Workshop on Computer Systems

Three Methods for Subcube determination in Faulty Hypercubes

Yao-ming Yeh and Yao-ming Chang

● Yao-ming Yeh and Yao-ming Chang are with the Department of Information and Computer Education, National Taiwan Normal University, Taipei, Taiwan, R.O.C.

All correspondence should be addressed to Professor Yao-ming Yeh, Department of Information and Computer Education, National Taiwan Normal University, 162 Hoping E. Rd., Sec. 1, Taipei, Taiwan, R.O.C.

Three Methods for Subcube determination in Faulty Hypercubes

Yao-ming Yeh and Yao-ming Chang
Department of Information and Computer Engineering
National Taiwan Normal University

Abstract

For a large hypercube system, the probability of fault occurrence can be high. It is often desired to reconfigure the faulty hypercube that operates in a gracefully degraded manner as to retain as many nonfaulty nodes and links as possible. Therefore the subcube determination problem is essential that the time for executing a parallel algorithm trends to depend on the dimension of the assigned subcube.

Here, we present three different methods to determine prime subcubes which are hypercube function method, Q-map method , and synchronized message passing (SMP) method. The basic ideas of these three methods are from the similarities between Boolean algebra and hypercube topology. A faulty hypercube system can be described by a hypercube function. The hypercube function is similar to the logic function of a switching circuit. The nonfaulty subcubes in the system can be obtained from the complement of set of faulty nodes by DeMorgan's law easily. We propose the Q-map method which improved from the K-map method of logic circuit to provide a easy method for finding subcubes in a small faulty hypercube system. The proposed SMP method is a parallel algorithm, which exhibits polynomial time complexity with respect to the system's dimension. This method can deal with node failures and link failures.

Keywords: Hypercube System, Parallel Processing, Fault-Tolerance

1 Introduction

The hypercube has been studied extensively as an interconnection network topology for multi-computer systems [1][2]. Due to their regular structure and low diameter, hypercube multi-computers are well suited for parallel processing. These advantages have led to numerous experimental and commercial machines including the recent development of a system with more 6,000 nodes by NCUBE. Most parallel programs developed for the hypercube can be executed on various system sizes, but they experience certain slowdowns on a small sized system. The extent of execution slowdown trends to grow as the system size decreases.

For a large hypercube system, the probability of fault occurrence can be high, make it necessary to consider the fault-tolerant issue in system design. Fault-tolerant techniques suggested for the hypercube system fall into two categories, depending on whether or not redundant nodes/links are employed. If redundancy is added, the design goal is to keep the system size unchanged in the presence of operational failures after reconfiguration by replacing the failed components with spares [3]. On the other hand, if no redundancy is involved in a hypercube, fault-tolerance is achieved either by utilizing the workable portion of the system to emulate the whole machine with certain slowdown, or by reconfiguring the machine into a smaller sized system after faults occur.

Many prior reconfiguration techniques attempt to identify complete fault-free subcubes with the maximum possible dimension in a faulty hypercube. Using these fault-free subcubes, the faulty hypercube system can obtain minimum performance degradation. Tzeng and Lin [4] proposed an efficient centralized algorithm to determine maximum fault-free subcubes. It has to run on a single processor with its time complexity being $O(m^2 N)$, where $N(=2^n)$ is the system size, but it can't determinate all possible complete subcubes. Another centralized algorithm for all complete subcubes recognition was presented by Burch and Ercal [6], it has time complexity of $O(n3^n)$ and space complexity of $O(3^n)$, where n is the dimensions of hypercube. Distributed algorithm is based on parallel multiprocessor system. Chen and Tzeng [5] proposed an distributed procedure for locating subcubes in a faulty hypercube, the complexity at each candidate node is $O(n^2m^2)$, where m is the numbers of faulty nodes and n is the dimensions of hypercube. Burch and Ercal [6] also discuss the parallelization of its centralized algorithm, the run time of the parallel algorithm is $O(n2^n)$ and utilizes $O(2^n)$ processors.

In this paper, we present three different kinds of methods for determinate complete subcubes in a hypercube that are *hypercube function* [7], *Q-map*, and *synchronized message passing* (*SMP*). Hypercube function can be regarded as a centralized algorithm and based on the complement of faulty nodes and nonfaulty nodes. According the information of faulty nodes, we can easily obtain the complete subcube of nonfaulty nodes by DeMorgan's law. Other useful rules for simplifying the hypercube function are proposed by Chen and Tzeng [5] ,and Yeh and Chang [7]. *Q-map* a visual method for simplified hypercube function which modified from K-map in Boolean algebra. Since the topology of an $i \times j$ K-map is isomorphic to an n-dimensional hypercube where $n = log_2(i \times j)$. In a 4×4 Q-map and a 4-dimensional hypercube, cells in Q-map can represent nodes in hypercube, and walls around a cell can be regarded as links of a node. Along the same rules of K-map, we

can find the complete subcubes in a small faulty hypercube by sight easily. The *SMP* method is a parallel algorithm for all complete subcubes determination. Starting with the nodes knowing their own ID, and then pass this information to all it's neighbors. Whereon, if each node receives the information from a healthy neighbor, then it compares with its own information and combines itself with its neighbor into a larger subcube. This procedure is performed in synchronous for every node in the hypercube. It stops when no more larger subcubes can be generated.

The rest of this paper is organized as follows. Section 2 summarizes the notations and definitions that will be used throughout the paper. Section 3 introduces the method which use hypercube functions and useful rules. Section 4 proposes the concept of cube map to help us finding prime subcubes. Section 5 presents a parallel algorithm for determine all complete subcubes in a faulty hypercube. The comparison of three methods and conclusion is followed in Section 6.

