

Prefix Computation on Faulty Hypercubes *

Jyh-Wen Mao and Chang-Biau Yang

Department of Applied Mathematics
National Sun Yat-sen University
Kaohsiung, Taiwan 804, China**Abstract**

In a parallel computer system, fault-tolerance is more obtainable since there are more than one processor cooperated in the system. And, for solving parallel prefix computation, we need not use all processors all the time. Thus, we shall propose a fault-tolerant mechanism when we solve the prefix computation problem on faulty hypercube computers. By our mechanism, prefix computation can be done in $O(\log n)$ time on a faulty hypercube computer if there exists a fault-free subcube with size being at least a half of the hypercube, and in $O(\log^2 n)$ time in the worst case.

1. Introduction

In a hypercube multiprocessor computer, each processor has a local memory. This model has some desirable characteristics with respect to error confinement. Most of the research effort on hypercube multiprocessors has focused on the fault-tolerant situation [2, 3, 4, 11, 12]. Since the increasing use of hypercube for critical applications has made their fault tolerance an important issue, fault-tolerant architectures and fault-tolerant algorithms are therefore emerging as an important area of research. We will address the fault tolerance on prefix computation in a hypercube architecture.

The **prefix computation** problem [5, 6, 8, 9] is defined as follows. Given a sequence of numbers a_0, a_1, \dots, a_{n-1} and an associative operator \otimes , we are asked to find the products $a_0 \otimes a_1 \otimes \dots \otimes a_i, 0 \leq i \leq n-1$. Note that we have to calculate n results. When parallel algorithms are desired, many applications of prefix computation have been found in a wide variety of problems [1, 10].

In this paper, we shall issue an off-line mechanism for detecting permanent faulty processors in a hypercube. Mao [10] proposed prefix computation algorithm on a nonfaulty hypercube, which requires $O(\log n)$ time, where n is the number of processors in the hypercube. In Mao's prefix computation algorithm

, he views the processors of a hypercube similar to the topology of a linear array with the head = PE(000..0) and the tail = PE(100..0) with the Gray code representation. In his algorithm, the processor (000..0) is used only once. And, the smaller identifier a processor has, the less it is used. Thus, the basic idea of our fault detection scheme is to look for a Hamiltonian path in the hypercube such that all faulty processors appear at the front of the path as closely as possible, since in Mao's algorithm, the front processors are less used. We shall propose an off-line mechanism in $O(\log n)$ time for detecting faulty processors and then use a renumbering scheme to renumber the processors which is suitable for Mao's prefix computation algorithm. It still needs $O(\log n)$ time when exactly one node is faulty. In another case, $O(\log n)$ is required when at most $d-1$ faulty nodes exist with at least a half of the hypercube being fault-free where $n = 2^d$. If the faulty nodes are spread in the whole hypercube, we can apply the divide-and-conquer strategy to solve the problem. In this case, it needs $O(\log^2 n)$ time.

The organization of this paper is as follows. In Section 2, we shall propose a mechanism for detecting one faulty node. In Section 3, we shall describe how to find a partition such that all faulty nodes are in the same subcube. In Section 4, we shall apply Mao's prefix computation algorithm with our fault-tolerant mechanism in a faulty hypercube. And finally, a concluding remark will be given in Section 5.

2. A Mechanism for Detecting One Faulty Node

In this section, we shall discuss the situation that exactly one node is faulty in a d -dimensional hypercube. It is assumed that the faulty node can be detected by all of its neighbors. By the assumption, let the neighbor of the faulty node which differs in the leftmost bit be the **initiate node**. The initiate node initiates a broadcasting procedure which causes the faulty node to be known by every other node. For simplifying description, we use $i(j)$ to denote the node differing in bit i with node j . The broadcasting procedure is as follows.

Procedure Broadcast-Faulty-Node
 $R := \{ c \}; /* c : the initiate */$

* This research work was partially supported by National Science Council, under contract NSC 81-0408-E-110-513

```

for i := 0 to d - 1 Do
  for each node j ∈ R Do
    node j sends the identifier of the faulty node to
    node i(j);
    R := R ∪ { i(j) };
  End-Do;
End-Do;

```

It is clear that in Procedure Broadcast-Faulty-Node, set R doesn't include the faulty node from the 0th iteration to the $(d-2)$ th iteration since the initiate node differs in the leftmost bit with the faulty node. Thus, in Procedure Broadcast-Faulty-Node, the initiate node broadcasts the faulty information without using the faulty node. It requires $O(d) = O(\log n)$ time for the broadcasting.

