Fault-Tolerant Sorting Algorithm on Hypercube Multicomputers

Jang-Ping Sheu, Yuh-Shyan Chen, and Chih-Yung Chang

Department of Electrical Engineering, National Central
University, Chung-Li 32054, Taiwan, R.O.C.
sheujp@ncu.dnet.ncu.edu.tw

Correspondence Address: Prof. Jang-Ping Sheu
Department of Electrical Engineering, National Central University
Chung-Li 32054, Taiwan, R.O.C.

April 1, 1992

Abstract

In this paper, algorithmic fault-tolerant techniques are introduced for sorting algorithms on n-dimensional hypercube multicomputers. We propose a fault-tolerant sorting algorithm that can tolerate up $n-1$ faulty processors. First, we indicate that the bitonic sorting algorithm can perform sorting operations correctly on the hypercubes with one faulty processor. In order to tolerate up $r \leq n-1$ faulty processors, a partition algorithm is presented. The algorithm partitions the original hypercube with minimum number of cuts into a set of subcubes such that each subcube has at most one faulty processor. The bitonic sorting algorithm can then be applied in each subcube correctly. Finally, each subcube is viewed as a node and a bitonic-like sorting procedure is applied to the subcubes with little communication overhead. Besides, we implement our algorithm on NCUBE/7 MIMD hypercube machines with 64 processors. The simulation results show that the performance of our fault-tolerant sorting algorithm on hypercubes is better than the approach for finding the maximal fault-free subcubes.

Keyword: *Bitonic sorting algorithm, fault tolerance, hypercubes, parallel processing, parallel sorting algorithms.*

1. Introduction

Hypercube multicomputers [16] have become commercially available in the past few years due to its high degree of connectivity, symmetry and low degree of diameter [14]. A great many of scientific algorithms directly developed for the hypercubes have been more efficient than mapping onto other parallel architectures or embedding to other topologies, such as sorting algorithms [15], matrix multiplication [4], network flow problems [17], and graph theories [18]. As the n-dimensional hypercube multicomputers interconnect exactly $N = 2^n$ processors, system performances will be seriously reduced and system resources will be severely consumed when faulty processors/links occur in the hypercube multicomputer. Fault tolerance has then been very important in such a large distributed computing environment for continuing operations of the hypercube multicomputers after failure of one or more processors/links. Efficient sorting algorithms have generally been the fundamental components and factors of many scientific algorithms. Designing a fault-tolerant sorting algorithm that can tolerate $n - 1$ faulty processors on the n-dimensional hypercube multicomputers is, consequently, the purpose of this study.

Most of the recently proposed fault-tolerant schemes address the issue of *reconfiguration* once the faulty processors are identified [1] [3] [6] [8] [12] [13]. The reconfiguration approaches may comprise hardware and software strategies. The hardware reconfiguration strategy suggests a new fault-tolerant hypercube architecture in expectation of high availability and error-free computations. The key concept of the scheme is in employing redundant spare processors for serving a number of normal processors; each of the spare processors, as a result, can replace any detected faulty processor. The use of hardware switches in tolerating faults on hypercubes has first been suggested by Rennels [13]. Chau [6] recently presented a fault-tolerant reconfiguration scheme for hypercubes. The scheme can achieve the same reliability as Rennels' by using more decoupling switches and fewer spare processors. However, Chau's scheme takes a longer time than Rennels' in reconfiguring the environment of hypercubes. Alam [1] also proposed an efficient modular spare allocation method using the same number of spare processors and less switches for achieving the same reliability as Chau's scheme. The module replacement strategy has restored the system to full operations but requires redundant modules which are not used for normal operations. Such a strategy then has a serious shortcoming of high hardware complexity and low processors utilization.

Some researchers in software reconfiguration strategy have exerted themselves to researching fault-tolerant algorithms on hypercubes with one faulty processor [8]. This has been done under the considerations of optimally redistributing the load for each processor, minimizing the effect on the normal processors, and maintaining the low communication overhead. Elster [8] designed fault-tolerant matrix operations on the hypercube multicomputers. The system, however, could only tolerate one faulty processor. The reconfiguration of the fault-tolerant algorithm for handling more than one faulty processor or link on hypercube multicomputers has, up to date, been reduced to finding a subset of fault-free processors that is still connected by the hypercube connection of a lower dimension. Özgüner proposed *the maximum dimensional fault-free subcubes* [12] method for tolerating two or more r faulty processors, $r \geq 2$. Once the faulty processors had been identified, $(n-t)$-dimensional $(1 \leq t \leq n)$ fault-free subcubes could be used while $2^n - 2^{n-t} - r$ normal processors obviously run idle in this strategy. Idle processors are denoted to be the *dangling processors*, which are normal but not used for the fault-tolerant policy. The maximum dimensional fault-free subcubes strategy results in a tremendous underutilization of resources. For example, the resultant working system would be a 5-dimensional hypercube if one faulty processor exists in a 6-dimensional hypercube. This would reduce the performance almost 50% even though less than 2% of the system is faulty. A parallel sorting algorithm is then approached here that can tolerate multiple faults, improve the processors utilization, and provide low communication overhead without any hardware modification.