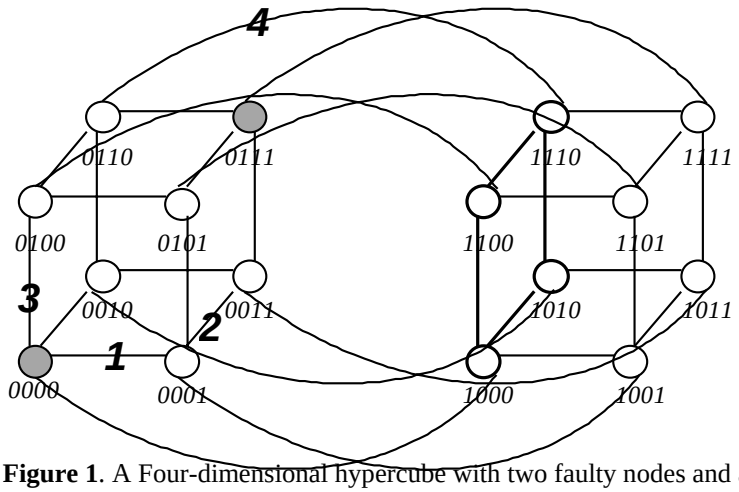

Figure 1. A Four-dimensional hypercube with two faulty nodes and a two-dimensional subcube *1**0* (bold line).

2 Preliminaries

An n-dimensional hypercube, Q_n, consists of 2^n nodes and $n2^{n-1}$ links. Each node has an unique address ($b_n b_{n-1} \cdots b_2 b_1$), $b_i \in \{0, 1\}$ for i = 1, 2,…, n. The ith bit is referred to as ith dimension. Two nodes are connected by a link if and only if their addresses differ by exactly one bit, and they are called adjacent to each other. A four-dimensional hypercube is depicted in Figure 1. Each subcube in Q_n can also be uniquely represented as address by a string of n symbols over the set {0, 1, *}, where

* is a don't care symbol. Specifically, a *k*-dimensional subcube S_k has exactly *k* *s in its address, as it involves a collection of 2^k cube nodes. As an example, nodes *1000*, *1010*, *1100*, and *1110* in Q_n constitute a two-dimensional subcube addressed by *1**0 (Figure 1). For each hypercube node, the communication link in dimension *i* is called the *i*th link of this node. For simplicity, each link is represented by a binary string with a "-" symbol in the corresponding dimension. For example, the link between nodes 0000 and 0010 is represented by 00-0. With cube address representation, one can easily see that the total number of different subcubes in all dimensions (not necessarily disjoint) is equal to 3^n.

Let a subcube be represented as a set of Boolean variables, obtained from the address of the subcube by replacing bit position *i* with b_i (or $\overline{b_i}$), if position *i* is *1* (or *0*), called a *dimension variable*. Then dropping all *s. For example, a 2-dimensional subcube *0**1* is represented by $\overline{b_4}b_1$, called a *cubeterm* which is composed by two dimension variables. It is defined subsequently.

DEFINITION 1. *A* cubeterm *is composed by m dimension variables, where* $1 \le m \le n$, *and the* cubeterm *is a (n - m)-dimensional subcube in* Q_n, *denoted as* $x_1x_2\cdots x_m$ *where* x_i *can be any dimension variable among* $b_nb_{n-1}\cdots b_1$ *and* $\overline{b_n}\,\overline{b_{n-1}}\cdots\overline{b_1}$.

We can represent a subcube with respect a given node as its dimension number only which is called *containment vector*. For example, a 2-dimensional subcube 0**1 with respect to a given node 0101 can be represented as 0101 with (2,3). Using this notation, one can reduce the space for storing all information of subcube by every node in Q_n.

DEFINITION 2. *A* containment vector *of a m-dimensional subcube* S_k *in* Q_n *can be represented by its dimension number and a node* $b_nb_{n-1}\cdots b_1$ *which is involved in this subcube as* $b_nb_{n-1}\cdots b_1$ *with* (D_i), *where* D_i *is a set of dimension position of * in* S_k, *and* $1 \le i \le n$.

Subcube determination problem is defined as given a collection of faulty nodes and faulty links on an n-dimensional hypercube, find out all possible maximum complete subcubes that consists entirely of nonfaulty nodes and links.

DEFINITION 3. *A* prime subcube *with respect to a given node, say P, is a fault-free subcube with involves P but is not contained entirely in any other fault-free subcube involving P.*

Prime subcubes with respect to different nodes could be of different sizes. No proper subcube of a prime subcube can be a candidate largest subcube. In other words, the prime subcubes with respect to all nonfaulty nodes are the all possible maximum complete subcubes in faulty hypercube and are what we expect.

3 Hypercube Functions

Hypercube algebra [7] is a new notation of hypercube. It can elegantly describe the sets of nodes or subcubes in a hypercube, and even the incomplete hypercubes [4] [8] of a faulty hypercube. The concept of this method is that the set of faulty nodes and the set of nonfaulty nodes are complement in hypercube system. Through the known from the set of faulty nodes, we can easily obtain the set of nonfaulty nodes by some rules in hypercube algebra. Fortunately, the hypercube function of nonfaulty nodes we obtain is all the prime subcubes in the faulty hypercube that we want. In the next three paragraphs, we will introduce the functional notation, rules for simplifying the functions, and the main method.

3.1 Hypercube and Truth Table

A Hypercube function is an expression formed with binary variables, the two binary operators + and ·, complement, parentheses, and an equal sign. For a given value of variables, the function can be either 0 or 1. Consider, for example, the Hypercube function of Q_n

$$F_1(b_1,b_2,b_3,b_4) = \overline{b_4}\,\overline{b_3}\,\overline{b_2}b_1 + \overline{b_4}\,\overline{b_3}b_2\overline{b_1} + \overline{b_4}\,\overline{b_3}b_2b_1 + \overline{b_4}b_3\overline{b_2}\,\overline{b_1} + \overline{b_4}b_3\overline{b_2}b_1 + \overline{b_4}b_3b_2\overline{b_1} + b_4\overline{b_3}\,\overline{b_2}\,\overline{b_1} + b_4\overline{b_3}\,\overline{b_2}b_1 + b_4\overline{b_3}b_2\overline{b_1} + b_4\overline{b_3}b_2b_1 + b_4b_3\overline{b_2}\,\overline{b_1} + b_4b_3\overline{b_2}b_1 + b_4b_3b_2\overline{b_1} + b_4b_3b_2b_1$$

The above is an example of Hypercube function represented as an algebra expression. A Hypercube function may also be represented in a truth table (Table 1) and a Hypercube graph (Figure 1). To represent a function in a truth table, we need a list of the 2^n combinations of 1's and 0's of the n binary variables, and a column showing the combinations for which the function is equal to 1 or 0. (Note that the statement $\overline{b_0}$ = 1 is equivalent to saying that b_0 = 0.)

