Tolerating Faults in Injured Hypercubes Using Maximal Fault-Free Subcube-Ring

Jang-Ping Sheu and Yuh-Shyan Chen

Department of Computer Science and Information Engineering
National Central University, Chung-Li 32054, TAIWAN
sheujp@mbox.ee.ncu.edu.tw
April 18, 1995

Abstract

In this paper, we present a reconfiguration approach to identify the maximal fault-free subcube-ring for tolerating faults in injured hypercubes. The fault-free subcube-ring is connected by a ring of fault-free subcubes with dilation 3. By exploiting the size of fault-free subcubes as large as possible, the maximal fault-free subcube-ring with higher processor utilization is obtained. Using this approach, we can tolerate more than n faults in n-dimensional hypercubes. To demonstrate the fault-tolerant capability of our approach, we implement two fault-tolerant algorithms, matrix-multiplication and sorting algorithms, on the nCUBE/2E hypercube machine with 32 processors. The simulation results show that our reconfiguration approach has low performance slowdown and high processor utilization.

Keywords: Fault tolerance, injured hypercubes, matrix-multiplication, sorting.

1. Introduction

The n-dimensional hypercube (n-cube), which interconnects exactly $N = 2^n$ processors, is one of the most popular interconnection topologies for parallel computers. As the size of the hypercube system increases, fault tolerance has become an important issue for such a large system to continue operations after failure of one or more processors/links. In this paper, we study how algorithms that are originally designed for fault-free hypercubes can be implemented on hypercubes that contain any number of faults with reasonable slowdown. To measure the efficiency of processor utilization, several researchers use *slowdown* ratio [10] [25], which is the execution time in the injured n-cube divided by its time requirement in the fault-free hypercube. The lower the *slowdown* ratio is, the higher the processor utilization of system obtains.

Most of the recently proposed fault-tolerant strategies address the issue of *reconfiguration* once the faulty processors are identified. These reconfiguration strategies have been developed [10] [12] [13] [17] [19] [20] [24] without adding any redundancy to the desired architecture. They attempt to mask the effects of faults by using the healthy part of the hypercube architecture. The desired goal of these strategies is to obtain the same functionality with a reasonable slowdown. One approach of the reconfiguration strategies is to identify the *largest fault-free subcube* and use the subcube to emulate the entire hypercube [12] [17] [19]. However, this approach results in a tremendous underutilization of resource and high degree of performance slowdown. A different but related approach, which is to identify the *maximal incomplete subcube*, is proposed by Chen and Tzeng [13] by using a *reject-region* concept. The maximal incomplete subcube involves one maximal complete subcube plus certain smaller complete subcubes, and may accommodate multiple jobs with different size. Their approach cannot apply a complete algorithm on such an identified maximal incomplete subcube.

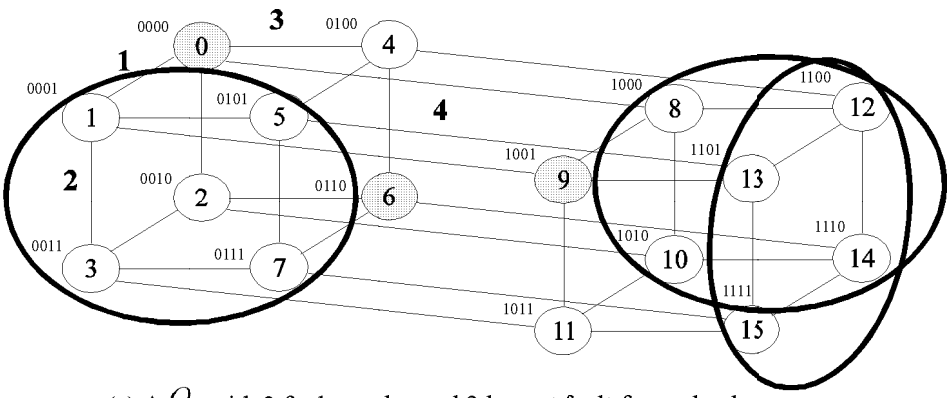
An effective approach, namely the *free dimension*, is presented by Raghavendra, Yang, and Tien [20] to achieve the fault tolerance in a faulty n-cube. Using free dimension approach [25], simulation of any SIMD algorithm on faulty n-cube takes 2 slowdown ratio of computation and 4 slowdown ratio of communication when the number of faulty nodes is no more than $\lceil \frac{n}{2} \rceil$. Other application algorithms with the free-dimension approach [20], such as embedding rings and meshes [27], embedding binary tree [28], and a prefix computation [21], have been successfully developed on the faulty n-cube. In addition, Sheu, Chen, and Chang [24] proposed a subcube partitioning method for designing a fault-tolerant sorting algorithm that can tolerate at most n - 1 faulty processors on n-dimensional hypercubes. However, all of these algorithms are restricted in the number of faulty processors. Their algorithms can tolerate at most n - 1 faulty processors.

Recently, Bruck, Cypher, and Soroker [10] proposed a technique in n-cube using the *subcube-partitioning* approach. In the approach, any regular algorithm can be implemented on an n-cube that has fewer than n faults with slowdown ratios of 2 for computation and 4 for

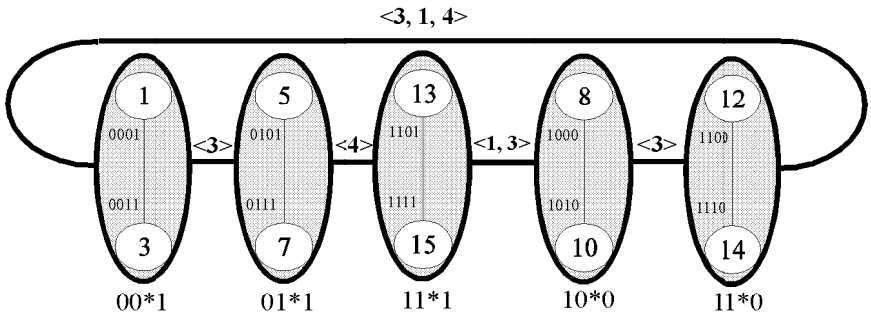
communication. Moreover, this is the first result showing that an n-cube can tolerate more than n arbitrarily placed faults with a constant factor slowdown. The approach is to partition a faulty hypercube into subcubes such that each subcube consists fewer faults. Any regular algorithm can be performed on an n-cube by using a single healthy node in each subcube to simulate all actions of the same subcube. The simulation action of subcube-partitioning approach causes high computation and communication slowdown. Developing an efficient reconfiguration strategy, which can tolerate arbitrarily number of faulty nodes and reduce the performance slowdown for any regular algorithms, is consequently the purpose of our study. Our fault model, similar to [10], is defined as follows. All faults including node/link faults are permanent. We only consider node faults and an edge fault is assumed that one of the nodes incident upon it is faulty. We also assume that faulty nodes can neither perform calculations nor route data.

In this paper, we firstly present a recognition algorithm which can recognize all possible largest fault-free subcubes in injured hypercubes. Each fault-free subcube with same size is treated as a processing unit. Our reconfiguration approach identifies the *fault-free subcube-ring* which is constructed by a ring of these processing units with dilation 3 at most. Based on these recognized fault-free subcubes, we propose an efficient algorithm to identify the maximal fault-free subcube-ring. Identifying the maximal fault-free subcube-ring is achieved by exploiting the size of each fault-free subcube and the number of fault-free subcubes as large as possible. For illustrating the fault-tolerant capability of our approach, we implement two application algorithms, fault-tolerant matrix-multiplication and sorting algorithm, on the nCUBE/2E hypercube machines with 32 processors. If the number of faults is less than n, the *slowdown* ratio for running these application algorithms on the maximal fault-free subcube-ring, compared with fault-free algorithms, is smaller than 2. This is due to the reason that the processor utilization of our approach is larger than 50%. Even when number of faults $\leq 2^{n-2}$, the average *slowdown* ratio of these two algorithms with our fault-tolerant scheme is smaller than 2.5.

The rest of this paper is organized as follows. The primary properties of maximal fault-free subcube-ring are introduced in Section 2. Systematic technique for identifying the maximal fault-free subcube-ring is addressed in Section 3. Two important application algorithms, fault-tolerant matrix-multiplication and sorting algorithms, on the maximal fault-free subcube-ring are implemented in Section 4. The performance improvement of our approach with the subcube-partitioning approach is analyzed in Section 5. The conclusions will be finally given in Section 6.



(a) A Q_4 with 3 faulty nodes and 3 largest fault-free subcubes.



(b) A feasible fault-free subcube-ring $R_S(1, 5)$.

Fig. 1. A feasible fault-free subcube-ring $R_s(1,5)$ from a 4-cube with 3 faulty nodes.

2. Notation and Preliminary

Let Q_n be the n-dimensional Boolean cube, or n-cube, which consists of 2^n nodes with each node representing a processor and each edge between two nodes in Q_n corresponding to a communication link between two processors. Every node b has address $b_n b_{n-1} \cdots b_1$ with $b_i \in \{0,1\}$, $1 \leq i \leq n$, where b_i is called the *i-th* bit (also called the *i-th* dimension or dimension i) of the address. Each m-dimensional subcube Q_m, or m-subcube, has a unique address $x_n x_{n-1} \cdots x_1$ with $x_i \in \{0,1,*\}$, $1 \leq i \leq n$, where exactly m bits take the value $*$, ($*$ is a don't care symbol). For example, $*^{n-1}0$ and $*^{n-1}1$ denote the two $(n-1)$-subcubes separated by dimension 1, where $*^i$ stands for i consecutive $*$'s.

Now we introduce the concept of *prime-subcube* [12]. Let F denote a set of faulty nodes in an injured n-cube. Given a nonfaulty node P, a *prime-subcube* with respect to a nonfaulty node P is a fault-free subcube which involves P but is not contained entirely in any other fault-free subcube involving P, where $P \in Q_n - F$. Note that there may be more than one prime-subcube corresponding to P. For example, given an injured 4-cube with set $F = \{0000, 0110, 1001\}$ as shown in Fig. 1(a), there exists one prime-subcube 0**1 with respect to node

0001 and 3 prime-subcubes *10*, 1**0, and 11** with respect to node 1100. Recently, Chen and Tzeng [12] have proposed a subcube identification algorithm to identify the fault-free subcubes in a faulty hypercube. However, their algorithm may fail to identify the maximum fault-free subcube and each healthy node must obtains all faulty nodes' addresses before running their algorithm. The work presented here is different from those carried out in [12] in the sense that all maximum fault-free subcubes can be recognized by our algorithm and each healthy node only keeps the status of its neighboring nodes. The maximum fault-free subcubes exist in set of the prime-subcubes. Alternatively, in our algorithm, we will recognize all *prime-subcubes* for each healthy node P. Our reconfiguration scheme with more processor utilization can be identified from these recognized prime-subcubes.