The faults in the proposed model here are considered to be *permanent* faults [11]. The number of faulty processors is also assumed to be $r \leq n-1$. A processor surrounded by n faulty neighboring processors may exist if the number of faulty processors is $r \geq n$; it can then not send and receive messages to and from the others. The locations of the faulty processors and links are also assumed to be known before running the proposed fault-tolerant sorting algorithm. Some distributed *fault diagnosis* algorithms [2] [5] exist which can be used in identifying the set of faulty processors and links by the fault-free processors. The assumption is reasonable since the *off-line diagnosis* concept proposed by Banerjee [3] can be applied before running the proposed algorithm. The proposed development in an algorithm-based fault-tolerant sorting algorithm has been contributed for handling more than one faulty processor. The bitonic sorting algorithm [10] [15] is first indicated for being able to correctly perform sorting operations on hypercubes with one faulty processor. A partition algorithm with time complexity $O(rN)$ is then presented in order to tolerate $r \leq n-1$ faulty processors, where $N = 2^n$. The purpose of this algorithm is to partition the original hypercube with minimum number of cutting dimensions into a set of subcubes such that each subcube has at most one faulty processor. The bitonic sorting algorithm can then be correctly applied in each subcube. In general, many different subcube partitions exist which will split the hypercube into different sets of subcubes. Communication overhead exists among the subcubes for message-passing during execution of the proposed sorting algorithm. Different partitions will lead to different communication overheads. One of the partitions will be selected here, such that the communication overhead is as less as possible. Each subcube is then viewed as a node and the bitonic-like sorting procedure being applied to these subcubes. The sorting result on hypercubes can, consequently, be obtained in the presence of multiple faults.

The proposed algorithm can reduce more dangling processors than the maximum dimensional fault-free subcubes approach. In particular, if an n-dimensional hypercube has two faulty processors, the n-dimensional hypercube is partitioned into two $(n-1)$-dimensional subcubes each having one faulty processor. Therefore, we do not have any dangling processor. In the worst case, there at most $N/4$ dangling processors exist when an n-dimensional hypercube has $n-1$ faulty processors. It will still have better resources utilization than the maximum dimensional fault-free subcubes approach by which the number of dangling processors is $N/2$ (in the best case) and is $\frac{3}{4}N$ (in the worst case). We also implement the proposed algorithm on an NCUBE/7 MIMD hypercube machine with 64 processors. The performance of the proposed fault-tolerant sorting algorithm is shown by simulation results to be better than that of finding the maximum dimensional fault-free subcube method [12].

The rest of this paper is organized as follows. The bitonic sorting algorithm being able to run correctly on hypercubes with one faulty processor will be indicated in Section 2. A partition algorithm for tolerating multiple faults with minimum number of cutting dimensions such that each subcube will have exactly one faulty processor is proposed. A fault-tolerant sorting algorithm based on the partition algorithm is presented in Section 3. The implementation and performance analysis of the proposed algorithm are discussed in Section 4. The conclusions will finally be presented in Section 5.

2. Partition Algorithm on Faulty Hypercubes

In this section, we first point out that the bitonic sorting algorithm can correctly work on hypercubes with one faulty processor. To tolerate multiple faults, a partition algorithm with a minimum number of cutting dimensions is presented in Section 2.2 for finding a set of subcubes such that each subcube has at most one faulty processor.

2.1 Sorting Operations on Single-Fault Hypercubes

Assume $M >> N$ unsorted elements uniformly distributed to N processors in hypercubes exist. Some dummy keys (∞) will be filled in processors if the distribution of each processor is not uniform. By applying the bitonic sorting algorithm, all the M unsorted elements will be sorted on each processor of the hypercube Q_n in the address order. The key concept of the bitonic sorting algorithm [10] [15] is recursively executing the comparison-exchange operations on each pair of sorted subcubes such that the first half elements are located in one subcube and the last half elements are located in the other subcube. For example, an n-dimensional hypercube Q_n can be divided into two subcubes Q_{n-1} and Q'_{n-1} that one consists of processors P_i for $0 \leq i \leq \frac{N}{2} - 1$ and the other consists of the remaining processors P_j for $\frac{N}{2} \leq j \leq N - 1$. Assume that all the $\lceil M/2 \rceil$ elements in subcube Q_{n-1} (Q'_{n-1}) are sorted in ascending (descending) order. Each pair of processors P_i and P_j $(j = i + \frac{N}{2})$ will then execute the comparison-exchange operations as follows. Processor P_i (P_j) in Q_{n-1} (Q'_{n-1}) will send the first (last) half $\lceil M/(2N) \rceil$ elements to its neighboring processor P_j (P_i) and receive $\lceil M/(2N) \rceil$ elements from P_j (P_i). Processor P_i (P_j) then compares the unsending $\lceil M/(2N) \rceil$ elements with the receiving elements, reserves the smaller (larger) element in each comparison and sends the larger (smaller) element to P_j (P_i). The half smallest $\lceil M/2 \rceil$ elements will then be located in Q_{n-1} and the others will be located in Q'_{n-1}.

The bitonic sorting algorithm can also actually work correctly on hypercube Q_n when the faulty processor is P_0. The M elements can be uniformly distributed to $N-1$ normal processors and treat the faulty processor P_0 in Q_{n-1} as a dead node. Each normal processor has exactly $\lceil M/(N-1) \rceil$ elements. When each pair of processors execute the comparison-exchange operations, the corresponding processor of P_0 just keeps its $\lceil M/(N-1) \rceil$ elements without doing any operation. Assume that Q_{n-1} and Q'_{n-1} are now respectively sorted in ascending and descending orders after some steps of executing the bitonic sorting algorithm. Comparison-exchange operations need to be done here such that the $(\frac{N}{2}-1)(\lceil\frac{M}{N-1}\rceil)$ smallest elements are located in Q_{n-1} and the $(\frac{N}{2})(\lceil\frac{M}{N-1}\rceil)$ largest elements are located in $Q^{'}_{n-1}$. The elements in $P_{\frac{N}{2}}$ are known to be larger than the elements in the other $\frac{N}{2}-1$ processors $P_{\frac{N}{2}+1}$, $P_{\frac{N}{2}+2}$, $\ldots$, and P_{N-1}. If each processor P_i, $1 \leq i \leq N-1$, executes the comparison-exchange operations excepting the processor $P_{\frac{N}{2}}$, all the first $(\frac{N}{2}-1)(\lceil\frac{M}{N-1}\rceil)$ elements will still be located in Q_{n-1} and the others will be located in $Q^{'}_{n-1}$. The bitonic sorting algorithm can obviously work correctly on hypercubes with one faulty processor P_0. If the location of faulty processor is not P_0, its address may be logically set to be 0 and then we reindex the other processors by using *bit-wise exclusive-or* operation on their actual binary addresses. Consequently, we can obtain the correct result on hypercubes with one faulty processor located in arbitrary address.