In the other hand, let us consider a 4-dimensional hypercube with some faulty nodes. We can take the item which has $F_1(b_1,b_2,b_3,b_4)$ is 0 to be represented as a faulty node. Otherwise it is a nonfaulty node. So the above hypercube function notates that there are three 2-dimensional subcubes and two 1-dimensional subcubes

in this 4-dimentional faulty hypercube.

3.2 Simplifying Rules

Since the similarities between Boolean algebra and hypercube system, many theorems and rules of Boolean algebra can be used here. Besides basic rules which are proposed by Yeh and Chang [7] and an advanced rule which is proposed by Chen and Tzeng [5] are sometimes appear in pairs, but we only list one of them that we will use in the method. These rules are introduced as follows.

b_4	b_3	b_2	b_1	$F_1(b_1, b_2, b_3, b_4)$	b_4	b_3	b_2	b_1	$F_1(b_1, b_2, b_3, b_4)$
0	0	0	0	0	1	0	0	0	1
0	0	0	1	1	1	0	0	1	1
0	0	1	0	1	1	0	1	0	1
0	0	1	1	1	1	0	1	1	1
0	1	0	0	1	1	1	0	0	1
0	1	0	1	1	1	1	0	1	1
0	1	1	0	1	1	1	1	0	1
0	1	1	1	0	1	1	1	1	1

Table 1. Truth table for $F_1(b_1, b_2, b_3, b_4)$

Rule 1 [DeMorgan's law] $\overline{(b_i + b_j)} = \overline{b_i} \cdot \overline{b_j}$.

Rule 2 [Distributive law] $b_i \cdot (b_j + b_k) = b_i \cdot b_j + b_i \cdot b_k$.

Rule 3 [Absorption] $b_i \cdot S_i = b_i$ where S_i is a sum term which contains b_i.

Rule 4 [Advanced absorption] $(b_i + b_j) \cdot S_i S_j S_k = b_i S_j S_k + b_j S_i S_k = b_i S_j S_k + b_j S_i' S_k$ where b_i is in the sun term S_i but b_j is not, b_j is in the sum term S_j but b_i is not, neither b_i and b_j is in the sum term S_k, and S_i' is sum term S_i excluding variable b_i.

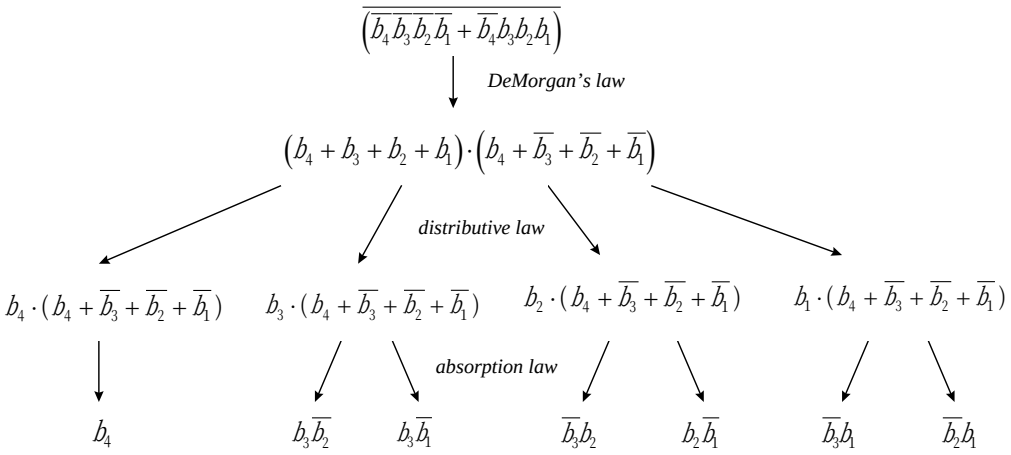

Figure 2. Converting function of nonfaulty nodes F_p by complementing function of faulty nodes F_f.

3.3 Prime Subcube Determination with Hypercube Function

In a faulty hypercube, the set of faulty nodes and the set of nonfaulty nodes are not only exclusive but also complementary. Using the hypercube function, we can obtain a function in sum-of-product(SOP) form which presents the set of faulty nodes from it's address easily. Since we know the complement of F_f (which is the set of faulty nodes) is F_p (which is the set of nonfaulty nodes), so we can just complement the function F_f and apply the DeMorgan's law to get the function F_f. The procedure is as following.

1. Listing the function F_f in SOP form according the addresses of faulty nodes.
2. Complement F_f and apply the rule 1 to transfer it in POS form.
3. Using rule 2 to expand the expression.
4. Rule 3 and 4 are employed to reduce the redundant terms in expanding process.
5. We can obtain the expanded SOP form F_p which is the set of prime subcubes.

Consider the example in 3.1, the faulty hypercube has two faulty nodes, whose addresses are 0000 and 0111 in 4-dimensional hypercube, so the function in SOP form is $F_2 = \overline{b_4}\overline{b_3}\overline{b_2}\overline{b_1} + \overline{b_4}b_3b_2b_1$, then we complement it as $\overline{F_2} = \overline{(\overline{b_4}\overline{b_3}\overline{b_2}\overline{b_1} + \overline{b_4}b_3b_2b_1)}$. Using the DeMorgan's law to transform the function into POS form. The transformed function has two sum-terms S_1 and S_2 where $S_1 = (b_4 + b_3 + b_2 + b_1)$ and $S_2 = (b_4 + \overline{b_3} + \overline{b_2} + \overline{b_1})$. Then, we expand the expression by distributed rule in four parts such as b_4S_2, b_3S_2, b_2S_2, and b_1S_2. We can employ the two absorption rules to reduce the product. Since b_4 is in S_2, so $b_4S_2 = b_4$ by rule 3, and $b_3S_2 = b_3S_2' = b_3\overline{b_2} + b_3\overline{b_1}$ where S_2' is S_2 excluding b_4 by rule 4.