Assume there exists a sequence of k disjoint fault-free m-subcubes $[Q_m^1, Q_m^2, \cdots, Q_m^k]$. Sequence $[Q_m^1, Q_m^2, \cdots, Q_m^k]$ is selected from set of prime-subcubes which are collected from all prime-subcubes of each healthy node. Each m-subcube is treated as a processing unit and we reconfigure these processing units into a ring with dilation 3 at most, namely *fault-free subcube-ring* $R_s(m,k)$, which is defined as follows.

Definition 1 : Fault-free subcube-ring $R_s(m,k)$

Let $R_s(m,k)$ be a sequence pair $([Q_m^1, Q_m^2, \cdots, Q_m^k], [j_1, j_2, \ldots, j_{k-1}, j_k])$ to denote a feasible fault-free subcube-ring, where $[Q_m^1, Q_m^2, \cdots, Q_m^k]$ is a sequence of k disjoint fault-free m-subcubes and $[j_1, j_2, \ldots, j_{k-1}, j_k]$ is a sequence of dimension sequences, for each dimension sequence $j_i = <d_1^i, \cdots, d_w^i>$, $d_w^i \in \{1, 2, \cdots, n\}$, $1 \leq w \leq 3$, and $1 \leq i \leq k$. The $R_s(m,k)$ is constructed by each node in Q_m^i connects to a node in $Q_m^{(i \bmod k)+1}$ along dimensions $d_1^i, \cdots, d_w^i$. Therefore,

$$Q_m^1 \overset{j_1}{\longleftrightarrow} Q_m^2 \overset{j_2}{\longleftrightarrow} Q_m^3 \overset{j_3}{\longleftrightarrow} \cdots Q_m^{k-1} \overset{j_{k-1}}{\longleftrightarrow} Q_m^k \overset{j_k}{\longleftrightarrow} Q_m^1.$$

□

If the connection $Q_m^k \overset{j_k}{\longleftrightarrow} Q_m^1$ does not exist, then a fault-free subcube-chain, denoted by $C_s(m,k)$, is constructed. The processor utilization of $R_s(m,k)$ and $C_s(m,k)$ are $2^m \times k$. The diameter of $R_s(m,k)$ and $C_s(m,k)$ are at most $3 \times \lceil k/2 \rceil + m$ and $3 \times k + m$, respectively. Obviously, a fault-free subcube-ring $R_s(m,k)$ can be treated as a fault-free subcube-chain $C_s(m,k)$ with the same size of m-subcube. On the contrary, a fault-free subcube-ring $R_s(m-1, 2\times k)$ can be directly obtained from fault-free subcube-chain $C_s(m,k)$ with double diameter. Recall above example, we may recognize 3 largest fault-free subcubes as shown in Fig. 1(a). In Fig. 1(b), a fault-free subcube-ring $R_s(1,5) = 00{*}1 \overset{<3>}{\longleftrightarrow} 01{*}1 \overset{<4>}{\longleftrightarrow} 11{*}1 \overset{<1,3>}{\longleftrightarrow} 10{*}0 \overset{<3>}{\longleftrightarrow} 11{*}0 \overset{<3,1,4>}{\longleftrightarrow} 00{*}1$ is constructed.

Throughout this paper, we will only focus our attention on constructing a feasible fault-free subcube-ring $R_s(m,k)$ in an injured hypercube. In order to preserve low diameter and obtain better processor utilization, identifying the maximal fault-free subcube-ring $R_s(m,k)$ is the main objective of this study. The value of m and k of $R_s(m,k)$ are the two major

consideration factors. Determining the maximal fault-free subcube-ring $R_s(m,k)$ is controlled by what values of m and k being are the best selection. The maximal fault-free subcube-ring $R_s(m,k)$ is defined here that the maximum value of m is determined firstly and then the maximum value of k is exploited.

3. Identifying the maximal fault-free subcube-ring $R_s(m,k)$

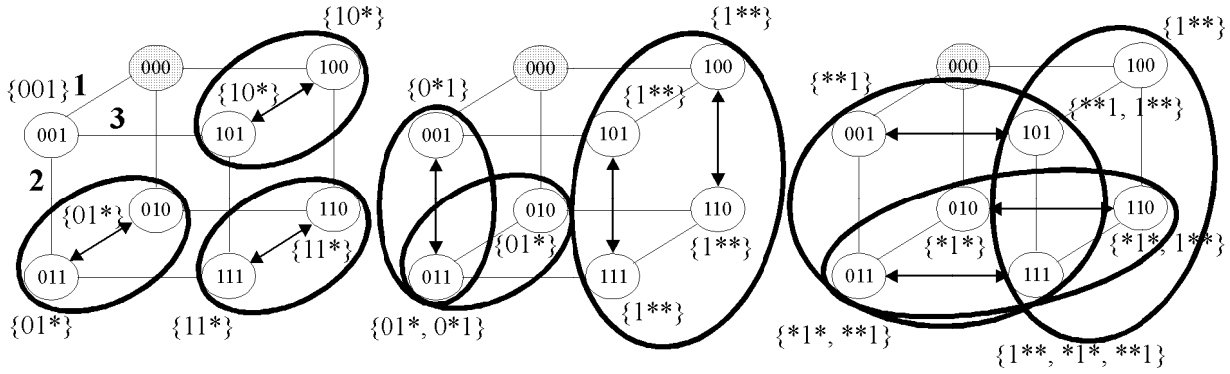
In this section, an efficient algorithm for identifying the maximal fault-free subcube-ring $R_s(m,k)$ in an injured n-cube is proposed. The identification algorithm can be categorized into two parts. First, a distributed algorithm is derived in Section 3.1 to recognize all possible *prime-subcubes* with respect to each nonfaulty node P, where $P \in Q_n - F$. Second, a distributed algorithm is presented in Section 3.2 to efficiently identify the maximal fault-free subcube-ring $R_s(m,k)$ based on the recognized *prime-subcubes*.

3.1 Recognizing the prime-subcubes

In this subsection, we describe how to recognize all *prime-subcubes* corresponding to each healthy node P, where $P \in Q_n - F$.

Without loss of generality, we focus on recognizing a set of prime-subcube with respect to a given node P, where $P \in Q_n - F$. Assume that the i-dimensional neighboring node of P is P_i', where $1 \leq i \leq n$. Before running our recognition algorithm, nonfaulty node P keeps a variable $\gamma = (\gamma_n, \gamma_{n-1} \ldots, \gamma_1)$ to record the status of its i-dimensional neighboring node P_i'; if P_i' is nonfaulty then set bit $\gamma_i = 1$; else set bit $\gamma_i = 0$, for all $1 \leq i \leq n$. To demonstrate our algorithm, we first introduce some terms. Assume that the address of node P is $x_n x_{n-1} \cdots x_{i+1} x_i \cdots x_1$ and address of node P_i' is $x_n x_{n-1} \cdots x_{i+1} \overline{x}_i \cdots x_1$, where $x_i \in \{0,1\}$ and $1 \leq i \leq n$. Let H_i^P represent an i-subcube whose address is $x_n x_{n-1} \cdots x_{i+1} *^i$. We denote $\mho_{H_i^P}$ as a set of prime-subcube with respect to node P and subcube H_i^P such that each subcube in set $\mho_{H_i^P}$ involving P is not contained entirely in part of any other fault-free subcube of H_i^P, where $1 \leq i \leq n$. Similarly, $\mho_{H_i^{P'}}$ is the set of prime-subcube with respect to node P_i' and subcube $H_i^{P'}$, where $1 \leq i \leq n$. As a consequence, $\mho_{H_n^P}$ is the set of prime-subcube with respect to node P and Q_n. Each subcube x in set $\mho_{H_i^P}$, $1 \leq i \leq n$, has the property that there is no larger subcube containing x in $\mho_{H_i^P}$. As an example, consider an injured hypercube Q_3 with $F = \{000\}$ as shown in Fig. 2. For a given node P is 011 and its 2-dimensional neighboring node P_2' is 001. Subcube H_2^{011} is 0** and sets $\mho_{H_2^{011}}$ and $\mho_{H_2^{001}}$ are {01*, 0*1} and {0*1}, respectively.

Our recognition algorithm is an ASCEND algorithm to recursively concatenate smaller healthy subcubes into larger healthy subcube. Initially, sets of $\mho_{H_0^P}$ and $\mho_{H_0^{P'}}$ are the respective addresses of node P and P_i'. The set $\mho_{H_i^P}$ is derived by the $SC(\mho_{H_{i-1}^P}, \mho_{H_{i-1}^{P'}})$ (*subcube-concatenation* or SC) operation. The $SC(\mho_{H_{i-1}^P}, \mho_{H_{i-1}^{P'}})$ operation is defined here to repeatedly recognize $\mho_{H_i^P}$, where i is ranging from 1 to n. The SC operation is divided



(a) Recognizing $\mho_{H_1^P}$. (b) Recognizing $\mho_{H_2^P}$. (c) Recognizing $\mho_{H_3^P}$.

Fig. 2. Recognizing prime-subcubes on a 3-cube with faulty node 000.

into two phases as described in what follows.

In phase I, if node P_i' is fault-free, then node P sends its set $\mho_{H_{i-1}^P}$ to its i-dimensional neighboring node P_i' and receives a set $\mho_{H_{i-1}^{P'}}$ from node P_i'. After this phase, node P contains sets $\mho_{H_{i-1}^P}$ and $\mho_{H_{i-1}^{P'}}$ and then continue to perform phase II of SC operation. If node P_i' is a faulty node, then set $\mho_{H_i^P}$ to be $\mho_{H_{i-1}^P}$ and skip the phase II of SC operation, where $1 \leq i \leq n$.