2.2 The Partition Algorithm

The bitonic sorting algorithm to tolerate one fault as described in Section 2.1 may be applied when a hypercube Q_n has only one faulty processor. If the number of faulty processors in Q_n is $2 \leq r \leq n-1$, the proposed algorithm will partition the hypercube Q_n into subcubes such that each one has at most one faulty processor. For balancing the workload of each subcube, we also determine a dangling processor in each fault-free subcube. The smaller number of cutting dimensions will result in the fewer dangling processors. An optimal partition algorithm is then proposed in this section for finding the minimum number of cutting dimensions. Some terms will be defined before the description of the proposed algorithm.

Definition 1 : Single-fault subcube structure

An n-dimensional hypercube Q_n consisting of 2^n processors can be partitioned into 2^k subcubes, each consisting of 2^{n-k} processors. If each subcube of the partitioned hypercube Q_n has at most one faulty processor, we denote the partitioned hypercube Q_n by the *single-fault subcube structure* F^k_n, where $0 \leq k \leq n-2$.

□

A hypercube Q_5 with 4 faulty processors can be partitioned into single-fault subcube structure F^3_5 as shown in Fig. 1. Consider a hypercube Q_n with r faults for $2 \leq r \leq n-1$. All the possible selections of cutting dimensions can be represented by a cutting dimension tree T_n. A cutting dimension tree T_5 for Q_5 is shown in Fig. 2.

Nodes in level i in tree T_5 denote the ith selected cutting dimension, $1 \leq i \leq 5$. The first cutting dimension can be selected from dimension 0 to dimension 4. Continuing to select the second cutting dimension will be proceeded if the selected dimension can not partition Q_5 into F_5^1. All the cutting dimension sequences that can partition Q_5 into single-fault subcube structure F_5^k $(1 \leq k \leq 3)$ can be obtained by using the depth-first search. The total number of nodes in tree T_5 is $\sum_{i=1}^{5} C_i^5 = 31$.

Let the *address space* of hypercube Q_n be denoted by $\{u_{n-1}u_{n-2} \ldots u_0\}$, for $u_i \in \{0,1\}$, $0 \leq i \leq n-1$. The Q_n can be partitioned into two subcubes Q_{n-1} along dimension d, where $0 \leq d \leq n-1$. The two partitioned subcubes then have address space $\{u_{n-1}u_{n-2} \ldots u_d \ldots u_0\}$ and $\{u_{n-1}u_{n-2} \ldots \bar{u}_d \ldots u_0\}$, where u_d=1 and $\bar{u}_d$=0, respectively. For checking whether the single-fault subcube structure F_n^k has been obtained or not, we construct a checking tree T_n^c as follows. Each node in T_n^c represents a subcube and contains some faulty processors of Q_n. A root of checking tree T_n^c initially reserves all the faulty processors. When a cutting dimension d_1 is traversed in T_n, the faulty processors of a node in T_n^c can be divided along dimension d_1 into two subcubes as its children. If the bit d_1 of a faulty processor is 0 then put the faulty processor in the left child of current node in T_n^c; else put it in the right child. When the current node of the checking tree T_n^c contains more than one faulty processor, the traversal of T_n continues along dimensions d_2, d_3, ..., d_k, $0 \leq k \leq n-2$, until each of terminal nodes in T_n^c has at most one fault. A feasible solution (d_1, d_2, ..., d_k) that can partition Q_n into F_n^k can be obtained. In fact, many feasible solutions exist. Let m be the minimum number of cutting dimensions in all the feasible solutions. These m cutting dimensions d_1, d_2, ..., d_m are collected into D. The D is named the *cutting dimension sequence*, with the value of m being defined as *mincut*.

Definition 2 : Cutting dimension sequence and mincut

The *cutting dimension sequence* $D = (d_1, d_2, \ldots, d_m)$ consists of the m cutting dimensions that can construct a single-fault subcube structure F_n^m with minimum number of cutting dimensions when a hypercube Q_n has $r \leq n-1$ faulty processors. The value m is defined as *mincut*.

□

Consider a Q_4 with 3 faulty processors 0, 6, 9 as shown in Fig. 3(a). The cutting dimension sequence $D = (d_1, d_2) = (1, 3)$ can construct a single-fault subcube structure F_4^2 as shown in Fig. 3(b). A checking tree T_4^c is built as shown in Fig. 4. All the faulty processors are first put in the root of checking tree T_4^c. The address space of Q_4 is $\{u_3u_2u_1u_0\}$. When the first cutting dimension $d_1 = 1$ is traversed in T_4, Q_4 is partitioned into two Q_3 along dimension 1. The faulty processors {0, 6, 9} will be divided into two sets {0, 9} and {6} as children of root as shown in Fig. 4. The two Q_3 then have address spaces $\{u_3u_2 0 u_0\}$ and $\{u_3u_2 1 u_0\}$, respectively. Since the left child of T_4^c in Fig. 4 contains more than one fault, the cutting dimension $d_2 = 3$ is continued to be traversed in T_4 and each Q_3 is then partitioned into two Q_2 along dimension 3. Each of the terminal nodes of the checking tree T_4^c, as a result, only has at most one faulty processor. A single-fault subcube structure F_4^2 is obtained by using the cutting dimension sequence $D = (d_1, d_2) = (1, 3)$.