Similarly, $b_2 S_2 = b_2 S_2' = \overline{b_3} b_2 + b_2 \overline{b_1}$, and $b_1 S_2 = b_1 S_2' = \overline{b_3} b_1 + \overline{b_2} b_1$. Finally, we obtain seven product- terms $b_4 + b_3\overline{b_2} + b_3\overline{b_1} + \overline{b_3}b_2 + b_2\overline{b_1} + \overline{b_3}b_1 + \overline{b_2}b_1$ which are the seven prime subcubes 1***, *10*, *1*0, *01*, **10, *0*1, and **01 (Figure 2).

$$\begin{aligned}\overline{F_2} &= \overline{\left(\overline{b_4}\,\overline{b_3}\,\overline{b_2}\,\overline{b_1} + \overline{b_4} b_3 b_2 b_1\right)} \\ &= \left(b_4 + b_3 + b_2 + b_1\right)\cdot\left(b_4 + \overline{b_3} + \overline{b_2} + \overline{b_1}\right) \\ &= b_4 + b_3\overline{b_2} + b_3\overline{b_1} + \overline{b_3}b_2 + b_2\overline{b_1} + \overline{b_3}b_1 + \overline{b_2}b_1\end{aligned}$$

4 Q-Map

Q-map is a presentation of hypercube and is abbreviated from Cube-map. It is a visual method for simplifying hypercube function, which modified from K-map in digital logic design. Whereas the connectivity of node addresses in an n-dimensional hypercube is alike to a n-bit Gray code, and the principle of Q-map is also relevant to it. The main idea behind a Q-map is that it attempts to position the *0*s and *1*s so that logically adjacent cells are also physically adjacent. This makes it easy to recognize terms that can be combined into a single, simpler term. The topology of an $i \times j$ Q-map is isomorphic to an *n*-dimensional hypercube where $n = log_2(i \times j)$. A cell in a map is corresponding to a node in a hypercube system. Moreover, a wall(ie., line) between two cells of a map is corresponding to a link in hypercube system. Each cell in a Q-map is like anode in hypercube, which has an unique address. The address is composed according to its row number and column number. For example, consider a 4×4 Q-map, address of the cell on left-up corner is 0000 which is combined by 00 (row number) and 00 (column number), and the address of the cell on right-down corner is 1010 which is combined by 10 (row number) and 10 (column number). The wall that between cells 0000 and 0001 can represent the 1st dimensional link 000-, and the wall between cells 0101 and 1101 is the 4th dimensional link -101. Figure 3a is a Q-map for a 4-dimensional hypercube with 2 faulty nodes 0000 and 0111.

4.1 Drawing Map and Encircling

Like the usage in digital design, the procedure for finding prime subcubes with Q-map is as following.

1. According the size of hypercube, form a corresponding Q-map. For example, an n-dimensional hypercube is corresponding to a $i \times j$ Q-map where $n = log_2(i \times j)$ that *i* is row, *j* is column and $j - i \le 1$.
2. Fill 0 in the cells whose corresponding nodes are failure, and fill 1 in the other cells. Then, mark an X on the walls whose corresponding links are failure.

3. Identify all prime subcubes by encircling appropriate maximum-size groups of 2^k cells which is 1, where $1 \le k \le n$. But, these groups can not contain the wall with X-mark.
4. Select a minimum set of groups that contain or cover all the cells which is 1.

For example, consider the figure 1, a 4-dimensional hypercube with two faulty nodes 0000 and 0111. We can draw the corresponding Q-map as the following figure 3a :

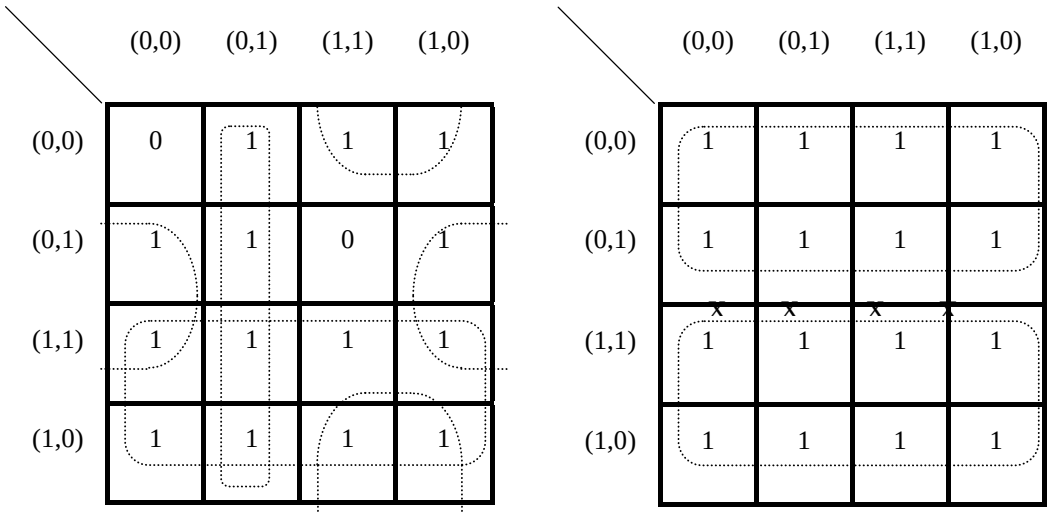

Figure 3. Two Q-maps : (a) A Q_4 with two faulty nodes 0000 and 0111. (b) A Q_4 with four faulty links -100, -101, -111, and -110.