In phase II, each node P performs the *bit-concatenation* (BC) operation $\alpha = \oplus(x, y)$ for each element $x = b_n b_{n-1} \cdots b_1 \in \mho_{H_{i-1}^P}$, $y = b_n' b_{n-1}' \cdots b_1' \in \mho_{H_{i-1}^{P'}}$ to recognize a new set $\mho_{H_i^P}$ with $\alpha = z_n z_{n-1} \cdots z_1$ and $1 \leq i \leq n$. The $\oplus$ is a bitwise operator on each pair of b_k and b_k', i.e., $z_k = b_k \oplus b_k'$ for $1 \leq k \leq n$. The bit operation $\oplus$ is divided into three cases and defined as follows. In case 1, it yields 0, 1, and $*$ if the bits b_k and b_k' have the same value "0", "1", "$*$". In case 2, if one of bits is "0" and another one is 1, then it yields $*$. In case 3, if one of bits is "$*$" and another one is "0" (or "1") then it restores to 0 (or 1). Let subcubes x and y are neighboring subcubes, symbol $x + y$ denotes the union of subcubes x and y and all links between x and y.

Proposition 1: If subcubes $x \in \mho_{H_{i-1}^P}$ and $y \in \mho_{H_{i-1}^{P'}}$ are two disjoint neighboring subcubes, then subcube $\oplus(x, y)$ is the prime-subcube containing node P and belong to $x + y$.

Proposition 2: For any given two subcubes a and b, where a and b are not disjoint, then subcube $\oplus(a, b)$ is the intersection subcube of subcubes a and b.

For instance, subcubes 01* and 1** are two disjoint neighboring subcubes as shown in Fig. 2(b), $\oplus$(01*, 1**) = *1* is the prime-subcube containing node 011 and belong to 01* + 1**. Subcubes 01* and 0*1 are not disjoint as shown in Fig. 2(b), the intersection subcube of 01* and 0*1 is $\oplus$(01*, 0*1) = 011.

Algorithm: Recognizing prime-subcubes (RPS)

Input: Each nonfaulty node P maintains a variable $\gamma = (\gamma_n, \gamma_{n-1} \ldots, \gamma_1)$ for keeping status of all neighboring nodes, where $P \in Q_n - F$. The initial set of set $\mho_{H_0^P}$ on node P is its node address.

Output: Each nonfaulty node P obtains its set of prime-subcube $\mho_{H_n^P}$, where $P \in Q_n - F$.

Step 1: **for** $i := 1$ to n **do step 2** through **step 3**

Step 2: **if** $(\gamma_i = 1)$ **then** do **step 3**; **else** $\mho_{H_i^P} = \mho_{H_{i-1}^P}$;

Step 3: For each pair of neighboring nodes P and P_i' along dimension i, applying the $SC(\mho_{H_{i-1}^P}, \mho_{H_{i-1}^{P'}})$ operation:

(a) /* Phase I of SC operation */
Each node P sends its $\mho_{H_{i-1}^P}$ to neighboring node P_i' and receives a set $\mho_{H_{i-1}^{P'}}$ from node P_i'.

(b) /* Phase II of SC operation */
For each subcube x in $\mho_{H_{i-1}^P}$ and subcube y in $\mho_{H_{i-1}^{P'}}$ do:
$\alpha = \oplus(x, y)$; /* bit-concatenation (BC) operation */
if $x = \oplus(x, \alpha)$ **then** adds α into $\mho_{H_i^P}$ if α is not existed in $\mho_{H_i^P}$;
else add x and α into $\mho_{H_i^P}$ if x and α are not existed in $\mho_{H_i^P}$;

Fig. 3. Recognizing prime-subcubes (RPS) algorithm.

If a is a subcube of b then equation $a = \oplus(a, b)$ holds. For each $x \in \mho_{H_{i-1}^P}$ and $y \in \mho_{H_{i-1}^{P'}}$, subcubes x and y are two disjoint neighboring subcubes and let $\alpha = \oplus(x, y)$, then set $\mho_{H_i^P}$ is recognized as follows.

$$\mho_{H_i^P} = \begin{cases} \mho_{H_i^P} \cup \alpha & \text{if } x = \oplus(x, \alpha) \text{ and } \alpha \notin \mho_{H_i^P} \\ \mho_{H_i^P} \cup x \cup \alpha & \text{if } x \neq \oplus(x, \alpha) \text{ and } x \notin \mho_{H_i^P} \text{ and } \alpha \notin \mho_{H_i^P} \end{cases}$$

After applying the two phases of $SC(\mho_{H_{i-1}^P}, \mho_{H_{i-1}^{P'}})$ operation for n times, in final, $\mho_{H_n^P}$ is a set of prime-subcube for node P. Recall above example, for a given pair of neighboring nodes P (011) and P_3' (111), the set $\mho_{H_2^{011}}$ is {01*, 0*1} and $\mho_{H_2^{111}}$ is {1**}. To obtain $\mho_{H_3^{011}}$, the SC operation is performed on sets $\mho_{H_2^{011}} = \{01*, 0*1\}$ and $\mho_{H_2^{111}} = \{1**\}$. Equations $\oplus$(01*, 1**) = *1* and $\oplus$(0*1, 1**) = **1 hold. Thus, $\mho_{H_3^{011}} = \{*1*, **1\}$ is obtained.

Fig. 3 shows the distributed algorithm of recognizing all prime-subcubes (RPS) on each nonfaulty node P, where $P \in Q_n - F$. As an example, consider an injured hypercube Q_3 with $F = \{0\}$, each set of prime-subcubes $\mho_{H_1^P}$, $\mho_{H_2^P}$, and $\mho_{H_3^P}$ for each nonfaulty node P are respectively recognized as shown in Figs. 2(a), 2(b), and 2(c), respectively.

The following theorem states that all possible set of prime-subcube is recognized by our recognition algorithm.

Theorem 1 : The set of all prime-subcube with respect to each nonfaulty node P is recognized by applying the $SC(\mho_{H_{i-1}^P}, \mho_{H_{i-1}^{P'}})$ operation n times by varying i from 1 to n, for all $P \in Q_n - F$.

Proof: Without loss of generality, consider that a pair of i-dimensional neighboring nodes P (with set $\mho_{H_{i-1}^P}$) and P_i' (with set $\mho_{H_{i-1}^{P'}}$), where $P \in Q_n - F$. We prove that the SC operation can recognize all prime-subcubes by induction as follows.

Basis $i = 1$: Initially, sets of $\mho_{H_0^P}$ and $\mho_{H_0^{P'}}$ are addresses of nodes P and P_1', respectively. If node P_1' is fault-free, set $\mho_{H_1^P}$ is easy to obtain since that addresses of P and P_1' differ in the first bit. Otherwise, set $\mho_{H_1^P}$ is original set $\mho_{H_0^P}$.

Hypothesis $i = k - 1$: After performing SC operations for $k - 1$ times, sets $\mho_{H_{k-1}^P}$ and $\mho_{H_{k-1}^{P'}}$ are assumed to be obtained with respect to P and P_i', respectively. Therefore, all prime-subcubes on subcubes H_{k-1}^P and $H_{k-1}^{P'}$ corresponding to nodes P and P_{k-1}' are constructed, respectively.

Induction $i = k$: Now apply the $SC(\mho_{H_{k-1}^P}, \mho_{H_{k-1}^{P'}})$ operation along dimension k. For each $x \in$ set $\mho_{H_{k-1}^P}$, $y \in \mho_{H_{k-1}^{P'}}$, x and y are part of subcubes $x_n x_{n-1} \cdots x_{k+1} x_k *^{k-1}$ and $x_n x_{n-1} \cdots x_{k+1} \overline{x}_k *^{k-1}$, respectively. Since subcube x containing node P and y containing neighboring node P_k', subcubes x and y are two disjoint neighboring subcubes. By proposition 1, $\oplus(x, y)$ is the prime-subcube containing node P and belong to $x + y$. For each $x \in$ set $\mho_{H_{k-1}^P}$, all $y \in \mho_{H_{k-1}^{P'}}$ are selected to validate whether a larger prime-subcube on H_k^P exists or not. In other words, it is guaranteed that a larger prime-subcube can be recognized by the $\oplus(x, y)$ operation if it exists. Consequently, set $\mho_{H_k^P}$ can be constructed by $SC(\mho_{H_{k-1}^P}, \mho_{H_{k-1}^{P'}})$ operation. By induction, we conclude that the recognition algorithm recognizes all prime-subcubes.

□

The time complexity T_{RPS} of algorithm RPS is analyzed as follows. In step 1, we perform n loops of step 2 and step 3. The time cost of step 2 and step 3(a) is constant. If the number of $|F|$ is larger than n, the number of $\mho_{H_{i-1}^P}$ on each iteration is $1 \leq |\mho_{H_{i-1}^P}| \leq C^i_{\lceil i/2 \rceil}$, for $1 \leq i \leq n$. We need $O((C^i_{\lceil i/2 \rceil})^2)$ times to perform the BC operations in step 3. The execution time of the BC operation is $O(n)$. Another time cost $O(n)$ is needed to check weather a given element exists in set $\mho_{H_i^P}$ or not (using the conventional hashing technique). Thus, the total time cost of T_{RPS} can be measured by the following equation.

$$T_{RPS} = O(2n \cdot \textstyle\sum_{i=1}^{n} (C^i_{\lceil i/2 \rceil})^2) = O(n^2 \cdot N), \text{ where } N = 2^n.$$

Note that if $|F| \leq n$, the size of $\mho_{H_{i-1}^P}$ is $1 \leq |\mho_{H_{i-1}^P}| \leq i$. The total time cost of T_{RPS} becomes $O(2n \cdot \sum_{i=1}^{n} i^2) = O(n \frac{n(n+1)(2n+2)}{6}) = O(n^4)$.

Algorithm: Identifying maximal fault-free subcube-ring $R_s(m,k)$ (IMSR)

Input: Each healthy node P has its corresponding prime-subcubes, where $P \in Q_n - F$.

Output: Two sequences $[Q_m^1, Q_m^2, \cdots, Q_m^k, Q_m^1]$ and $[j_1, j_2, \ldots, j_{k-1}, j_k]$ of $R_s(m,k)$ are obtained for each node P.

Step 1: Each node P obtains all prime-subcubes from all other fault-free nodes.

Step 2: Each node P constructs a subcube, namely maximum *bridge-subcube*, and then use the maximum *bridge-subcube* to eliminated useless subcubes existed in set of all prime-subcubes.

Step 3: Each node P constructs a tree, called *subcube-tree* T, from the remanding subcubes of all prime-subcubes. Fully using the *subcube-tree* T, a largest fault-free subcube-ring with dilation 3 is identified.