In general, many different cutting dimension sequences which can partition the Q_n into a single-fault subcube structure F_n^m (under the same value of *mincut* m) exist. Let the *cutting set* Ψ denote all the possible cutting dimension sequences. The issue of finding the cutting set Ψ is now addressed here. A *cutting dimension tree* T_n is first used. All possible cutting dimension sequences are contained in paths from root to terminal nodes in T_n. A depth-first search is first performed on T_n. When each node labeled by d_k is visited at depth k, each subcube Q_{n-k+1} is partitioned into two subcubes Q_{n-k} along dimension d_k. The initial values of the *mincut* and the cutting set Ψ are respectively n and ϕ. The current traversal will be cutoff if the depth k of the cutting dimension tree is larger than the current value of *mincut*. The checking tree T_n^c may be used to check whether a cutting dimension sequence $D = (d_1, d_2, \ldots, d_k)$ can construct a single-fault subcube structure F_n^k or not. If the answer is yes, the cutting set Ψ and the *mincut* value will be modified by the following rule: If the number of cutting dimensions k is less than the current *mincut* value, then set $\Psi = \{D\}$ and *mincut* $= k$, else set $\Psi = \Psi \cup \{D\}$ and the *mincut* value is the same as before. When all nodes of T_n are visited, the minimum value of *mincut* m and the cutting set $\Psi = \{D_1, D_2, \ldots D_\alpha\}$ can be obtained, where α is the number of cutting dimension sequences. All the cutting dimension sequences D_i in Ψ can partition the Q_n into F_n^m. The number of nodes visited in the worst case in tree T_n is $\sum_{i=1}^{n} C_i^n = 2^n - 1 = N - 1$. When travelling each node of T_n, each faulty processor's address should be checked to determine the faulty processor belong to left or right child of current node in T_n^c. The time complexity in determining the cutting set Ψ is then $O(rN)$, where r is number of faulty processors and $N = 2^n$.

Example 1: Consider a Q_5 with 4 faulty processors FP_1, FP_2, FP_3, and FP_4 whose addresses are respectively 00011, 00101, 10000, and 11000. The cutting dimension tree T_5 is constructed in Fig. 2. The cutting set Ψ = $\{D_1, D_2, D_3, D_4, D_5\}$ = $\{(0, 1, 3), (0, 2, 3), (1, 2, 3), (1, 3, 4), (2, 3, 4)\}$ and the *mincut* value $m = 3$ will be obtained if all nodes of T_5 are traversed in the depth-first search. Each of the cutting dimension sequence in Ψ can construct a single-fault subcube structure F_5^3.

In the next section, we will describe how to select one of the cutting dimension sequence D_β from the cutting set $\Psi = \{D_1, D_2, \ldots, D_\alpha\}$ such that the communication overhead for performing the proposed algorithm is as less as possible. A formal algorithm for finding the cutting set Ψ and *mincut* value m is given in the following.

The Partition Algorithm:

Input: An n-dimensional hypercube Q_n with r faulty processors, for $2 \leq r \leq n-1$.

Output: The *mincut* value and cutting set $\Psi = \{D_1, D_2, \ldots D_\alpha\}$.

Step 1: Respectively set the initial value of *mincut* m and cutting set Ψ to n and ϕ.

Step 2: Traverse cutting dimension tree T_n by using the depth-first searching method until all nodes of T_n are visited. Perform **Step 3** when each node in T_n is visited.

Step 3: The traversal is cutoff if the depth k of the current node is larger than the current value of *mincut*. As soon as the node in depth k of T_n labeled by d_k is visited, each subcube Q_{n-k+1} is partitioned into two subcubes Q_{n-k} along dimension d_k. Use the checking tree T_n^c to check whether the current cutting dimension sequence $D = (d_1, d_2, \ldots, d_k)$ can partition the Q_n into a single fault-subcube structure F_n^k or not. If the answer is yes, the cutting set Ψ and the *mincut* value will be modified by the following rule: Set Ψ = $\{D\}$ and *mincut* = k if the number of cutting dimensions k is less than the current *mincut* value; else set $\Psi = \Psi \cup \{D\}$.

The processor utilization in the proposed algorithm is better than the maximum dimensional fault-free subcubes approach. Assume that the number of faulty processors $r \leq n-1$. The number of dangling processors in the proposed partition algorithm is shown to be less than $N/4$ as follows. In an n-dimensional hypercube Q_n, each processor connects to n neighboring processors exactly. Our purpose is to find a F_n^m. A faulty processor in the worst case may connect $n-2$ neighboring faulty processors. We will partition the Q_n at most $n-2$ times along $n-2$ different dimensions. The Q_n will then be partitioned into F_n^{n-2} in which each subcube has 3 normal processors and 1 faulty or dangling processor. Therefore, the processor utilization is at least $\frac{3}{4}N$ in the worst case. When an n-dimensional hypercube has $n-1$ faulty processors, the processor utilization of the maximum fault-free subcubes approach is $N/2$ (in the best case) and is $N/4$ (in the worst case). The proposed algorithm will have better processor utilization than the maximum fault-free subcubes approach. Note that, the proposed partition algorithm is also suitable for faulty hypercube Q_n with $r \geq n$ faulty processors if no normal processor surrounded by n neighboring faulty processors exists.