According to the above left Q-maps, we can encircle a minimum set of four groups which cover all the cells that is 1 such as 1***, **01, *1*0, and *01*. As we mentioned in 3.1, a hypercube function can represent a faulty hypercube. So, we can further form a hypercube function, $b_4 + b_3\overline{b_1} + \overline{b_2}b_1 + \overline{b_1}b_2$, which represents one 3-dimensional subcube and three 2-dimensional subcubes.

4.2 Faulty Links in Q-map

Besides node failures, link failures may also occur in a hypercube system. Q-map is extended to handle a hypercube in which both nodes and links could fail. Links in a hypercube are represented as the walls among cells in the Q-map. Consider a Q-map of a four dimensional hypercube, each cell has 4 walls which are represented as the four links connecting with one node. Like the situation of faulty nodes, the circle by encircling process can not contain any faulty components, that are nodes and links. For example, In figure 3b, there are four faulty links that -100, -101, -111, and

-110. We easily encircle two circles that cover all available nodes but without faulty links. So, the original Q_4 is partitioned into two subcubes that 0*** and 1*** by these faulty links.

5. Synchronized Message Passing (SMP) Method

Another approach to find the prime subcube is the tabular method. This method is similar to the method proposed by Quine and McCluskey [9] which is widely used in digital logic design. This method can deal with large number of Boolean variables. Here, we present a parallel algorithm which stems from the idea of the tabular method to find the prime subcube in any size of faulty hypercube. The basic idea of this method is to form a larger subcube from two small subcubes dimension-by-dimension repeatedly by synchronized message passing. Although it is more complex than two previous methods but the advantage of this method is that it can process any size of faulty hypercube.

5.1 Tabular Method and Subcube Joining

The tabular method uses a series of subcube table, which are 0-cube table, 1-cube table, 2-cube table and so on. Each cube table contains three columns: “group”, “address”, and “mark”. Where “address” is the node address of each subcube, and “group” is to identify the subcubes which have the same weight (i.e., the address of the subcube contains the same number of 1’s). The column “mark” is to identify the subcubes that can be combined into larger subcube. The procedure of the tabular method is fairly simple. We can start from 0-cube table, which contains all available 0-dimensional subcubes (i.e., these subcubes are also non-faulty individual nodes). The procedure tries to join them into larger subcubes by checking each entry of the adjacent groups. Then 1-cube table is created to store the resulting 1-cubes. This process continues iteratively until no larger subcubes can be created.

The main idea of the procedure is to repetitively apply join operation on the entries of i-cube table until no join operation can be applied. .The join operation on i-cube table is defined by two steps: Step 1 is to find and mark two available subcubes that belong to adjacent group respectively and whose addresses are one bit differ on *d*th dimension. Then Step 2 is to create a new entry on (i+1)-cube table whose address is copied from the address of the subcube being joined with a “*” placed on the *d*th dimension of the subcube with the same representation, and attach the next available table. For example, for a hypercube with 5 dimension, from

1-cube table, two 1-cubes 0010* and 1010* from G_1 and G_2 can be joined in to a 2-cube *010* in the 2-cube table.

The algorithm is described as following.

1. Form a table $T_i = T_0$ (i.e., 0-cube table) from all available nodes. Each entry is arranged by the weight of node address. All entries in T_i are divided into groups G_0, G_1, …, G_m, where G_h contain all nodes with exactly h 1's in their address, $0 \le h \le m$.
2. Scan T_i and compare each entry E' of G_h with each entry E'' of G_{h+1}, for all h where $0 \le h \le m$. If the addresses in E' and E'' are different on only one dimension, then mark off both entries. Then a new entry E (whose address is copied from the address of E' and E'' with a "*" replaced on that different dimension)is formed. Check if the new entry E is already in T_{i+1} before it is inserted into G_h of table T_{i+1}.
3. If T_{i+1} is not empty, change i to $i+1$ and repeat Step 2. If T_{i+1} is empty, the unmarked entries remained in all tables are the prime subcubes of the faulty hypercube.

For example, let us discuss the hypercube function F_1 mentioned in Section 3.1 (see Table 1). From the above algorithm, we first form a 0-cube table T_0 from all nodes except two faulty nodes 0000 and 0111. The entries in T_0 are divided into groups according to the number of 1's in their address. In Table 2a, there are 4 groups G_1, G_2, G_3, and G_4. Then we scan T_0 and compare each node address in G_1 and G_2 . And we find that 0001 and 0011 are different on only dimension 2, so we mark off these two entries. A new entry, which has address 00*1, is created from them. Since 00*1 is new to 1-cube table T_1, therefore this entry is added into group G_1 in table T_1, Similarly, 0001 and 0101 can be joined to 0*01 , 0010 and 0011 can be joined to 001*. When comparing G_1 and G_2, go on to compare G_2 and G_3, G_3 and G_4 in T_0. From the above algorithm, after no join operation can be applied to all cube table, we can obtain seven unmarked subcubes, which are *0*1, **01, *01*, **10, *10*, *1*0, and 1*** (see Table 2c, 2d). Since these subcubes can not be combined into larger subcubes, in other words, they are the prime subcubes.

Table 2a T_0

0-cube table

Group	address	mark
G_1	0001	ˇ
	0010	ˇ
	0100	ˇ
	1000	ˇ
G_2	0011	ˇ
	0101	ˇ
	0110	ˇ
	1001	ˇ
	1010	ˇ
	1100	ˇ
G_3	0111	ˇ
	1011	ˇ
	1101	ˇ
G_4	1111	ˇ

Table 2b T_1

1-cube table

Group	address	mark
G_1	00*1	ˇ
	0*01	ˇ
	*001	ˇ
	001*	ˇ
	0*10	ˇ
	*010	ˇ
	010*	ˇ
	01*0	ˇ
	*100	ˇ
	100*	ˇ
	10*0	ˇ
	1*00	ˇ
G_2	*011	ˇ
	*101	ˇ
	*110	ˇ
	10*1	ˇ
	1*01	ˇ
	101*	ˇ
	110*	ˇ
	11*0	ˇ
	1*10	ˇ
G_3	1*11	ˇ
	11*1	ˇ
	111*	ˇ

Table 2c T_2

2-cube table

Group	address	mark
G_1	*0*1	
	**01	
	01	
	**10	
	10	
	*1*0	
	10**	ˇ
	1*0*	ˇ
	1**0	ˇ
G_2	1*1*	ˇ
	1**1	ˇ
	11**	ˇ

Table 2d T_3

3-cube table

Group	address	mark
G_1	1***	

Unmarked subcubes are Prime Subcubes.