After the identifier of the faulty node is received by each processor (node), a renumbering procedure commences. Each processor performs a bitwise exclusive OR operation on its original identifier with the original identifier of the faulty node. This will cause the faulty node to have a new identifier (000...0) which is least used in our prefix computation algorithm. To illustrate our idea, let us consider a 3-dimensional hypercube. Assume that node 110 is the faulty node. In the first phase, node 010 will be the initiate node and it initiates the broadcasting procedure. And, in the renumbering procedure, each processor will perform a bitwise exclusive OR operation.

Next, we shall illustrate how the data are routed to the desired processors. Without loss of generality, in the original identifier of the faulty node, let the k th bit, $0 \leq k \leq d-1$, be the most significant bit which is equal to 1. That is, the original identifier of faulty node is $(00\dots01_{k-1}f_k\dots0_0)$ where f_j may be 0 or 1, $0 \leq j \leq k-1$. In the first step, each processor $(X\dots X_{1_k}X\dots X)$ routes its data through the following path:

$(X\dots X_{1_k}X\dots X) \rightarrow (X\dots X_{0_k}X\dots X) \rightarrow (X\dots X_{0_k}X_{k-1}\oplus f_{k-1}\dots X) \rightarrow \dots \rightarrow (X\dots X_{0_k}X_{k-1}\oplus f_{k-1}\dots X_j\oplus f_j\dots X) \rightarrow \dots \rightarrow (X\dots X_{0_k}X_{k-1}\dots X_j\oplus f_j\dots X_0\oplus f_0)$.

In the second step, each processor $(X\dots X_{0_k}X\dots X)$ routes its data through the following path:

$(X\dots X_{0_k}X\dots X) \rightarrow (X\dots X_{0_k}X\dots X_0\oplus f_0) \rightarrow (X\dots X_{0_k}X\dots X_0\oplus f_0) \rightarrow \dots \rightarrow (X\dots X_{1_k}X\dots X_j\oplus f_j\dots X_0\oplus f_0)$.

We stipulate that the node with the new identifier (000...01) performs the computation which is done originally by the faulty node. In other words, at the first step of prefix computation, the processor with the new identifier (000...01) does a favor for the faulty processor. At the same time, all other processors are idle. In the subsequent steps, every processor except the faulty one does its work normally in prefix computation. It is clear that it needs one more time step than that in a hypercube with no faulty node. Thus, the time required for our prefix computation is still $O(\log n)$.

3. A More General Detection Mechanism

In this section, we shall discuss the case that there are at most $d-1$ faulty nodes in a d -dimensional hypercube. We shall propose an mechanism to find a

subcube such that all faulty nodes are in the found subcube and the size of it is minimized.

Let $B = (b_{d-1}\dots b_1 b_0)$ be a d -bit vector where $b_j = 0$ or 1, $d-1 \leq j \leq 0$. B is said to be a **partition** of a d -dimensional hypercube since B can partition the hypercube into several subcubes (equivalence classes) by the following way. Let $X = (x_{d-1}\dots x_1 x_0)$ be a node of the d -dimensional hypercube. The equivalence class of X is defined by $\{(i_{d-1}\dots i_0) \mid i_j = x_j \text{ if } b_j = 0, d-1 \leq j \leq 0\}$. For example, assume that $d = 5$ and $B = (01011)$. B will partition the hypercube into four subcubes, each containing eight nodes. One of the subcubes consists of nodes (00100), (00101), (00110), (00111), (01100), (01101), (01110) and (01111). Note that if in B , the number of bits equal to 1 is y , then each subcube under the partition B consists of 2^y nodes.

In the following, we shall give a method to find a partition such that all faulty nodes are in the same equivalence class and the size of the equivalence class is minimized.

For $PE(i_{d-1}\dots i_1 \dots i_0)$, let $FC[PE(i_{d-1}\dots i_1 \dots i_0)]$ denote the current partition known to $PE(i_{d-1}\dots i_1 \dots i_0)$ and $FN[PE(i_{d-1}\dots i_1 \dots i_0)]$ denote the faulty node with the smallest identifier known to $PE(i_{d-1}\dots i_1 \dots i_0)$. Initially, $FC[PE(i_{d-1}\dots i_1 \dots i_0)]$ is set to (#) and $FN[PE(i_{d-1}\dots i_1 \dots i_0)]$ is set to (#) except the faulty nodes. It is assumed that a faulty node can be detected by its nonfaulty neighbors. Thus, initially, FN and FC of a faulty node can be viewed as being set to the processor identifier of itself and (000...0) respectively.