3. Fault-Tolerant Sorting Algorithm

In the previous section, we have described how to find all the cutting dimension sequences that can partition Q_n into F_n^m with the minimum value m. In this section, we will choose one cutting dimension sequence D_β from Ψ and determine the dangling processor in each fault-free subcube. The fault-tolerant sorting algorithm with presence of multiple faults will first be presented before describing the heuristic method of selecting the D_β and determining the dangling processors.

The notation of address for each subcube and processor used in our fault-tolerant algorithm is first introduced. Assume that the selected cutting dimension sequence is $D_\beta = (d_1, d_2, \ldots, d_m)$. An n-dimensional hypercube Q_n with address space $\{u_{n-1}u_{n-2} \ldots u_0\}$ can be partitioned in order along dimensions d_1, d_2, …, and d_m. By viewing each partitioned subcube as a node, the m-dimensional cube consists of 2^m nodes with m-bit address space $\{v_{m-1}v_{m-2} \ldots v_0\} = \{u_{d_m}u_{d_{m-1}} \ldots u_{d_1}\}$. The rest $s = n - m$ bits form the address space $\{w_{s-1}w_{s-2} \ldots w_0\}$ of the processors on each subcube. A Q_5 with address space $\{u_4u_3u_2u_1u_0\}$ is shown in Figure 5. The cutting dimension sequence $D_\beta = (0, 1, 3)$ will partition the Q_5 into F_5^3 with subcube's address space $\{v_2v_1v_0\} = \{u_3u_1u_0\}$ and 2-dimensional address space $\{w_1w_0\} = \{u_4u_2\}$ of processors in each subcube.

The proposed fault-tolerant sorting algorithm will now be described. Assume that M unsorted elements exist. The partitioned hypercube F_n^m consists of 2^m subcubes, each having one faulty processor or dangling processor. Since the number of normal processors is $N' = 2^n - 2^m = N - 2^m$, the M unsorted elements can be distributed to N' processors and each processor has $\lceil M/N' \rceil$ elements. For a partitioned single-fault subcube structure F_n^m, we perform the reindex operation on each subcube such that the address of the faulty processor in each subcube is 0. Then, we perform the following three sorting steps. Each processor first sorts its elements in ascending or descending order according to its reindexed address is even or odd. Secondly, the bitonic sorting algorithm is applied in each subcube with one faulty processor such that the $\lceil M/2^m \rceil$ elements are sorted in ascending or descending order depending on the address $v_{m-1}v_{m-2} \ldots v_0$ of subcube is even or odd, respectively. Each subcube can finally be viewed as a node and perform a bitonic-like sorting algorithm among subcubes such that the M elements are sorted on Q_n in the subcubes' address order.

Our fault-tolerant sorting algorithm is outlined as follows.

Fault-Tolerant Sorting Algorithm:

Input: A hypercube Q_n contains M unsorted elements and a partitioned single-fault subcube structure F_n^m. The address of each subcube in F_n^m is $v_{m-1}v_{m-2} \ldots v_0$ and the address of each processor in each subcube is $w_{s-1}w_{s-2} \ldots w_0$.

Output: Sorted elements located on F_n^m in the subcubes' address order.

Step 1: Perform the reindex operation on each subcube such that the address of faulty processor in each subcube is 0.

Step 2: The host processor distributes each normal processor $\lceil M/N' \rceil$ elements, where $N' = 2^n - 2^m$. This is since 2^m subcubes each exists which has exactly one faulty processor.

Step 3: Each processor sorts its elements by applying heapsort operations in ascending or descending order according to the reindexed address of the processor is even or odd, respectively. Then the bitonic sorting algorithm is applied on each subcube with one faulty processor such that the $\lceil M/2^m \rceil$ elements are sorted in ascending (descending) order if the address of subcube is even (odd).

Step 4: For i = 0, 1, ..., m - 1 do **Steps 5** through **8**.

Step 5: For each subcube, let variable *mask* be equal to the value of bit v_{i+1} of the subcube's address. Assume $v_m = 0$.

Step 6: For $j = i$, i - 1, ..., 0 do **Steps 7** and **8**.

Step 7: For each pair of neighboring subcubes in dimension j:

(a) If $v_j = 0$ ($v_j = 1$) then each reindexed normal processor sends the first (last) $\lceil M/(2N') \rceil$ elements of its subsequence to its corresponding reindexed normal processor.

(b) If $mask = v_j$ ($mask \neq v_j$), each processor of subcubes compares the unsending elements with the receiving elements, reserves the smaller (larger) element in each comparison and sends the larger (smaller) to its corresponding processor.

(c) Each processor merges the two ordered subsequences in ascending or descending order according to the reindexed address of processor is even or odd, respectively.

Step 8: Applying the bitonic sorting algorithm on each subcube with one faulty processor, the $\lceil M/2^m \rceil$ elements are sorted in ascending (descending) order if $v_{j-1} = mask$ ($v_{j-1} \neq mask$). Assume $v_{-1} = 0$.

For example, consider a Q_5 with 4 faulty processors as described in Example 1. The selected cutting dimension sequence $D_\beta = (0, 1, 3)$ will partition the Q_5 into F_5^3. We determine 4 dangling processors, perform the reindex operation on each subcube, and distribute 47 unsorted elements to reindexed normal processors as shown in Figure 6(a). By applying step 3 of the proposed algorithm, each subcube sorts the assigned 6 unsorted elements in ascending (descending) order if the subcube's address is even (odd) as shown in Figure 6(b). In step 4, the index values of i running on steps 5 through 8 are 0, 1, and 2. When $i = 0$ and $j = 0$, the result of executing steps 7(a) and 7(b) is shown in Figure 6(c). Each processor then performs the merge operation of step 7(c). After executing step 8 of the proposed algorithm with $i = 0$ and $j = 0$, the temporal result is shown in Figure 6(d). Similarly, when $i = 1$ and $j = 1$, the execution results of steps 7 and 8 are respectively shown in Figure 6(e) and 6(f). The results of executing steps 7 and 8 with $i = 1$, $j = 0$ are shown in Figure 6(g) and 6(h), respectively. Continually performing the steps 7 and 8, all the elements can be sorted as shown in Figure 6(i).

