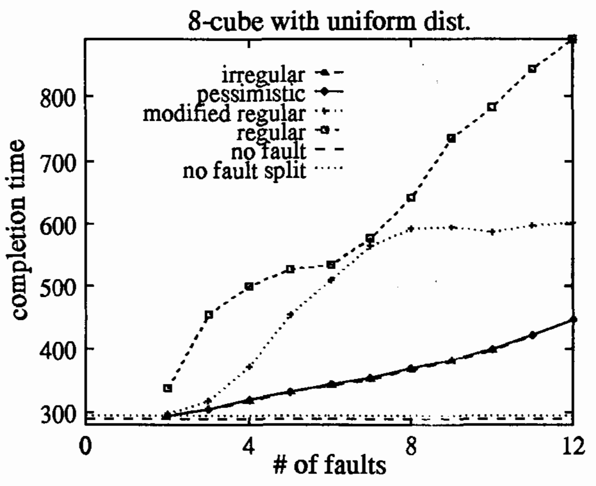

Figure 4: Completion time of Buddy strategy.

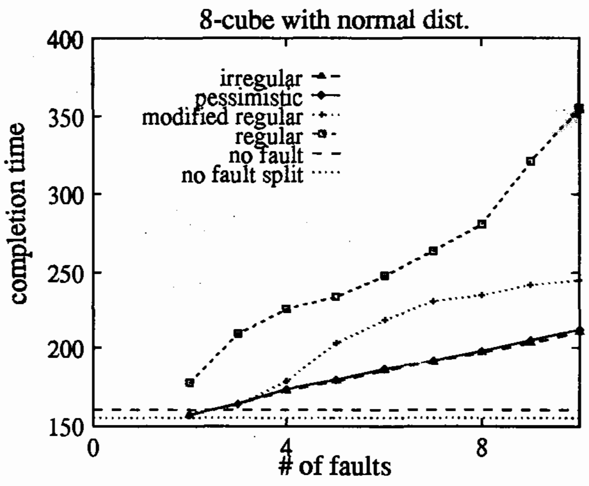

Figure 5: Completion time of Buddy strategy.

are finished. Under the above simulation model, the performance is measured in terms of completion time. For each K faults, 20 independent runs are performed. The average of these parameters for 20 runs are computed and are used in the plots.

Based on Buddy strategy, Fig.4 shows the completion time in the 8-cube with 2 to 12 faults and with uniform distribution of requested sizes. We can see that the performance of allocating irregular subcubes is always better than allocating regular subcubes. The *pessimistic* irregular approach is slightly worse than the *irregular* approach. This indicates that bigger subcubes play an important role in the allocation process since the *pessimistic* approach does not lose much bigger subcubes, especially the biggest possible available subcube, $(n-1)$-cube. Notice that *no fault split* approach does not work better than *no fault* approach since there is equal chance that an incoming task requests n-cube and *no fault split* approach assigns an $(n-1)$-cube with double the residence time. Fig.5 shows similar results.

6 Final Remarks

Wormhole routing, currently employed in commercial hypercube multiprocessors, is considered in constructing subcubes in the presence of faults. As long as the links of an irregular subcube assigned to one task are not shared by other subcubes, the performance of the task execution will be the same as the tasks running in the regular subcubes. This is because the subcubes generated by the modified subcube partitioning technique are disjoint. We developed a two-phase fault tolerant subcube allocation strategy which is general enough to apply to any existing fault-free subcube allocation strategy. The two-phase approach also improves the performance of larger fault-free hypercube systems. The extensive simulation results show that our approach always performs better than others.

References

[1] M. S. Chen and K. G. Shin, "Processor Allocation in an N-Cube Multiprocessor Using Gray Codes," *IEEE Transactions on Computers*, pp. 1396–1407, Dec. 1987.

[2] S. Dutt and J. P. Hayes, "Subcube Allocation in Hypercube Computers," *IEEE Transactions on Computers*, pp. 341–351, March 1991.

[3] J. Kim, C. R. Das, and W. Lin, "A Top-down processor Allocation scheme for hypercube Computers," *IEEE Transactions on Parallel and Distributed Systems*, pp. 20–30, January 1991.

[4] P. J. Chuang and N. F. Tzeng, "A Fast Recognition-Complete Processor Allocation Strategy for Hypercube Computers," *IEEE Transactions on Computers*, pp. 467–479, April 1992.

[5] Y. Chang and L. N. Bhuyan, "Parallel Algorithms for Hypercube Allocation," In *International Parallel Processing Symposium(IPPS)*, April 1993.

[6] M. Livingston and Q. F. Stout, "Parallel Allocation Algorithms for Hypercubes and Meshes," In *Proceedings of 4th Hypercube Concurrent Computers and Applications*, 1989.

[7] D. Jokanovic, N. Shiratori, and S. Noguchi, "Fault Tolerant Processor Allocation in Hypercube Multiprocessors," *Japan's IEICE Transactions*, vol. E 74, no. 10, pp. 3492–3505, Oct. 1991.

[8] nCUBE Corporation, *nCUBE 2 Processor Manual*, nCUBE Corporation, Dec. 1990.

[9] Intel, *Intel iPSC/2*, Intel Scientific Computers, 1988.

[10] Y. Chang and L. N. Bhuyan, "Constructing Subcubes in Faulty Hypercubes," Technical report, Texas A&M University, 1992.

[11] J. Bruck, R. Cypher, and D. Soroker, "Tolerating Faults in Hypercubes Using Subcube Partitioning," *IEEE Transactions on Computers*, pp. 599–605, May 1992.