In the following, we shall use \vee and \oplus to denote a bitwise OR operation and a bitwise exclusive OR operation respectively. In addition to the conventional meanings of \vee and \oplus , we have to handle the case when one of the operands is (#). We define that $A \vee B = A$ if $B = (#)$ and $A \oplus B = A$ if $B = (#)$. Our algorithm for condensing the identifiers of the faulty processors in each processor is as follows.

Procedure Condense

```

/* Assume that if a neighbor PE(i_{d-1}\dots i_1 \dots i_0) of
PE(i_{d-1}\dots i_1 \dots i_0) is faulty, PE(i_{d-1}\dots i_1 \dots i_0) can get
(000...0) and (i_{d-1}\dots i_1 \dots i_0) as FC and FN of the
neighbor. */
If PE(i_{d-1}\dots i_1 \dots i_0) is not a faulty node then begin
  Step 1 : FC[PE(i_{d-1}\dots i_1 \dots i_0)] <- FC[PE(i_{d-1}\dots i_1 \dots i_0)] \vee FC[PE(i_{d-1}\dots i_1 \dots i_0)] \vee ( FN[PE(i_{d-1}\dots i_1 \dots i_0)] \oplus FN[PE(i_{d-1}\dots i_1 \dots i_0)] );
  Step 2 : FN[PE(i_{d-1}\dots i_1 \dots i_0)] <- min {FN[PE(i_{d-1}\dots i_1 \dots i_0)], FN[PE(i_{d-1}\dots i_1 \dots i_0)] };
end;

```

Now, an example is given to illustrate the above procedure. Consider a 5-dimensional hypercube. Assume that $PE(01110)$, $PE(01101)$ and $PE(00101)$ are faulty nodes. If Procedure Condense is repeatedly performed, the final partition known to each processor is that $FC = (01011)$ and $FN = (00101)$. Having such FC and FN , we shall show that under partition FC , all the faulty nodes fall into the equivalence class of FN .

[Theorem 1] If the message of each node can reach any other node and Procedure Condense is repeatedly

used, we can find the partition in which all faulty nodes are in the same equivalence class and the size of the equivalence class is minimized.

We omit the proof here. Refer to Mao's thesis for detailed proof [10].

In the following, based upon Procedure Condense, we shall propose an algorithm to find the partition that all faulty nodes are in the same equivalence class.

Algorithm Fault-Detection

```

Step 1: For  $j = 0$  to  $d-1$  Do
    PE( $i_{d-1} \dots i_1 \dots i_0$ ) detects its neighbor
    PE( $i_{d-1} \dots i_1 \dots i_0$ ).
    .
    If PE( $i_{d-1} \dots i_1 \dots i_0$ ) is a faulty node then
        PE( $i_{d-1} \dots i_1 \dots i_0$ ) performs Procedure
        Condense.
    End-Do.
Step 2: For  $j = 0$  to  $d-1$  Do
    PE( $i_{d-1} \dots i_1 \dots i_0$ ) exchanges its data with
    PE( $i_{d-1} \dots i_1 \dots i_0$ ).
    Each processor performs Procedure
    Condense.
    End-Do.
Step 3: Perform step 2 once more.

```

Now, we shall show that in the above algorithm, there exists a routing path between any pair of nonfaulty nodes if at most $d-1$ nodes are faulty.

[Theorem 2] In a hypercube of $n = 2^d$ nodes, if at most $d-1$ nodes are faulty, the message of any faulty node can reach all other nodes after Algorithm Fault-Detection is executed.

Refer to Mao's thesis for the proof [10]. In Theorem 2, we proved that if at most $d-1$ nodes are faulty in a d -dimensional hypercube, there exists a routing path between any pair of nonfaulty nodes in Algorithm Fault-Detection. In Theorem 1, we proved that if a processor receives two pieces of faulty information from its two neighbors, it can condense them into one piece. Thus, the amount of data transmitted between processors is always constant. The time complexity of Algorithm Fault-Detection is $O(\log n)$ where $n = 2^d$. Combining Theorems 1 and 2, we know that Algorithm Fault-Detection can find the smallest subcube that all faulty nodes are in it.