In the cost estimation given below, symbol $t_{s/r}$ denotes the cost of sending or receiving an element between two neighboring processors; symbol t_c denotes the time cost of comparing a pair of elements. The derivation of total time cost T of the proposed fault-tolerant sorting algorithm is described as follows. The worst case of time cost for heapsort operations in step 3 is $[(\lceil \frac{M}{N'} \rceil - 1)\log\lceil \frac{M}{N'} \rceil + 1]\ t_c$. The bitonic sorting operations [15] perform $\frac{s(s+3)}{2}$ loops, each with time cost $[\lceil \frac{M}{N'} \rceil t_{s/r} + (\lceil \frac{3M}{2N'} \rceil - 1)t_c]$. The total time cost for step 3 is then

$$[(\lceil \frac{M}{N'} \rceil - 1)\log\lceil \frac{M}{N'} \rceil + 1]t_c + \frac{s(s+3)}{2}[\lceil \frac{M}{N'} \rceil t_{s/r} + (\lceil \frac{3M}{2N'} \rceil - 1)t_c].$$

The hops between two corresponding reindexed normal processors in step 7(a) are at most $s + 1$ since they are located in the s-dimensional neighboring subcubes. The time cost in the worst case for steps 7(a) and 7(b) are respectively $(s + 1)\lceil \frac{M}{2N'} \rceil t_{s/r}$ and $(s+1)\lceil \frac{M}{2N'} \rceil t_{s/r} + (\lceil \frac{M}{2N'} \rceil - 1)t_c$. The time cost for merge operation in step 7(c) is

$(\lceil \frac{M}{N'} \rceil - 1)t_c$. In step 8, the bitonic sorting algorithm performs $\frac{s(s+3)}{2}$ loops, each with time cost $[\lceil \frac{M}{N'} \rceil t_{s/r} + (\lceil \frac{3M}{2N'} \rceil - 1)t_c]$. Steps 4 and 6 of the proposed algorithm perform $\frac{m(m+3)}{2}$ loops of steps 7 and 8. The total time cost T of the proposed fault-tolerant sorting algorithm in the worst case is then

$$\begin{aligned} T = {} & [(\lceil \tfrac{M}{N'} \rceil - 1)\log\lceil \tfrac{M}{N'} \rceil + 1]t_c + \tfrac{s(s+3)}{2}[\lceil \tfrac{M}{N'} \rceil t_{s/r} + (\lceil \tfrac{3M}{2N'} \rceil - 1)t_c] + \\ & \tfrac{m(m+3)}{2}\{(s+1)\lceil \tfrac{M}{N'} \rceil t_{s/r} + (\lceil \tfrac{M}{2N'} \rceil - 1)t_c + (\lceil \tfrac{M}{N'} \rceil - 1)t_c + \\ & \tfrac{s(s+3)}{2}[\lceil \tfrac{M}{N'} \rceil t_{s/r} + (\lceil \tfrac{3M}{2N'} \rceil - 1)t_c]\} \\ = {} & \mathrm{O}(\max(\lceil \tfrac{M}{N'} \rceil \log\lceil \tfrac{M}{N'} \rceil, m^2 s^2 \lceil \tfrac{M}{N'} \rceil)). \end{aligned}$$

In the previous section, we have the time cost $O(rN)$ for the partitioning algorithm. The total cost for sorting M elements on n-dimensional hypercube is $T + O(rN)$. When the number of unsorted elements is large enough ($M >> N$), the time cost of our algorithm is closed to $\mathrm{O}(\lceil \frac{M}{N'} \rceil \log\lceil \frac{M}{N'} \rceil)$.

The above algorithm is based upon the assumption that the cutting dimension sequence D_β has been selected from Ψ, with the dangling processor having been determined in each fault-free subcube. A heuristic method for selecting the D_β and determining the dangling processors is proposed in the following. Consider each pair of neighboring subcubes Q_m and $Q_m^{'}$. Let faulty processors FP and FP' exist and have the same local reindexed addresses, being respectively located in Q_m and $Q_m^{'}$. Each pair of normal processors P and P', which have the same address, located in respective subcubes Q_m and $Q_m^{'}$ will execute the comparison-exchange operations in the bitonic-like sorting algorithm. The original address of each normal processor will be changed since all the address of the processors have been reindexed. Processors P and P' may then not be neighboring after performing the reindex operation and need extra-communication overhead for message-passing. The extra-communication overhead between processors P and P' is the same as the Hamming distance of their respective faulty processors FP and FP'. When Q_m and $Q_m^{'}$ perform the comparison-exchange operations along a fixed dimension i (for $0 \leq i \leq m-1$), the extra-communication overhead h_i can be measured by the Hamming distance of the s-bit addresses of faulty processors FP and FP'

$$h_i = HD(FP, FP').$$

The extra-communication overhead h_i of one pair Q_m and $Q_m^{'}$ may be different from the other pairs. Thus, the *max* function is taken here to estimate the turnaround time. The total extra-communication overhead can be estimated by $\sum_{i=0}^{m-1} \max(h_i)$

since the bitonic-like sorting algorithm performs the comparison-exchange operations from dimensions 0 to $m-1$. The selected D_β should satisfy the following minmax function.

$$\underset{0 \leq \beta \leq \alpha}{min} \sum_{i=0}^{m-1} max(h_i). \tag{1}$$

A dangling processor next needs to be determined in each fault-free subcube such that the workload for each subcube is balanced. The dangling processor in each fault-free subcube may be determined by the following heuristic rule: each fault-free subcube determines a dangling processor whose address $w_{s-1}w_{s-2}\ldots w_0$ is the same as the faulty processors' addresses which appear most frequency in the faulty subcubes. The communication among these faulty processors and dangling processors can then be discarded, with the purpose of the low extra-communication overhead being able to be achieved.