Step 4: Host node determines the maximal fault-free subcube-ring $R_s(m,k)$ among these largest fault-free subcube-ring and broadcasts $R_s(m,k)$ to each node P.

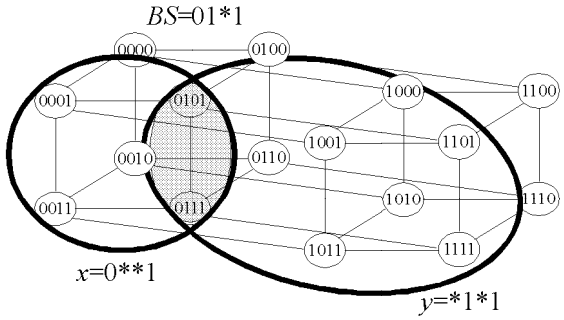
Fig. 4. Identifying maximal fault-free subcube-ring $R_s(m,k)$ algorithm.

3.2 Identifying the maximal fault-free subcube-ring $R_s(m,k)$

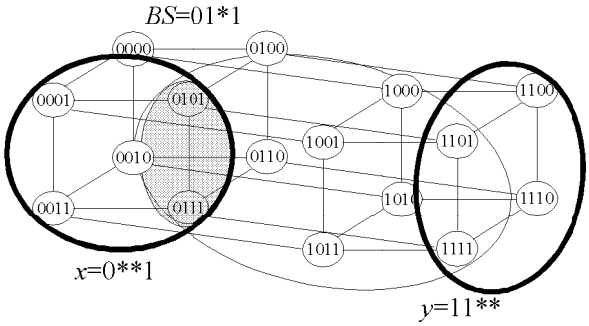
Using the algorithm RPS, each healthy node can construct its corresponding prime-subcubes. In this subsection, we show how to effectively identify the maximal fault-free subcube-ring $R_s(m,k)$ from these prime-subcubes. The proposed algorithm is a distributed one. The algorithm of identifying maximal fault-free subcube-ring $R_s(m,k)$ (IMSR) is divided into four steps and given in Fig. 4. Each step of algorithm IMSR is described in the following.

First, in step 1, all prime-subcubes are collected into set $\mho$ and each healthy node P obtains set $\mho$, where $P \in Q_n - F$. This is because that we will distributively select all subcubes of $R_s(m,k)$ from the set $\mho$. This tasks can be achieved as the following description. It is assumed that the hypercube has one host (also known as the service node of system manager), which has a direct connection to each cube node, like the nCUBE/2E [2] and the Intel *i*PSC/860 [1]. Each nonfaulty node sends its own prime-subcubes to host by the direct connection. Host node combines all prime-subcubes into a set $\mho$ and broadcasts the collected set $\mho$ to each nonfaulty node by the direct connection.

Second, in step 2, let's recall the description of maximal fault-free subcube-ring $R_s(m,k)$ $= ([Q_m^1, Q_m^2, \cdots, Q_m^k, Q_m^1], [j_1, j_2, \ldots, j_{k-1}, j_k])$. For each subcube of Q_m^1, Q_m^2, $\cdots$, and Q_m^k, we observe the fact that all '*' occur on the same positions of each subcube's address. In the following, we explain how to obtain these subcubes from set $\mho$. We firstly define the



(a) subcubes x and y are not disjoint



(b) subcubes x and y are disjoint

Fig. 5. Constructing *bridge-subcube* of $P = 0101$ and $P'_4 = 1101$.

subcube-sequence. For given a subcube $x = b_n b_{n-1} \cdots b_i \cdots b_1$, $b_i = \{0, 1, *\}$ and $1 \leq i \leq n$, let *subcube-sequence* $\mathcal{S}(x)$ be $<s_j, s_{j-1}, \ldots, s_1>$ such that $b_{s_k} = \{*\}$ and $b_h = \{0, 1\}$, where $1 \leq k \leq j$, $1 \leq h \leq n$, and $h \notin s_j, s_{j-1}, \ldots, s_1$. For instance, *subcube-sequences* $\mathcal{S}(*0*1)$ and $\mathcal{S}(*1*0)$ are $<4, 2>$, so subcubes $*0*1$ and $*1*0$ have same *subcube-sequence* $<4, 2>$. For given a subcube x, the *subcube-sequence* $\mathcal{S}(x)$ is used to eliminate useless subcubes existed in set $\mho$. The useless subcube is one existed in set $\mho$ has different *subcube-sequence* $\mathcal{S}(x)$.

Before describing how to select subcubes from set $\mho$, we must exploit a subcube and then use this subcube to eliminate useless subcubes in set $\mho$. Such subcube is called as the *bridge-subcube* throughout this work. We devoted to the basic aspects of finding the *bridge-subcube* in the following. For each nonfaulty node P, considering each pair of nodes P with $\mho_{H_n^P}$ and P'_i with $\mho_{H_n^{P'}}$, for $1 \leq i \leq n$, all pair of $x \in \mho_{H_n^P}$ and $y \in \mho_{H_n^{P'}}$ are selected to find the *bridge-subcube*. At the beginning, the *bridge-subcube* or $BS(x, y)$ is defined as follows. First, if x and y are not disjoint, then *bridge-subcube* is the intersection subcube of x and y; that is, $BS(x, y) = \oplus(x, y)$ by proposition 2. Second, if x and y are two disjoint neighboring subcubes, then $BS(x, y)$ is the intersection subcube of x and $\oplus(x, y)$; that is, $BS(x, y) = \oplus(x, \oplus(x, y))$, where $\oplus(x, y)$ is the prime-subcube containing node P and belong to $x + y$ by proposition 1. For instance, consider nodes P and P'_4 are 0101 and 1101, respectively. If $x = 0**1$ and $y = *1*1$, then $BS(0**1, *1*1)$ is $\oplus(0**1, *1*1) = 01*1$ as shown in Fig. 5(a). Fig. 5(b) shows that if $x = 0**1$ and $y = 11**$, then $BS(0**1, 11**)$ is $\oplus(0**1, \oplus(0**1, 11**)) =$

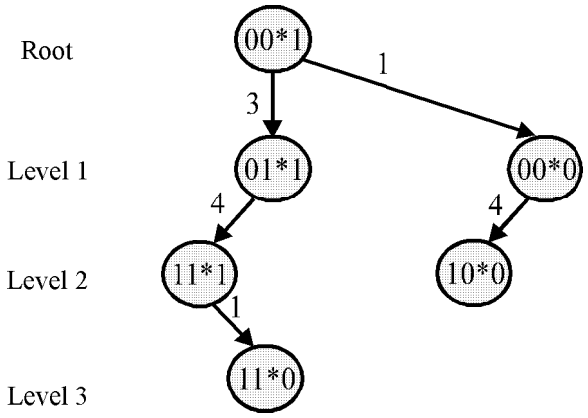


Fig. 6. Constructing a *subcube-tree* T, where MBS = 00*1, *subcube-sequence* $\mathcal{S}$(00*1) = <2>, and set Ψ = {00*0, 10*0, 00*1, 01*1, 11*1, 11*0}.

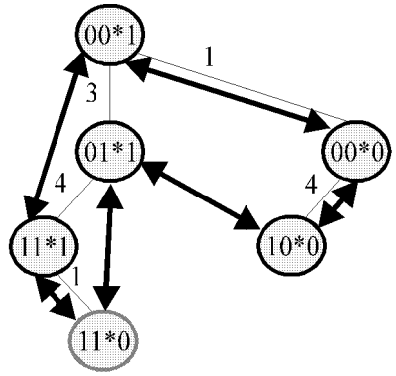
$\oplus$(0**1, *1*1) = 01*1. Note that, the criterion to determine $R_s(m,k)$ is the maximum value of m will be exploited and then to identify $R_s(m,k)$ with largest value of k. Therefore, we select the maximum *bridge-subcube* which is denoted as MBS. The selected MBS should satisfy the following max function.

$$MBS = \max_{x \in \mho_{H_n^P}, y \in \mho_{H_n^{P'}}} BS(x,y)$$

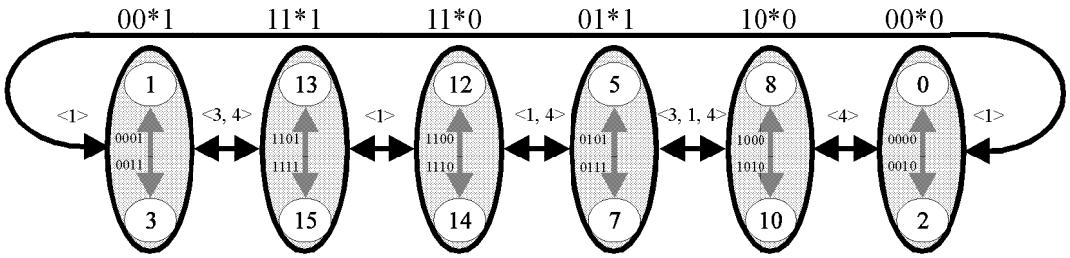
Since each subcube Q_m^i of $R_s(m,k) = ([Q_m^1, Q_m^2, \cdots, Q_m^k, Q_m^1], [j_1, j_2, \ldots, j_{k-1}, j_k])$ has the same *subcube-sequence* $\mathcal{S}(Q_m^i)$, for $1 \leq i \leq k$, therefore, we only keep subcubes from set $\mho$ with same *subcube-sequence* $\mathcal{S}(MBS)$. This work can be achieved as follows. All subcubes $x \in \mho$ with the same *subcube-sequence* $\mathcal{S}(MBS)$ are collected into set Ψ. If $|x| > m$, then subcube x is partitioned into m-subcubes with same *subcube-sequence* $\mathcal{S}(MBS)$ and collected into set Ψ.

Third, in step 3, each node P constructs a *subcube-tree* T based on set Ψ and MBS. Fully using the whole *subcube-tree* T, a feasible fault-free subcube-ring with dilation 3 is identified. The *subcube-tree* T is constructed as follows. Each node of *subcube-tree* T is a subcube of set Ψ. The total nodes of *subcube-tree* T is at most 2^{n-m} and all subcubes in set Ψ construct T. The *subcube-tree* T is a Breadth-First-Searching spanning tree. Root of tree T is the selected MBS and branches of tree T represent the possible neighboring subcubes. Each node v of *subcube-tree* T probes each of the $n-m-1$ neighboring m-subcubes. If the neighboring m-subcubes exist in set Ψ but not exist in *subcube-tree* T, node v connects to the m-subcube. Repeatedly performing above operations until no neighboring m-subcube can be further found, the *subcube-tree* T is constructed.