4. Fault-Tolerant Prefix Computation

In this section, we shall illustrate how to perform prefix computation in a faulty hypercube. By the previous section, we assume that all nonfaulty nodes know the smallest identifier of the faulty nodes and the smallest subcube that all faulty nodes are in it. Since $FC[PE(i_{d-1} \dots i_0)]$'s are the same in all processors, for simplifying the notation, we will use FC instead of it in the following. Similarly, FN will be used instead of $FN[PE(i_{d-1} \dots i_0)]$. Two cases have to be considered. In the first case, at least one bit in FC is 0. In other words, the size of the smallest faulty subcube is not greater than a half of the original hypercube. In the second case, $FC = (111 \dots 1)$. In other words, the

faulty nodes are spread in the whole hypercube.

Case 1 : At least one bit in FC is 0. Each processor holds three data elements. They are the partition FC , the minimum faulty node FN and the processor identifier of itself. In this case, each processor performs the following procedure to renumber its identifier.

Procedure Renumbering

```

/* In the following, X may be 0 or 1. */
Step 1: Let  $j$  be the rightmost bit position which is 0
    in  $FC$ . Let  $k$  be the leftmost bit position which
    is 1 in  $FC$ . Exchange bit  $j$  with bit  $k$  in  $FC$ ,
     $FN$  and the identifier of each processor.
Step 2: Each  $PE(X..X_{1_k}X..X_{0_k}X..X)$  routes its data
    via  $PE(X..X_{0_k}X..X_{1_j}X..X)$  to  $PE(X..X_{0_k}X..X_{1_j}X..X)$ . Each  $PE(X..X_{0_k}X..X_{1_k}X..X)$  routes
    its data via  $PE(X..X_{1_k}X..X_{1_j}X..X)$  to
     $PE(X..X_{1_k}X..X_{0_j}X..X)$ .
Step 3: Repeat steps 1 and 2 until  $j \geq k$ .
Step 4: Each processor performs the  $\oplus$  operation on
    the new identifier of  $FN$  and the new identifier
    of itself to get its final identifier.

```

We shall illustrate this procedure by the example used in the previous section. Assume that $PE(01110)$, $PE(01101)$ and $PE(00101)$ are faulty nodes in a 5-dimensional hypercube. After Procedure Fault-Detection is executed, $FC = (01011)$ and $FN = (00101)$. After step 3 of Procedure Renumbering is executed, the partition and the minimum faulty node become that $FC = (00111)$ and $FN = (01001)$ respectively, and the identifier of $(i_4 i_3 i_2 i_1 i_0)$ becomes $(i_4 i_2 i_3 i_1 i_0)$. Then, the final identifier will be obtained in step 4 by performing \oplus on new FN and $(i_4 i_2 i_3 i_1 i_0)$. The time complexity of Procedure Renumbering is obviously $O(\log n)$.

We have given new identifiers of all processors in the hypercube in which the identifiers of the faulty subcube are smaller than those in other subcube with the Gray code representation. It is clear that we have at least one nonfaulty subcube in which the identifiers of the nodes are consecutive in the Gray code representation. For clarity of presentation, we can view an equivalence class (subcube) as a solid node logically. Thus, the prefix computation can be performed as in section 2. Its time complexity is still $O(\log n)$.

Case 2 : $FC = (111 \dots 1)$. Under this situation, there must be a faulty node with identifier $(1XX \dots X)$ and another faulty node with identifier $(0XX \dots X)$ where X may be 0 or 1. These two faulty nodes cause the leftmost bit of FC to be 1. Thus, we can split the full cube into the lower subcube consisting of $PE(0X \dots X)$'s and the higher subcube consisting of $PE(1X \dots X)$'s. By this split, each subcube will contain at least one faulty node. In other words, there are at most $d-2$ faulty nodes in either the lower subcube or the higher subcube whose dimension is $d-1$. Now, each subcube has the same property as the d -dimensional hypercube. Thus, the divide-and-conquer strategy can be applied in this case. We write down the above procedure as follows.

Procedure Split

Step 1: Using the leftmost bit of identifiers, divide the

full cube into subcubes with equal sizes : the lower subcube(LC) and the higher subcube (HC).