Example 2: Cutting dimension sequence $D_1 = (0,\ 1,\ 3)$ is selected here in Example 1. The faulty processors FP_1, FP_2, FP_3, and FP_4 in F_5^3 are located in subcubes with respective addresses 011, 001, 000, and 100 as shown in Figure 5. Three pairs of faulty processors exist, (FP_1, FP_2), (FP_2, FP_3), and (FP_3, FP_4) being located on neighboring subcubes and their respective Hamming distances are $HD(011,\ 001) = 1$, $HD(001,\ 000) = 1$, and $HD(000,\ 100) = 1$. According to formula (1), the total extra-communication cost under the selection of $D_1 = (0,\ 1,\ 3)$ is $\sum_{i=0}^{2} max(h_i) = HD(01,\ 10) + HD(00,\ 01) + HD(10,\ 10) = 3$. The selection of a cutting dimension sequence $D_1 = (0,\ 1,\ 3)$ with communication overhead 3 is depicted in Figure 5. By selecting the other cutting dimension sequences $D_2 = (0,\ 2,\ 3)$, $D_3 = (1,\ 2,\ 3)$, $D_4 = (1,\ 3,\ 4)$, and $D_5 = (2,\ 3,\ 4)$, their respective extra-communication cost can similarly be obtained as 3, 4, 3, and 3. We may select $D_\beta = D_1 = (0,\ 1,\ 3)$ here. A dangling processor in each fault-free subcube is then determined. Processor with address 10 is treated as dangling processor in each fault-free subcube since the addresses of faulty processors $w_1w_0 = 10$ appears most frequency. The dangling processors' addresses 18, 25, 26 and 27 have finally been obtained by combining the 3-bit address $v_2v_1v_0$ and 2-bit address w_1w_0.

4. Implementation and Performance Analysis

The implementation of the proposed fault-tolerant sorting algorithm on an NCUBE /7 MIMD hypercube machine which consists of 64 processors each contains 512 K bytes of local memory will be described in this section. Implementation here logically treats some processors as faulty nodes and does not assign any unsorted element to them; the faulty nodes, as a result, can be run idle. The fault model can be classified into two types. The most serious fault would be one that completely destroys a processor and all links incident to it. Hastad [9] called such faults *total*. A less serious fault, name *partial fault* [9], would be one that destroys just the computational portion of a processor leaving the communication portion of the processor intact as well as the incident links. The data routing in NCUBE/7 is completely determined by the operating system VERTEX. Since the VERTEX may pass messages through the links of faulty processors, simulation is constrained in faults partial property. The *faults total* property can be achieved by rewriting a router to handle the fault-tolerant routing of message-passing [7]. The execution time will be more than the partial fault if the cube Q_n has the fault total property.

Assume that an n-dimensional hypercube has r faulty processors, where $1 \leq r \leq n-1$. An optimal partition algorithm have been proposed in Section 2 which can partition the hypercube Q_n into a single-fault subcube structure F_n^m with the minimum number of cuts. The proposed algorithm determines some dangling processors for balancing the workload of each subcube. The number of dangling processors is less than $N/4$ in the worst case. The relative locations of faulty processors will affect the value of *mincut*. For a fixed dimension n and a number of faulty processors r, the addresses of r faulty processors are randomly generated on hypercubes Q_n 10000 times. The percentages of all possible *mincut* values under fixed n and r, where $3 \leq n \leq 6$ and $0 \leq r \leq n-1$, are shown in Table 1. For instance, when $n = 6$ and $r = 5$, 93.85% cases of Q_n exist which can be partitioned into F_6^3 (*mincut* value m=3) and 6.15% cases exist which can be partitioned into F_6^4 (*mincut* value m=4). The smaller the value of *mincut* is, the fewer the dangling processors will generally be determined. This indicates that our partition algorithm is biased to small number of dangling processors in 93.85% cases.

The percentages of processor utilization in proposed algorithm and in the maximum dimensional fault-free subcubes method can be compared in Table 2. The value of percentage is evaluated by the ratio of total number of normal processors in original cube to total number of actually running processors. For instance, when $n = 6$ and $r = 4$, our algorithm will partition the Q_6 into F_6^2 in the best case (*mincut* value m = 2) and no dangling processor exists. Thus, the percentage of processor utilization is then $(2^6 - 4)/(2^6 - 4) = 100\%$. Q_6 in the worst case should be partitioned into F_6^3 ($m = 3$), depending on the locations of faulty processors. The number of dangling processors is 4 and the percentage of processor utilization is $(2^6 - 4 - 4)/(2^6 - 4) = 93.3\%$. The percentage of processor utilization is 53.3% in the best case and is 26.6% in the worst case by applying the maximum dimensional fault-free subcubes method. The proposed algorithm, in comparison, is better than the maximum dimensional fault-free subcubes method in resource utilization.