Tabular method is better than previous two methods that it can handle larger dimensional hypercube and can be implemented in program very easily. However, Tabular method has some drawbacks comparing with other methods. Tabular method needs space complexity of O(3^n) to record all possible subcubes of Q_n, and there are some overheads of redundant comparisons so it runs in O($n3^n$) time. If we simply transfer it to a parallel version algorithm with an n-dimensional hypercube, every nodes still needs to maintain O(3^n) space and runs in O($n2^n$) time [6]. In next section, we will present a parallel algorithm with n-dimensional hypercube and it only

needs O(2^n) space maintained by every nonfaulty nodes and runs in O(n^2) time.

5.2 Methodology of SMP

We first assume that the fault model is a fail-stop processor. When a node is faulty, it will not perform any incorrect actions and simply ceases functioning. Synchronized message passing method is a parallel algorithm to determine all subcubes in hypercube with faulty nodes and/or links. The idea of SMP is to pass a message to their neighbors through the same dimensional link synchronously by all nonfaulty nodes. It is shown as figure 4. By receiving those messages from dimension 1 to n, each nonfaulty node will know how its neighborhood is. Nodes then gradually gather the global view by exchanging local information to each other iteratively. Before we introduce the algorithm, the data structure of passing message and some essential ideas will be presented in next two paragraphs.

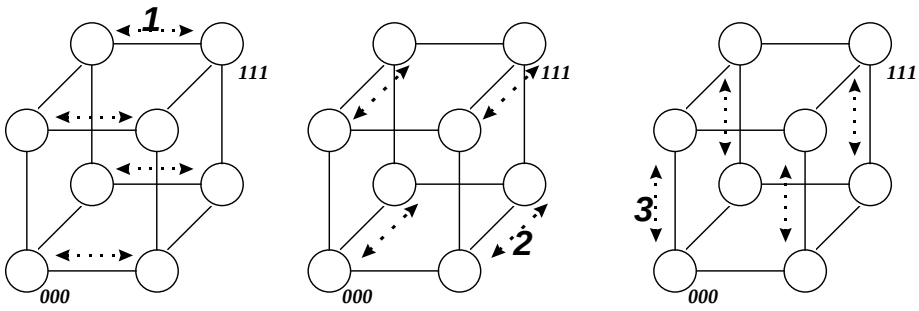

Figure 4. Nodes synchronously exchanged message through the same dimensional link.

Subcubes of a n-dimensional hypercube can be represented as a n-tuple of 0, 1, *, where * signifies “don’t care.” For example, 0*1* contains 0010, 0011, 0110, and 0111. With this representation, one can clearly see that the total number of different subcube in all dimensions is equal to 3^n. However, for implementing in the SMP method, every node only needs to maintain the information that the subcubes which contain itself in the process. So, to keep all the 3^n possible subcubes is not necessary. With definition 2, a subcube can also be described to a given node with some dimension numbers which is called *containment vector*. Using this dimensional representation, only 2^n possible combinations are needed. Therefore, each node needs space of O(2^n) to record the all possible combinations of available dimensions. This array will be regarded as information for exchanging and merging to neighbors in the algorithm.

```
Algorithm 1 : SMP
begin
    message←nil ;
    for i = 1 to d do
        for j = 1 to d do
            sent { message } through dimension j ;
            receive { message }' through dimension j ;
            compare { message } with { message }' ;
            select the same vectors from two messages ;
            for all same vectors,
                if ( vector + j ) > the vector then
                    mark the character and add ( character + j ) to the queue ;
                end if
        end for
    message ← queue ;
    result ← characters without mark ;
    if ( queue = nil ) then exit;
    end for
end
```

Procedure of SMP method is executed by every available node synchronously. Table 5 shows how the algorithm 1 works on a 4-simensional hypercube with two failed nodes 0000 and 0111. For presenting intelligibly, the message in each step is represent in dimensional notation not the bit array we use in algorithm. At the beginning of the algorithm, each available node hold a null message ( ) which means the original individual one. There is a nested loop in the algorithm that the inner loop denote the dimensions of a hypercube and the outer loop control the largest dimension of subcube which is combined. During the inner loop, nodes send message to neighbors from small dimension to large one and wait to receive others which like figure 4. Then, compare the message that select the same containment vectors and check if they can be combined in to a larger one. If yes, mark it off and generate a new term added in to the *next* array, else leave it to the *result* array. After finishing the inner loop, next array will be regard as new message and iterate this process in outer loop until no larger subcubes are generated. When the algorithm accomplish, each node will hold a result array which records the prime subcube involving the node itself. As an example illustrated in table 5, two prime subcubes involving node 0001 are (2, 4) and (3, 4) that are *0*1 and **01, and three subcubes involving node 1010 are (1, 4), (3, 4), and (1, 2, 3) that are *01*, **10, and 1***.

During the SMP algorithm, the comparison of two messages needs much time, so it takes O(2^n) time to scan the containment vectors and selects the same ones.

Then, the assembling process has to check that if two vectors could be merged to create a new one.