Step 2 : Recursively compute the prefix computation in LC and HC. Note that each subcube stops the recursive call while it reach the situation of case 1.

Step 3 : The node which holds the final result of LC broadcasts its result to all nodes in HC by applying Algorithm Fault-Detection with a simple Procedure Condense, i.e., the function of Procedure Condense is reduced to only the reception and the delivery of message.

We shall analyze the time required for the above procedure. Let $T(n)$ denote the time required for the procedure in an n -node cube. After two smaller subcubes of $\frac{n}{2}$ nodes finish their computations, we merge these two results by applying Algorithm Fault-Detection. This merging procedure needs $O(\log n)$ time. Thus, we have the following equation.

$$T(n) = T\left(\frac{n}{2}\right) + O(\log n).$$

It implies that $T(n) = O(\log^2 n)$. We can conclude that our prefix computation in a faulty hypercube can be finished in $O(\log^2 n)$ time in the worst case.

5. Concluding Remarks

A hypercube multiprocessor system can be regarded as a graph. By the definition of fault tolerance, a hypercube multiprocessor system is fault-tolerant under its topology. For solving parallel prefix computation, we need not use all processors all the time. Thus, we presented some fault-tolerant mechanisms for our prefix computation algorithm. We first discussed this mechanism under a special case in which exactly one node is faulty. The time complexity under this case is $O(\log n)$. When more than one node is faulty, we view the equivalence class which contains all faulty nodes as a bigger logical processor in a hypercube with reduced dimension. Thus, it has the same time complexity as the special case. If the faulty nodes are spread in the whole hypercube, we apply the divide-and-conquer strategy to solve the problem. Thus, the complexity of our fault-tolerant prefix computation is $O(\log^2 n)$ in the worst case.

Hypercubes that miss some of their nodes are called incomplete hypercubes [7]. Unlike hypercubes, the number of processors in an incomplete hypercube may not be a power of 2, i.e., 2^d , where d is a positive integer. Thus, it is possible to transform a faulty hypercube into a maximal incomplete hypercube and perform those algorithms whose processor utilization is the same as that of our prefix computation algorithm.

K. Roy, V. Balasubramanian and J. A. Abraham, "Algorithm-Based Fault Tolerance on a Hypercube Multiprocessor," *IEEE Transactions on Computers*, Vol. 39, No. 9, Sep. 1990, pp. 1132-1145.

3. M. Chean and J. A. B. Fortes, "A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays," *IEEE Computer*, Jan. 1990, pp. 55-69.
4. M. S. Chen and K. G. Shin, "Adaptive Fault-Tolerant Routing in Hypercube Multicomputers," *IEEE Transactions on Computers*, Vol. 39, No. 12, Dec. 1990, pp. 1406-1416.
5. R. Cole and U. Vishkin, "Faster Optimal Parallel Prefix Sums and List Ranking," *Information and Control*, Vol. 81, 1989, pp. 334-352.
6. F. E. Fich, "New Bounds for Parallel Prefix Circuits," *Proc. 15th Annual ACM Symposium on Theory of Computing*, May 1983, pp. 100-109.
7. H. P. Katseff, "Incomplete Hypercubes," *IEEE Transactions on Computers*, Vol. 37, No. 5, May 1988, pp. 604-608.
8. H. J. Kim and J. G. Lee, "Partial Sum Problem Mapping into a Hypercube," *Information Processing Letters*, Vol. 36, No. 5, Dec. 1990, pp. 221-224.
9. C. P. Kruskal, L. Rudolph and M. Snir, "The Power of Parallel Prefix," *IEEE Transactions on Computers*, Vol. C-34, No. 10, Oct. 1985, pp. 965-968.
10. J. W. Mao, "Fault-Tolerant Prefix Computation on the Hypercube," Master Thesis, Institute of Applied Mathematics, National Sun Yat-sen University, Kaohsiung, Taiwan, June 1992.
11. D. K. Pradhan, "Fault-tolerant multiprocessor Link and bus network architectures," *IEEE Transactions on Computers*, Jan. 1985, pp. 33-45.
12. P. Ramanathan and K. G. Shin, "Reliable Broadcast in Hypercube Multicomputers," *IEEE Transactions on Computers*, Vol. 37, No. 12, Dec. 1988, pp. 1654-1657.

References

1. S. G. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1989.
2. P. Banerjee, J. T. Rahmeh, C. Stunkel, V. S. Nair,