The enhancement of processor utilization will reduce the execution time for sorting operations on hypercubes. The proposed algorithm has been simulated on the n-dimensional hypercubes for n= 3, 4, 5, and 6. In our simulation, the number of faulty processors is $0 \leq r \leq n-1$ and the addresses of faulty processors are randomly generated on each of 10000 simulations for a fixed n and r. The simulation result of our algorithm is depicted in Figure 7 by thin lines and the maximum dimensional fault-free subcubes method, i.e., $r = 0$ by thick lines. In Figure 7(a), the number of data elements is ranged from $3.2 * 10^3$ to $3.2 * 10^4$. The execution time of our algorithm in Q_6 with $r = 1$ or $r = 2$ is shown to be less than the bitonic sorting algorithm running on the fault-free subcube Q_5 ($n = 5$). The execution time of our algorithm is also less than the bitonic sorting algorithm running on the fault-free subcube Q_4 ($n = 4$) when $n = 6$ and $r = 3$, 4, or 5. If a hypercube Q_6 has two faulty processors, the performance of the proposed algorithm is then better than the maximum dimensional fault-free subcubes method in both its best case $n = 5$ and worst case $n = 4$. The maximum dimensional fault-free subcubes method in the best case can utilize Q_5. Although the execution time of our algorithm running on Q_6 with $r = 3$, 4, or 5 is more than the execution time of bitonic sorting algorithm running on the fault-free subcube Q_5, the probability that Q_5 can be utilized is small. The execution time of the proposed algorithm running on Q_5 with $r = 1$ or 2 being less than the fault-free subcube Q_4 is illustrated in Figure 7(b). The execution time of $n = 5$ and $r = 3$ or 4 is also less than the fault-free hypercube Q_3. In Example 1, there are 4 faulty processors with addresses 3, 5, 16, and 24 in Q_5. The maximum fault-free subcube able to be utilized is Q_3. The total time cost of the proposed sorting algorithm for

$n = 5$ and $r = 4$ is less than the bitonic sorting algorithm running on Q_3. Similarly, Figures 7(c) and 7(d) display the execution time of the proposed algorithm with n =3 and 4, respectively. The performance of the proposed fault-tolerant sorting algorithm running on hypercubes shown by simulation results is better than the parallel sorting algorithms running on the maximum fault-free subcubes.

5. Conclusions

An algorithm-based fault-tolerant sorting algorithm on hypercube multicomputers, without any hardware modification, has been proposed. The bitonic sorting algorithm has first been indicated to be able to perform sorting operations correctly on the hypercubes with one faulty processor. As the number of faulty processors was $1 \leq r \leq n-1$, a partition algorithm in partitioning a hypercube Q_n into subcubes has been proposed here, such that each subcube contains at most one faulty processor. The proposed partition algorithm has found the *mincut* value m and cutting set Ψ. Heuristic methods for selecting one of the partitions in Ψ has been suggested under the consideration of little communication overhead and determining the dangling processor in each fault-free subcube such that the workload of each subcube has been balanced. A fault-tolerant sorting algorithm based on the partition algorithm has also been proposed that applies the bitonic sorting algorithm to each subcube and sorts the elements on faulty hypercubes. Our algorithm has finally been implemented on an NCUBE/7 MIMD hypercube machine with 64 processors. The execution results has shown that the performance of our fault-tolerant sorting algorithm on hypercubes has been better than the parallel sorting algorithms running on the maximum fault-free subcubes.

References

[1] 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, pp. 117-126, Jan. 1991.

[2] J. R. Armstrong and F. G. Gray, "Fault Diagnosis in a Boolean n Cube Array of Microprocessors," *IEEE Transactions on Computers*, Vol. C-30, No. 8, pp. 587-590, Aug. 1981.

[3] 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.

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

[5] Kabekode V. S. Bhat, "An Efficient Approach for Fault Diagnosis in a Boolean n-Cube Array of Microprocessors," *IEEE Transactions on Computers*, Vol. 32, No. 11, pp. 1070-1071, Nov. 1983.

[6] S. C. Chau and A. L. Liestman, "A Proposal for a Fault-Tolerance Binary Hypercube Architecture," *Digest of papers of the International Symposium Fault-Tolerant Computing*, The Computer Society, IEEE, pp. 323-330, 1989.

[7] M. S. Chen and K. G. Shin, "Adaptive Fault-Tolerant Routing in Hypercube Multicomputers," *IEEE Transactions on Computers*, Vol. 39, No. 12, pp. 1406-1416, Dec. 1990.

[8] A. C. Elster, M. Ü. Uyar, and A. P. Reeves, "Fault-Tolerant Matrix Operations on Hypercube Multiprocessors," *International Conference on Parallel Processing*, Vol. 3, pp. 169-176, 1989.

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

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

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

[12] 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.

[13] D. A. Rennels, "On Implementing Fault-Tolerance in Binary Hypercubes," *Digest of papers of the International Symposium Fault-Tolerant Computing*, The Computer Society, IEEE, pp. 344-349, 1986.

[14] Y. Saad and M. H. Schultz, "Topological Properties of Hypercube," *IEEE Transactions on Computers*, Vol. 37, No. 7, pp. 867-872, Jul. 1988.

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

[16] C. L. Seitz, "The Cosmic Cube," *Communication of the ACM*, Vol. 28, No. 1, pp. 22-33, Jan. 1985.

[17] J. P. Sheu, C. L. Wu, and G. H. Chen, "Selection of the First k Largest Processes in Hypercubes," *Parallel Computing*, No. 11, pp. 381-384, 1989.

[18] J. P. Sheu, N. L. Kuo, and G. H. Chen, "Graph Search Algorithms and Maximum Bipartite Matching Algorithm on the Hypercube Network Model," *Parallel Computing*, No. 13, pp. 245-251, 1990.

[19] S. B. Tien and C. S. Raghavendra, "Simulation of SIMD Algorithms on Faulty Hypercubes," *International Conference on Parallel Processing*, Vol. 1, pp. 716-717, 1991.