Continually, the whole *subcube-tree* T is used to identify the $R_s(m,k)$ for the purpose of maximizing the processor utilization. It is known that an N-node ring can be one-to-



(a) Identifying a largest fault-free subcube-ring from a *subcube-tree T*.



(b) A largest fault-free subcube-ring $R_s(1, 6)$.

Fig. 7. Identification of a largest fault-free subcube-ring $R_s(1, 6)$ from the *subcube-tree* T.

one embedded with dilation 3 in any connected N-node network [5]. A *subcube-tree* T is a connected network, as a consequence, a largest fault-free subcube-ring with dilation 3 can be identified. Consider an injured hypercube Q_4 with $F = \{6, 9\}$. All prime-subcubes are collected into set $\mho = \{$*0*0, 00**, 0**1, 0*0*, **00, *01*, **11, *1*1, *10*, 1**0, 11**, 1*1*$\}$. The MBS is 00*1 and *subcube-sequence* $\mathcal{S}$(00*1) is <2>, so set Ψ is {00*0, 10*0, 00*1, 01*1, 11*1, 11*0}. Root of *subcube-tree* T is 00*1, therefore the *subcube-tree* T is easily constructed from set Ψ as shown in Fig. 6. Fig. 7(a) displays how to identify a largest fault-free subcube-ring $R_s(1, 6)$ from the *subcube-tree* T. Consequently, a largest fault-free subcube-ring $R_s(1, 6)$ = ([00*1, 11*1, 11*0, 01*1, 10*0, 00*0, 00*1], [<3,4>, <1>, <1,4>, <3,1,4>, <4>, <1>]) is constructed as shown in Fig. 7(b).

Finally, in step 4, the maximal fault-free subcube-ring $R_s(m, k)$ is determined among these largest fault-free subcube-ring. Each nonfaulty node $P \in Q_n - F$ has constructed its largest fault-free subcube-ring. Then each node P sends its largest fault-free subcube-ring to host node by the direct links. The host node determines the maximal fault-free subcube-ring $R_s(m, k)$ from the view of the processor utilization. If there exist more than one of largest fault-free subcube-ring with same processor utilization, host node randomly selects one from them. Finally, host node broadcasts the final $R_s(m, k)$ to each nonfaulty node P by the direct links.

We now analyze the total time cost T_{IMSR} of algorithm IMSR. The maximum number of prime-subcubes with respect to a given nonfaulty node P is $C^n_{\lceil n/2 \rceil}$ [13], where $P \in Q_n - F$. From Stirling's approximation, $C^n_{\lceil n/2 \rceil} \approx \frac{2^n}{\sqrt{n}}$ for a large n [13]. In step 1, time $O(\frac{2^n}{\sqrt{n}} \times nN)$ is needed for each nonfaulty node sending its prime-subcubes to host node. In step 2, we take time $O(n^2 \times (\frac{2^n}{\sqrt{n}})^2)$ to determine the MBS. We takes time $O(\frac{2^n}{\sqrt{n}} \times nN)$ to keep elements in set $\mho$ with the same *subcube-sequence* $\mathcal{S}(MBS)$. The total number of nodes in the *subcube-tree* T does not exceed 2^n and each probing step takes $O(n)$ to probe whether its children exist or not. The time cost of step 3 is $O(nN)$. In step 4, time cost $O(N)$, where $N = 2^n$, is needed. Thus, the total time cost of T_{IMSR} can be measured by the following equation.

$$\begin{aligned} T_{IMSR} &= O(\tfrac{2^n}{\sqrt{n}} \times nN) + O(n^2 \times (\tfrac{2^n}{\sqrt{n}})^2) + O(\tfrac{2^n}{\sqrt{n}} \times nN) + O(nN) + O(N) \\ &= O(\sqrt{n} \times N^2) + O(n \times N^2) + O(\sqrt{n} \times N^2) + O(nN) + O(N) \\ &= O(n \times N^2) \end{aligned}$$

The total time cost T of T_{RPS} and T_{IMSR} is $= O(n^2 \times N) + O(n \times N^2) = O(n \times N^2)$.

4. Fault-tolerant algorithms on the $R_s(m,k)$ and $C_s(m,k)$

Many scientific algorithms already successfully developed on the n-cube. The main objective of our fault-tolerant scheme is to tailor these existed algorithms onto $R_s(m,k)$. Two application algorithms, matrix-multiplication and sorting, are respectively presented in Section 4.1 and 4.2 to illustrate the work of designing the fault-tolerant algorithms on the $R_s(m,k)$ and $C_s(m,k)$.

4.1 Fault-tolerant matrix-multiplication algorithm on $R_s(m,k)$

J. Berntsen [9] proposes a communication efficient matrix multiplication algorithm on n-cube. The n-cube includes a two-dimensional mesh interconnect with wrap around which is mapped by a binary Gray code. The matrix multiplication $C = A * B$ is performed where C, A, and B are full $N \times N$ matrices. Since there are $P = 2^n$ identical processors, the matrices are distributed on a $\sqrt{P} \times \sqrt{P}$ mesh of processors. In order to compute C, the neighbour to neighbour communication and computation for each processor are needed to perform [9]. Based on J. Berntsen's algorithm, we simulate each fault-free subcube of $R_s(m,k)$ as a processing unit. Our major concentration then is to arrange the jobs between each pair of neighboring processing units of the maximal fault-free subcube-ring $R_s(m,k)$. Our algorithm is divided into two phases. The first phase is to redistribute the matrices C, A, and B onto a $R_s(m,k)$. Consider a $R_s(m,k)$, there are k m-subcubes each containing 2^m nodes. Each m-subcube of $R_s(m,k)$ includes a ring which is mapped by a binary Gray code. A two-dimensional mesh interconnect with wrap around containing $k \times 2^m$ processors is formed. The matrices C, A, and B are distributed into a $k \times 2^m$ mesh of processors as follows. Let LCM(x, y) denote the least common multiple of x and y. First, we split the full $N \times N$ matrices into $\mu \times \mu$ submatrices, where $\mu = \text{LCM}(2^m, k)$. The matrix C is divided

in square submatrices C_{ab} holding the elements c_{ik}, with $Na/\mu \leq i < N(a+1)/\mu$, $Nb/\mu \leq k < N(b+1)/\mu$ and $0 \leq a, b < \mu$. Similarly, matrices A and B are split up in the same way. Second, we evenly partition the $\mu \times \mu$ submatrices onto $k \times 2^m$ processors of $R_s(m,k)$. As a result, each processor contains $(\mu/2^m) \times (\mu/k)$ submatrices of C, A, and B.

The second phase is to rearrange the jobs of the neighbour to neighbour communications and computations of each processor of $R_s(m,k)$. Since each processor contains $(\mu/2^m) \times (\mu/k)$ submatrices of C, A, and B, it can be viewed as a matrix with $(\mu/2^m)$ rows and (μ/k) columns. The communication and computation are modified as follows. First, each processor must multiply the respectively submatrices of A and B and sum the product to the respective part of C. Then, we let all submatrices of A shifts left one row. The shift left operation of boundary column is achieved by sending leftmost column with $(\mu/2^m)$ submatrices of A to the west and receiving a column with $(\mu/2^m)$ submatrices of A from the east. Third, we let all submatrices of A shifts up one row. The shift up operation of boundary row is achieved by sending the top row with (μ/k) submatrices of B to the north and receiving a row with (μ/k) submatrices of B from the south. After executing above communication and computation steps μ times, matrix C is thus obtained.

The derivation of time cost T_{FM} of the fault-tolerant matrix-multiplication algorithm is described as follows. Assume that τ is the time to do a floating multiplication or addition, t_{comm} is the time to communicate a single real word and t_{start} is the startup time. Our algorithm performs μ iterations of the neighbour to neighbour communication and computation operations. Time cost of computation operations is $2 \times \frac{\mu}{2^m} \times \frac{\mu}{k} \times (\frac{N}{\mu})^3\tau$. Time cost of communication operations is $t_{start} + 3 \times (\frac{\mu}{2^m} \times (\frac{N}{\mu})^2)t_{comm} + t_{start} + 3 \times (\frac{\mu}{k} \times (\frac{N}{\mu})^2)t_{comm}$. Therefore, the total time cost T_{FM} is

$$\begin{aligned} T_{FM} &= \mu \ (\ 2 \times \tfrac{\mu}{2^m} \times \tfrac{\mu}{k} \times (\tfrac{N}{\mu})^3\tau + 2t_{start} + (3 \times (\tfrac{\mu}{2^m} + \tfrac{\mu}{k}) \times (\tfrac{N}{\mu})^2)t_{comm}\ ) \\ &= 2\tfrac{N^3}{P}\tau + 2\mu t_{start} + \tfrac{3\times(k+2^m)N^2}{P}t_{comm} \end{aligned}$$

, where $P = 2^m \times k$.

4.2 Fault-tolerant matrix-multiplication algorithm on $R_s(m,k)$

J. Berntsen [9] proposes a communication efficient matrix multiplication algorithm on n-cube. The n-cube includes a two-dimensional mesh interconnect with wrap around which is mapped by a binary Gray code. The matrix multiplication $C = A * B$ is performed where C, A, and B are full $N \times N$ matrices. Since there are $P = 2^n$ identical processors, the matrices are distributed on a $\sqrt{P} \times \sqrt{P}$ mesh of processors. In order to compute C, the neighbour to neighbour communication and computation for each processor are needed to perform [9]. Based on J. Berntsen's algorithm, we simulate each fault-free subcube of $R_s(m,k)$ as a processing unit. Our major concentration then is to arrange the jobs between each pair of neighboring processing units of the maximal fault-free subcube-ring $R_s(m,k)$. Our algorithm is divided into two phases. The first phase is to redistribute the matrices C, A, and B onto a $R_s(m,k)$. Consider a $R_s(m,k)$, there are k m-subcubes each containing

2^m nodes. Each m-subcube of $R_s(m,k)$ includes a ring which is mapped by a binary Gray code. A two-dimensional mesh interconnect with wrap around containing $k \times 2^m$ processors is formed. The matrices C, A, and B are distributed into a $k \times 2^m$ mesh of processors as follows. Let LCM(x, y) denote the least common multiple of x and y. First, we split the full $N \times N$ matrices into $\mu \times \mu$ submatrices, where $\mu = \text{LCM}(2^m, k)$. The matrix C is divided in square submatrices C_{ab} holding the elements c_{ik}, with $Na/\mu \leq i < N(a+1)/\mu$, $Nb/\mu \leq k < N(b+1)/\mu$ and $0 \leq a, b < \mu$. Similarly, matrices A and B are split up in the same way. Second, we evenly partition the $\mu \times \mu$ submatrices onto $k \times 2^m$ processors of $R_s(m,k)$. As a result, each processor contains $(\mu/2^m) \times (\mu/k)$ submatrices of C, A, and B.