However, in the SMP algorithm, the message exchanging is frequent and the message sometimes contains more than one containment vectors. Moreover, it will take a lot of time during the subcube merging process which need to compare two set of containment vectors and decide if they can be merged to a new containment vector. Hence, for saving the communication cost and quickly processing in SMP algorithm, we propose a *subcube vector* which is a only 2^n bits array to maintain all the containment vectors which are encoded to a number, called *location number*. The encoding process of containment vectors is quite simple that each vector was transferred to this number which is regarded as a bit position as following.

$$\text{Location number} = \begin{cases} 0 & \text{if containment vector is empty, ie. ( ).} \\ \sum 2^{D_j-1} & \text{otherwise.} \end{cases}$$

DEFINITION 4. *A* subcube vector *is a* 2^n *bits array* $S_m[i]$*, where i is from 0 to* 2^n*-1, with respect to a given node m that records its all* 2^n *possible containment vectors in* Q_n*. The bit value in subcube vector is 1 if this bit position is equal to the location number, otherwise, the value is 0.*

For example, consider a node 011100 which is involved in three subcubes that 01*1*0, 0**10*, and 01***0 in a Q_6, i.e. the containment vectors are (2,4), (1,2,5), and (2,3,4) respectively. We can transfer them to three location numbers such that $2^1 + 2^3 = 10$, $2^0 + 2^1 + 2^4 = 14$, and $2^1 + 2^2 + 2^3 = 19$ by encoding process. Finally, we set 10^{th}, 19^{th}, 14^{th}, bit to 1 in the subcube vector which is a 2^6 bits array (Figure 5).

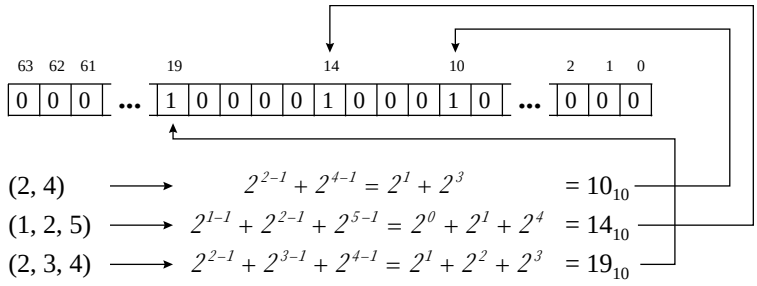

Figure 5. Subcube vector of node 011100 with (2,4), (1,2,5), and (2,3,4) in Q_6.

Furthermore, we propose an auxiliary vector called *dimension vector* to accelerate the merging process and the definition is as following,

DEFINITION 5. *A* dimension vector *is a* 2^n *bits array* $D_i[j]$*, where j is from 0 to* 2^n*-1, with respect to a given dimension i in* Q_n*.*

$$D_i[j] = 1 \quad \text{if } j \bmod 2^i < 2^{i-1}$$
$$= 0 \quad \text{otherwise}$$

The dimension vector is fixed of each dimension for accelerating the merging process. For example, the following table shows the four dimension vectors of Q_4:

Dimension	Dimension vector	Hexadecimal number
D_1	[0101010101010101]	5555
D_2	[0011001100110011]	3333
D_3	[0000111100001111]	0F0F
D_4	[0000000011111111]	00FF

Table 3. Dimension vectors of Q_4.

Using these improved data structure such that subcube vector and dimension vector, one can lower not only the space and communication complexity but also the time significantly. The origin comparison operation needs O(2^n) time to scan and select, but it only need a O(*1*) time to perform a bitwise operation. Moreover, the merging operation also be reduced to two bitwise operation which run in O(*1*) time. The modified SMP algorithm with subcube vector is listed as following.

Algorithm 2: SMP with subcube vector

```
/* n is dimension of the overall hypercube.
   S denotes subcube vector which is a 2^n boolean array, initialized to 0.
   D_j denoted dimension vector of dimension j.
   TEMP, NEXT are temporary array,
   RESULT is the output array, all initialized to 0. */
begin
    if ( I am non-faulty ) then S[0] ← 1
    else exit;
    end if
    for i = 1 to n do
        RESULT ← RESULT | S
        for j = 1 to n do
            sent S through dimension j ;
            receive S_j' through dimension j ;
            TEMP ← S & S_j' & D_j ;
            RESULT ← RESULT ⊕ TEMP ;
            NEXT ← NEXT | ( TEMP << 2^j ) ;
        end for
        S ← NEXT ;
        if ( NEXT = 0 ) then exit;
        NEXT ← 0
    end for
end
```

Note: &, |, ⊕, and << means the operation AND, OR, XOR, and LEFT SHIFT respectively.

The SMP method can handle a hypercube in which both nodes and links failure. Since the basic idea of our method is to pass message through links, if there is a faulty link connecting to a available node, we can just cancel the operation of message passing through the faulty dimension. Although its neighbor of that dimension will regard it a faulty node, other neighbor nodes of other dimension will still continue the process of subcube assembling. This method can still work for link failure without modification.

5.3 Complexity Analysis

Basically, we present a parallel algorithm and run on a *n*-dimensional hypercube system, it contains two loops which are nested ,a messages comparison operation, and an assembling process in the nested loop. Since using certain bitwise operations such as AND, OR, XOR, and SHIFT to achieve the comparison and merging mechanisms, it needs only O(*1*) time when the adequate bandwidth for the 2^n bit arrays are given that two arrays can be operated in constant time. Since the inner loop is form 1 to *n* and the outer loop is from 1 to *n* in the worst case (no faulty nodes and links), n^2 time iteration is needed for this nested loop. Through the analysis above, we can obtain a run time complexity as below:

$$O(1) \times n^2 = O(n^2)$$

Although the SMP algorithm run in $O(n^2)$ time, the output of each node is still the subcube vector which is needed to transfer into containment vector. Therefore additional decoding operation will take $O(2^n)$ time that the overall run time complexity will be

$$O(n^2) + O(2^n) = O(2^n) = O(P)$$

where $P = 2^n$, is the number of processor of Q_n.

Another important consideration is the communication cost of the algorithm. We use the subcube vector which is a 2^n bits array as message of every available node for exchanging each other. Thus, $O(2^n)$ bit communication cost is needed in this algorithm.

A similar algorithm for subcube recognition was proposed by Burch and Ercal [6], it needs $O(3^n)$ space complexity and takes $O(n\,3^n)$ time complexity by the linear version, and the further parallel version ran in $O(n\,2^n)$ time complexity on an n-dimensional hypercube.

6 Comparison and Conclusion

In this paper, we present three methods for prime subcube determination which

are Hypercube Function, Q-map, and SMP Method. The original concepts of these three methods are all from the hypercube algebra. We summarize the difference of features and uses in the following listing.