The second phase is to rearrange the jobs of the neighbour to neighbour communications and computations of each processor of $R_s(m,k)$. Since each processor contains $(\mu/2^m) \times (\mu/k)$ submatrices of C, A, and B, it can be viewed as a matrix with $(\mu/2^m)$ rows and (μ/k) columns. The communication and computation are modified as follows. First, each processor must multiply the respectively submatrices of A and B and sum the product to the respective part of C. Then, we let all submatrices of A shifts left one row. The shift left operation of boundary column is achieved by sending leftmost column with $(\mu/2^m)$ submatrices of A to the west and receiving a column with $(\mu/2^m)$ submatrices of A from the east. Third, we let all submatrices of A shifts up one row. The shift up operation of boundary row is achieved by sending the top row with (μ/k) submatrices of B to the north and receiving a row with (μ/k) submatrices of B from the south. After executing above communication and computation steps μ times, matrix C is thus obtained.

The derivation of time cost T_{FM} of the fault-tolerant matrix-multiplication algorithm is described as follows. Assume that τ is the time to do a floating multiplication or addition, t_{comm} is the time to communicate a single real word and t_{start} is the startup time. Our algorithm performs μ iterations of the neighbour to neighbour communication and computation operations. Time cost of computation operations is $2 \times \frac{\mu}{2^m} \times \frac{\mu}{k} \times (\frac{N}{\mu})^3\tau$. Time cost of communication operations is $t_{start} + 3 \times (\frac{\mu}{2^m} \times (\frac{N}{\mu})^2)t_{comm} + t_{start} + 3 \times (\frac{\mu}{k} \times (\frac{N}{\mu})^2)t_{comm}$. Therefore, the total time cost T_{FM} is

$$\begin{aligned} T_{FM} &= \mu \ (\ 2 \times \tfrac{\mu}{2^m} \times \tfrac{\mu}{k} \times (\tfrac{N}{\mu})^3\tau + 2t_{start} + (3 \times (\tfrac{\mu}{2^m} + \tfrac{\mu}{k}) \times (\tfrac{N}{\mu})^2)t_{comm}\ ) \\ &= 2\tfrac{N^3}{P}\tau + 2\mu t_{start} + \tfrac{3\times(k+2^m)N^2}{P}t_{comm} \end{aligned}$$

, where $P = 2^m \times k$.

4.3 Fault-tolerant sorting algorithm on $C_s(m,k)$

A $C_s(m,k)$ can be directly obtained from $R_s(m,k)$. Based on the Batcher's bitonic sorting algorithm [8] and odd-even sorting algorithm [3], our fault-tolerant sorting algorithm is developed on $C_s(m,k)$ as follows. For a given identified $C_s(m,k)$, there is a sequence of fault-free m-subcube $\{Q_m^1, Q_m^2, \cdots, Q_m^i, \cdots, Q_m^k\}$ with $2^m \times k$ processors. Depending on the value of i is odd or even, sequence $\{Q_m^1, Q_m^2, \cdots, Q_m^i, \cdots, Q_m^k\}$ is classified into odd-numbered and even-numbered subcubes, where $1 \leq i \leq k$. Assume each processor has $\lceil M/(2^m \times k) \rceil$

unsorted elements, where M is the number of total unsorted elements. Initially, we let each node execute the bitonic sorting algorithm such that unsorted elements on each m-subcube of $C_s(m,k)$ are sorted. That is, $\lceil M/(2^m \times k) \rceil$ unsorted elements on each subcube have been sorted such that data elements located on each odd-numbered subcube are sorted in ascending order and located on each even-numbered subcube are sorted in descending order. Next, each fault-free m-subcube of $C_s(m,k)$ is viewed as a processing unit and we repeatedly perform the odd-even-like sorting operations on each pair of processing units of $C_s(m,k)$ as follows.

The odd-even-like sorting operations is divided into two steps as follows. First, all odd-numbered subcubes Q_m^i are activated, where i is the odd number. Consider an odd-numbered subcube Q_m^i and its neighboring subcube Q_m^{i+1}. The compare-exchange and merge-compare-exchange operations of bitonic sorting algorithm [8] are performed on each pair of processors P and P', where P and P' existed in subcubes Q_m^i and Q_m^{i+1}, respectively. Thus smaller data elements of subcubes Q_m^i and Q_m^{i+1} are reserved to subcube Q_m^i and larger data elements are sending to subcube Q_m^{i+1}. Performing the bitonic sorting algorithm again on each subcube so that data elements of each subcube are sorted to be ascending or descending order according to its subcube address is odd or even, respectively. The second step is identical to the first one except that this time even-numbered subcubes are activated. These two steps are repeatedly performed in this order. After $\lceil k/2 \rceil$ iterations, no further exchange data can take place. Finally, data elements of all subcubes will be sorted in the ascending order by using the bitonic sorting algorithm.

The derivation of total time cost T_{FS} of fault-tolerant sorting algorithm is described as follows. Time cost of sorting data elements on each m-subcube is $O(\lceil \frac{M}{2^m \times k} \rceil \log(\lceil \frac{M}{2^m \times k} \rceil) + \frac{m(m+1)}{2} \lceil \frac{M}{2^m \times k} \rceil)$. Our algorithm performs $\lceil k/2 \rceil$ iterations of odd-even-like sorting operations. Time cost of odd-even-like sorting operations is $O(\lceil \frac{3 \times M}{2^m \times k} \rceil + \frac{m(m+1)}{2} \lceil \frac{M}{2^m \times k} \rceil) = O(\lceil \frac{m(m+7)}{2} \lceil \frac{M}{2^m \times k} \rceil)$. Therefore, the total time cost T_{FS} is

$$T_{FS} = O(\lceil \tfrac{M}{2^m \times k} \rceil \log(\lceil \tfrac{M}{2^m \times k} \rceil)) + O(\tfrac{m(m+1)}{2} \lceil \tfrac{M}{2^m \times k} \rceil) + O(\lceil k/2 \rceil \ (2 \times \lceil \tfrac{m(m+7)}{2} \lceil \tfrac{M}{2^m \times k} \rceil))$$
$$= O(\lceil \tfrac{M}{2^m \times k} \rceil \log(\lceil \tfrac{M}{2^m \times k} \rceil)) + \tfrac{m(m(k+1)+7k+1)}{2} \lceil \tfrac{M}{2^m \times k} \rceil).$$

In the case of $\log(\lceil \frac{M}{2^m \times k} \rceil) \geq \frac{m(m(k+1)+7k+1)}{2}$, our fault-tolerant sorting algorithm is a time-cost optimal algorithm.

5. Performance Analysis and Experiment Results

The *largest fault-free subcube* scheme [17] [12] [19], *free-dimension* scheme [20], and *subcube-partitioning* scheme [10] are the present reconfiguration strategies for tolerating faults in injured hypercubes. The largest fault-free subcube scheme results in a tremendous under-utilization of resource and high performance slowdown. The free-dimension scheme [20] has the shortcoming that the number of faults is limited to the order of hypercube dimensions. Here, simulations then mainly compare the execution time of two application algorithms (as

described in Section 4) that are respectively use subcube-partitioning scheme [10] and our reconfiguration scheme.

The percentage of processor utilization of $R_s(m,k)$ is analyzed as follows. It is obviously that the processor utilization of $R_s(m,k)$ is $2^m \times k$ and a healthy Q_n can be transferred to $R_s(n-s, 2^s)$, where $s \leq n$. The $R_s(n-s, 2^s)$ has $2^{n-s} \times 2^s = 2^n$ processors and the $R_s(m,k)$ has $2^{n-s} \times k = k \cdot 2^{n-s}$ processors if $n - s = m$. The percentage of processor utilization of $R_s(m,k)$ is evaluated by the ratio of number of processors used in $R_s(m,k)$ to total number of processors of n-cube. That is, $\frac{k2^{n-s}}{2^s \times 2^{n-s}} = \frac{k}{2^s}$. If $k = 2^s - 1$, then the best percentage of processor utilization is $\frac{2^s-1}{2^s}$. The larger value s is, the higher percentage of processor utilization and the higher diameter of $R_s(m,k)$ will be. There is a tradeoff to determine a suitable value of s such that both the high percentage of processor utilization and the low diameter are achieved. In our scheme, we consider to possibly select the largest subcube to avoid the high diameter problem. Let there exist two suitable fault-free subcube-rings $R_s(3,3)$ (the best selection by identification algorithm described in Section 3.2) and $R_s(2,7)$ in an injured Q_5 with only one faulty node. The percentage of processor utilization of $R_s(3,3)$ and $R_s(2,7)$ are respective 75% and 87.5%, and the diameter of $R_s(3,3)$ and $R_s(2,7)$ are respective 9 and 15. Here we select the $R_s(3,3)$ as our result.

Our simulation is executed on an nCUBE/2E hypercube machines with 32 processors each contains 4 Mega bytes of local memory. In our simulation, two cases of number of faulty processors are assumed. The addresses of faulty processors are randomly generated on each of 10000 simulations for fixed n and $|F|$. In the case of $|F| < n$, the requirement of processor utilization is demand to at least larger than 50% for improving the slowdown factor. If percentage of processor utilization of $R_s(m,k)$ is larger than 50%, the slowdown factor of computation will reduce to be smaller than 2. The percentage of processor utilization of any recognized maximal fault-free subcube-ring $R_s(m,k)$ is larger than 50% when the number of faulty nodes is smaller than n. This is because that there at least existed a $R_s(0, 2^{n-1})$ on an injured n-ncube with $n-1$ faulty nodes [26]. All possible maximal fault-free subcube-ring $R_s(m,k)$, processor utilization, and distributed percentage of processor utilization under fixed n and $|F|$, where $n = 5$, $1 \leq |F| \leq 4$ are shown in Table I. For instance, when $n = 5$ and $|F| = 4$, 7.65% cases of Q_5 can be identified into $R_s(2,4)$ with 50% processor utilization, 38.53% cases can be identified into $R_s(2,5)$ with 62.5% processor utilization, 50.67% cases can be identified into $R_s(2,6)$ or $R_s(3,3)$ with 75% processor utilization, and 3.15% cases can be identified into $R_s(2,7)$ with 87.5% processor utilization. As shown in Table I, 100%, 84.6%, 65.45%, and 50.67% cases to exploit the 75% processor utilization in an injured Q_5 when $|F|$ is 1, 2, 3, and 4 respectively. This indicates that the percentage of processor utilization of maximal fault-free subcube-ring $R_s(m,k)$ is always larger than 50% if $|F| < n$. Furthermore, if the number of F is larger than n, we exploit the maximal fault-free subcube-ring $R_s(m,k)$ such that the processor utilization of $R_s(m,k)$ is as high as possible. Two factors, the value of $|F|$ and the locations of faulty nodes, mainly effect the processor utilization. The percentages

Table I. Distributed percentage of processor utilization of the maximal fault-free subcube-ring $R_s(m,k)$ in injured 5-cube with $|F| = 1, 2, \cdots$, and 8.

Percentage of processor utilization	$R_s(m,k)$	$\|F\|=1$	$\|F\|=2$	$\|F\|=3$	$\|F\|=4$
10% ~ 20%	(1, 2), (1, 3)	0	0	0	0
20% ~ 30%	(1, 4)	0	0	0	0
30% ~ 40%	(2, 3)	0	0	0	0
40% ~ 50%	(1, 7)	0	0	0	0
50% ~ 60%	(2, 4)	0	4.74	5.64	7.65
60% ~ 70%	(2, 5)	0	4.6	26.54	38.53
70% ~ 80%	(2, 6), (3, 3)	100	84.6	65.45	50.67
80% ~ 90%	(2, 7)	0	6.25	4.37	3.15
Percentage of processor utilization	$R_s(m,k)$	$\|F\|=5$	$\|F\|=6$	$\|F\|=7$	$\|F\|=8$
10% ~ 20%	(1, 2), (1, 3)	0	0	0	0.02
20% ~ 30%	(1, 4)	0	0.05	0.11	0.53
30% ~ 40%	(2, 3)	0.51	1.26	3.01	5.51
40% ~ 50%	(1, 7)	2.82	9.28	17.65	27.20
50% ~ 60%	(2, 4)	2.70	8.84	16.10	20.85
60% ~ 70%	(2, 5)	60.28	64.06	55.98	42.74
70% ~ 80%	(2, 6), (3, 3)	31.35	15.99	7.05	3.12
80% ~ 90%	(2, 7)	2.34	0.52	0.10	0.03

of processor utilization under the fixed n and $|F|$, where $n = 5$ and $5 \leq |F| \leq 8$, are also shown in Table I. For instance, there are 96.67%, 89.41%, 79.23%, and 66.74% cases to exploit the processor utilization higher than 50% in an injured Q_5, where $|F|$ is 5, 6, 7, and 8, respectively. The smaller the value of $|F|$ is, the maximal fault-free subcube-ring with high processor utilization generally be determined.

We simulate our fault-tolerant algorithms on $R_s(m,k)$. The proposed algorithms have been simulated on the n-dimensional hypercubes for $n = 5$. In the following, we will discuss these two simulation results. First, the execution result of our fault-tolerant matrix-multiplication algorithm is described. Simulation here compares the execution time of our fault-tolerant matrix-multiplication algorithm and J. Berntsen's [9] matrix-multiplication algorithm with subcube-partitioning scheme [10] in an injured hypercube. Using subcube-partitioning scheme, J. Berntsen's matrix-multiplication algorithm at least has 2 performance slowdown if $|F| < n$. The simulation result of our matrix-multiplication algorithm on $R_s(m,k)$ is depicted in Fig. 8. The number of data elements of matrices C, A, and B are ranged from 64×64 to 320×320. As illustrated in Table I, if Q_5 with $|F| < 4$, $R_s(2,7)$, $R_s(3,3)$, $R_s(2,6)$, $R_s(2,5)$, $R_s(2,4)$ are identified. The slowdown ratio of our algo-

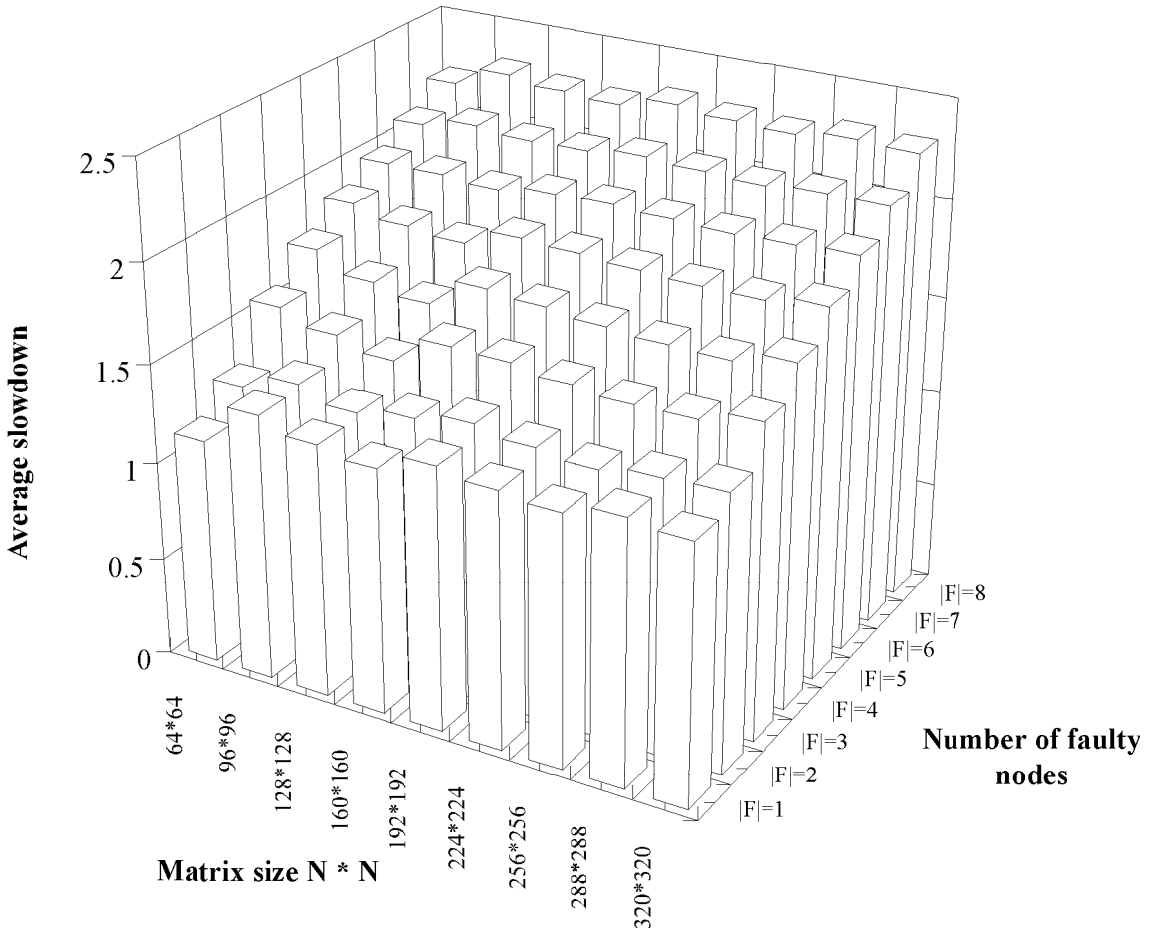


Fig. 8. The average slowdown of the fault-tolerant matrix multiplication algorithm running on 5-cube with set $|F|$ whose value is ranging from 1 to 8.

rithm in $R_s(2,7)$, $R_s(3,3)$, $R_s(2,6)$, $R_s(2,5)$, $R_s(2,4)$ is less than 2 since processor utilization is larger or equal than 50%. As depicted in Fig. 8, when $|F| < 5$, average slowdown ratio of our fault-tolerant matrix-multiplication algorithm running on maximal fault-free subcube-ring $R_s(m,k)$ is smaller than 2. For illustrating algorithms running on our reconfiguration scheme with reasonable slowdown even when $|F| > n$, we also simulate above fault-tolerant matrix-multiplication on faulty 5-cube with different value of set $|F|$, where $5 \leq |F| \leq 8$. As illustrated in Fig. 8, the more the data element is, the low the slowdown ratio obtains. The average slowdown of all cases of running the matrix-multiplication algorithm are smaller than 2.5.

Continually, the simulation result of our fault-tolerant sorting algorithm on $C_s(m,k)$ is discussed. Simulation here compares the execution time of our fault-tolerant sorting algorithm and Batcher's [8] bitonic sorting algorithm with subcube-partitioning scheme [10]. Using subcube-partitioning scheme, the bitonic sorting algorithm running on injured hypercube at least has 2 performance slowdown. The simulation result of our sorting algorithm running