1. Fault Tolerance

Lets first analyze the fault tolerance about them. Hypercube function we present in the paper can only deal with the failed nodes in a hypercube. Chen and Tzeng [5] has propose a similar algorithm which is a distributed version can be extended to tolerate the failed links. But, the time complexity is depended on the number of faulty component and the degree of a hypercube. Since Q-map can complete represent nodes and links of a hypercube, subcubes can be determined correctly by the encircling process. SMP is a parallel algorithm which is modified from the tabular method also can tolerate the failed nodes and links by it's originally idea.

2. Visuality and programmability

We secondly consider the visuality and programmability about these methods. Since Q-map is designed a method that easily be operated by vision and handed work, the visuality is certainly well. However, the encircling operation graphical method is disadvantageous to represent in data structure and programming. The hypercube function is operated like other mathematical expressions, simplifying rules can be easily applied in the program. As we know, one of the characteristics of tabular and SMP method is that they can be implemented in program very easily. Therefore, the hypercube function and SMP have the better programmability.

3. Scalability

Finally, Q-map method is adapted to the four and under dimensional hypercube but not recommended when the dimension of hypercube is higher than six. SMP is a parallel algorithm that can deal with higher dimensional hypercube by nodes themselves. The hypercube function can processes the large hypercube but the function will be copious when the numbers of dimension are very large.

Group		***Group 0***	***Group 1***				***Group 2***						***Group 3***				***Group 4***
node id		0000	0001	0010	0100	1000	0011	0101	0110	1001	1010	1100	0111	1011	1101	1110	1111
initial		faulty	( ) ˅	( ) ˅	( ) ˅	( ) ˅	( ) ˅	( ) ˅	( ) ˅	( ) ˅	( ) ˅	( ) ˅	faulty	( ) ˅	( ) ˅	( ) ˅	( ) ˅
i = 1	*j = 1*			(1) ˅	(1) ˅	(1) ˅	(1) ˅	(1) ˅		(1) ˅	(1) ˅	(1) ˅		(1) ˅	(1) ˅	(1) ˅	(1) ˅
	j = 2		(2) ˅		(2) ˅	(2) ˅	(2) ˅		(2) ˅	(2) ˅	(2) ˅	(2) ˅		(2) ˅	(2) ˅	(2) ˅	(2) ˅
	j = 3		(3) ˅	(3) ˅		(3) ˅		(3) ˅	(3) ˅	(3) ˅	(3) ˅	(3) ˅		(3) ˅	(3) ˅	(3) ˅	(3) ˅
	j = 4		(4) ˅	(4) ˅	(4) ˅		(4) ˅	(4) ˅	(4) ˅	(4) ˅	(4) ˅	(4) ˅		(4) ˅	(4) ˅	(4) ˅	
i = 2	*j = 1*			(1,4)	(1,4)	(1,2) ˅ (1,3) ˅	(1,4)	(1,4)		(1,2) ˅ (1,3) ˅	(1,2) ˅ (1,3) ˅ (1,4)	(1,2) ˅ (1,3) ˅ (1,4)		(1,2) ˅ (1,3) ˅ (1,4)	(1,2) ˅ (1,3) ˅ (1,4)	(1,2) ˅ (1,3) ˅	(1,2) ˅ (1,3) ˅
	j = 2		(2,4)		(2,4)	(2,3) ˅	(2,4)		(2,4)	(2,3) ˅ (2,4)	(2,3) ˅	(2,3) ˅ (2,4)		(2,3) ˅ (2,4)	(2,3) ˅	(2,3) (2,4) ˅	(2,3) ˅
	j = 3		(3,4)	(3,4)				(3,4)	(3,4)	(3,4)	(3,4)				(3,4)	(3,4)	
	j = 4																
i = 3	*j = 1*					(1,2,3)				(1,2,3)	(1,2,3)	(1,2,3)		(1,2,3)	(1,2,3)	(1,2,3)	(1,2,3)
	j = 2																
	j = 3																
	j = 4																

Seven prime subcubes are determined such that *0*1, **01, *01*, **10, *10*, *1*0, and 1***.

Table 5. Illustration of SMP algorithm on a faulty hypercube with two faulty nodes 0000 and 0111. Duplicated terms were omitted.

References

【1.】J. Squire and S. M. Palais, “Programming and design considerations of a highly parallel computer,” in *Proc. AFIP Spring Joint Comput. Conf.*, vol. 23, pp. 395-400, 1963.

【2.】Y. Saad and M. H. Schultz, “Topological Properties of Hypercubes,“ *IEEE Trans. Computers*, vol. 37, no. 7, pp. 867-872, July 1988.

【3.】J. Bruck, R. Cypher, and C.-T. Ho, “Fault-Tolerant Meshes and Hypercubes with Minimal Numbers of Spares,” *IEEE Trans. Computers*, vol.41, no.5, pp.1,089-1,104, Sep 1993.

【4.】N.-F. Tzeng and G. Lin, “Efficient Determination of Maximal Incomplete Subcubes in Hypercubes with Faults,” *IEEE Trans. Computers*, vol.45, no.11, pp.1303-1308, Nov 1996.

【5.】H.-L. Chen, N.-F. Tseng. “Subcube Determination in Faulty Hypercubes.” *IEEE Trans. Computers*, vol.46, no.8, pp.871-879, Aug 1997.

【6.】H. J. Burch and F. Ercal. “A Fast Algorithm For Complete Subcube Recognition.” *Proc. of IEEE 1997 Int’l Symp. Parallel Architectures, Algorithms and Networks*, pp.85-90, 1997.

【7.】Y. M. Yeh and Y. M. Chang. “Hypercube Algebra”

【8.】Katseff, “Incomplete Hypercube,” *IEEE Trans. Computers*, vol.37, no.5, pp.604-608, May 1988.

【9.】John P. Hayes. “Introduction to Digital Logic Design.” *Addison-Wesley, Reading, MA*, pp.279-330, 1993.