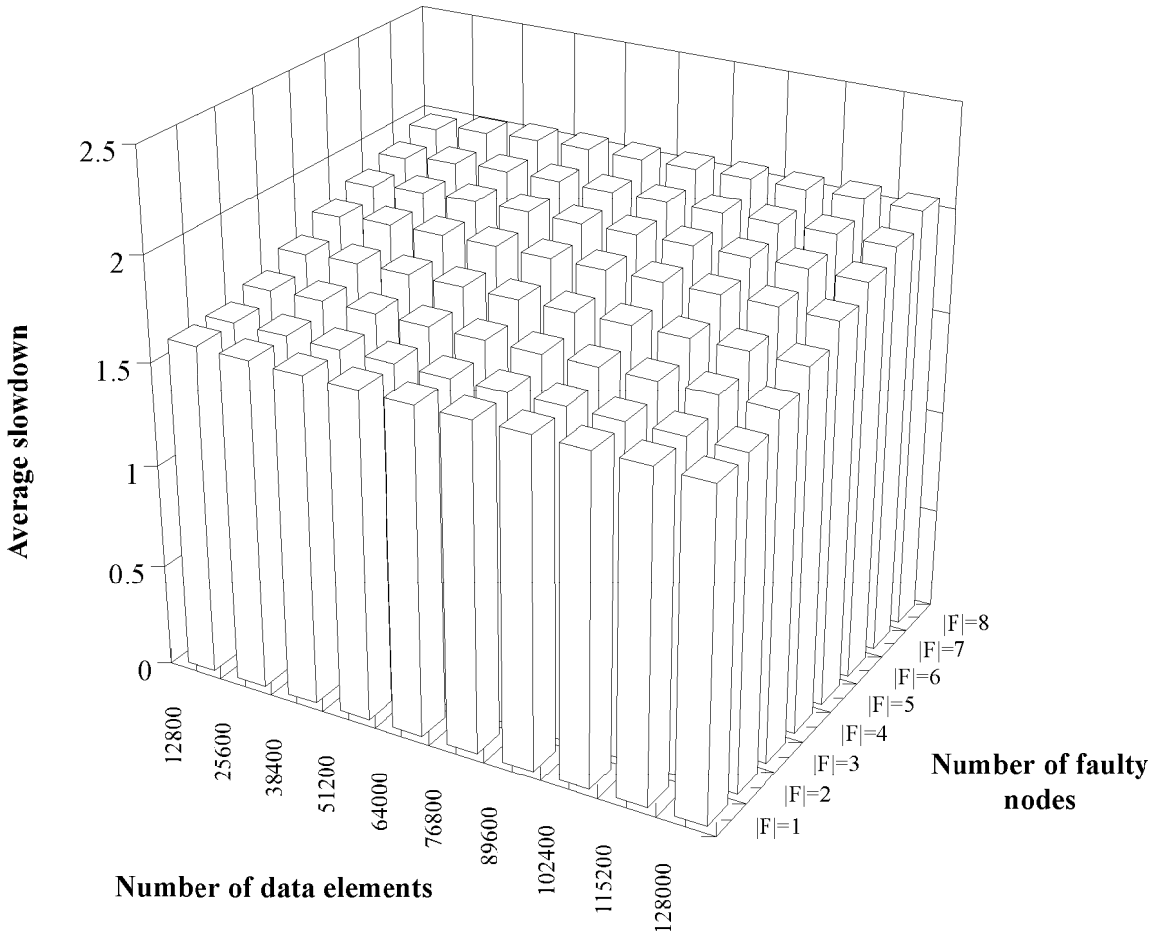


Fig. 9. The average slowdown of the fault-tolerant sorting algorithm running on 5-cube with set $|F|$ whose value is ranging from 1 to 8.

on maximal fault-free subcube-chain $C_s(m,k)$ is illustrated in Fig. 9. The number of data elements is ranged from $1.28 * 10^4$ to $1.28 * 10^5$. From Table I, when $|F| < 5$ on Q_5, $R_s(2,7)$, $R_s(3,3)$, $R_s(2,6)$, $R_s(2,5)$, $R_s(2,4)$ are identified, the $C_s(2,7)$, $C_s(3,3)$, $C_s(2,6)$, $C_s(2,5)$, $C_s(2,4)$ can be obtained. As depicted in Fig. 9, in the case of $|F| < 5$, the average slowdown of the our fault-tolerant sorting algorithm running on $C_s(m,k)$ shown by simulation results is better than the bitonic sorting algorithm using subcube-partitioning scheme. On the other hand, in the case of $5 \leq |F| \leq 8$, we found that the average slowdown ratio of all cases of running the sorting algorithm are slightly larger than 2.

Compared with matrix-multiplication algorithm, sorting algorithm has low communication/computation ratio. From the results of Figs 8 and 9, the maximal fault-free subcube-ring $R_s(m,k)$ is more suitable for running algorithms with low communication/computation ratio. Moreover, when fixed the number of faults, the more the data element is sorted, the low the slowdown ratio is obtained. The proposed scheme, in comparison, is better than the subcube-partitioning scheme of the two fault-tolerant algorithms. As a consequence, using

the maximal fault-free subcube-ring $R_s(m,k)$, the reasonable performance slowdown can be obtained.

6. Conclusion

In this paper, a new reconfiguration approach, identifying the maximal fault-free subcube-ring $R_s(m,k)$, is presented for tolerating more than n arbitrarily placed faults in hypercubes. We can reconfigure an injured hypercube into the maximal fault-free subcube-ring $R_s(m,k)$ with dilation 3 so as lower potential performance degradation is obtained. To demonstrate the fault-tolerant capability of our approach, we implement two fault-tolerant algorithms, matrix-multiplication and sorting, on the nCUBE/2E hypercube machines with 32 processors. Using the approach, if the number of faults is less than n, the *slowdown* ratio of running the application algorithms on $R_s(m,k)$ is smaller than 2. Moreover, if number of faults is not larger than 2^{n-2}, the average *slowdown* ratio of running the application algorithms on $R_s(m,k)$ is smaller than 2.5 by our simulation results.

References

[1] Intel Corporation, "*i*PSC/2 and *i*PSC/860 User's Guide," Intel Corporation, June 1990.

[2] NCUBE Corporation, "nCUBE/2 Processor Manual," NCUBE Corporation, 1990.

[3] S. G. Akl, Parallel Sorting Algorithms, Academic Press, Inc, 1985.

[4] S. G. Akl, The Design and Analysis of Parallel Algorithms, *Prentice-Hall International Editions,* 1989.

[5] F. T. Leighton, Introduction to Parallel Algorithms and Architecture: Array · Tree · Hypercube, *Morgan Kaufmann Publishers,* 1992.

[6] M. S. Alam and R. G. Melhem, "An Efficient Modular Spare Allocation Scheme and Its Application to Fault Tolerant Binary Hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 1, pp. 117-126, Jan. 1991.

[7] P. Banerjee and J. T. Rahmeh, "Algorithm-Based Fault Tolerance on a Hypercube Multiprocessor," *IEEE Transactions on Computers*, Vol. 39, No. 9, pp. 1132-1145, Sep. 1990.

[8] K. Batcher, "Sorting networks and their applications," *Proc. 1968 Spring Joint Comput. Conf.*, Reston, VA: AFIPS Press, 1968, Vol. 32, pp. 307-314.

[9] J. Berntsen, "Communication Efficient Matrix Multiplication on Hypercubes ," *Parallel Computing*, No. 12, pp. 335-342, 1989.

[10] J. Bruck, R. Cypher, and D. Soroker, "Tolerating Faults in Hypercubes Using Subcube Partitioning," *IEEE Transactions on Computers*, Vol. 41, No. 5, pp. 599-605, May 1992.

[11] J. Bruck, R. Cypher, and C. T. Ho, "Fault-Tolerant Mesh and Hypercube Architectures with Minimal Number of Spares," *IEEE Transactions on Computers*, Vol. 42, No. 9, pp. 1089-1104, Sep. 1993.

[12] H. L. Chen and N. F. Tzeng, "Quick Determination of Subcubes in a Faulty Hypercube," *Proc. 1992 International Conference on Parallel Processing*, 1992, Vol. III, pp. 338-345.

[13] H. L. Chen and N. F. Tzeng, "Distributed Identification of All Maximal Incomplete Subcubes in a Faulty Hypercube," *Proc. of the 8th International Parallel Processing Symposium*, 1994, pp. 723-728.

[14] J. Hastad, T. Leighton, and M. Newman, "Reconfiguring a Hypercube in the Presence of Faults," *Proc. of the 19th Annual ACM Symposium on Theory of Computing*, 1987, pp. 274-284.

[15] K. H. Huang and J. A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *IEEE Transactions on Computers*, Vol. C-33, No. 6, pp. 518-528, June 1984.

[16] S. L. Johnsson, "Combining Parallel and Sequential Sorting on a Boolean n-cube," *Proc. 1984 International Conference on Parallel Processing*, 1984, pp. 21-24.

[17] S. Latifi, "Distributed Subcube Identification Algorithms for Reliable Hypercubes," *Information Processing Letters*, Vol. 38, No. 6, pp. 315-321, June 1991.

[18] V. P. Nelson "Fault-Tolerant Computing: Fundamental Concepts," *Computer*, pp. 19-25, July 1990.

[19] F. Özgüner and C. Aykanat, "A Reconfiguration Algorithm for Fault Tolerance in a Hypercube Multiprocessor," *Information Processing Letters*, Vol. 29, No. 5, pp. 247-254, Nov. 1988.

[20] C. S. Raghavendra, P. J. Yang, and S. B. Tien, "Free Dimension - An Effective Approach to Achieving Fault Tolerance in Hypercubes," *Int. Symp. Fault-Tolerant Computing*, 1992, pp. 170-177.

[21] C. S. Raghavendra, M. A. Sridhar, "Prefix Computation on a Faulty Hypercubes," *Proc. 1993 International Conference on Parallel Processing*, 1993, Vol. III, pp. 280-283.

[22] S. R. Seidel and L. R. Ziegler, "Sorting on Hypercubes," *Proc. of the Second Conference on Hypercube Multiprocessors*, 1987, pp. 285-291.

[23] J. P. Sheu, "Fault-Tolerant Parallel k Selection Algorithm in n-cube networks," *Information Processing Letters*, Vol. 39, No. 2, pp. 93-97, July 1992.

[24] J. P. Sheu, Y. S. Chen, and C. Y. Chang, "Fault-Tolerant Sorting Algorithm on Hypercube Multicomputers," *Journal of Parallel and Distributed Computing*, Vol. 16, No. 2, pp. 185-197, Oct. 1992.

[25] S. B. Tien, and C. S. Raghavendra, "Simulation of SIMD Algorithms on Faulty Hypercubes," *Proc. 1991 International Conference on Parallel Processing*, 1991, Vol. I, pp. 716-717.

[26] Y. C. Tseng and T. H. Lai, "Ring Embedding in an Injured Hypercube," *Proc. 1993 International Conference on Parallel Processing*, 1993, Vol. III, pp. 149-152.

[27] P. J. Yang, S. B. Tien, and C. S. Raghavendra, "Embedding of Rings and Meshes onto Faulty Hypercubes Using Free Dimensions," *IEEE Transactions on Computers*, Vol. 43, No. 5, pp. 608-613, May 1994.

[28] P. J. Yang and C. S. Raghavendra, "Embedding and Reconfiguration of Binary Trees in Faulty Hypercubes" *Proc. of the 7th International Parallel Processing Symposium*, 1992, pp. 2-9.