Theoretical Computer Science 255 (2001) 649–658
Theoretical Computer Science
www.elsevier.com/locate/tcs

Note

Improved fault-tolerant sorting algorithm in hypercubes

Yu-Wei Chen[a], Kuo-Liang Chung[b,*,1]

[a] *Department of Computer and Information Science, Aletheia University, No. 32, Chen-Li Street, Tamsui, Taipei, 25103 Taiwan, Republic of China*
[b] *Department of Information Management and Institute of Information Engineering, Institute of Computer Science and Information Engineering, National Taiwan University of Science and Technology, No. 43, Section 4, Keelung Road, Taipei, 10672 Taiwan, Republic of China*

Received February 1999; revised July 2000; accepted August 2000
Communicated by M.S. Paterson

Abstract

Consider M unsorted elements and an n-dimensional hypercube H_n with $\lfloor 3n/2 \rfloor - 1$ faulty nodes, where $M \gg N = 2^n$. Employing a newly proposed partition strategy and the light-occupied dimension concept, this paper improves Sheu et al.'s algorithm [Sheu, Chen, Chang, J. Parallel Distributed Comput. 16 (1992) 185] for sorting these M unsorted elements on the faulty H_n. With the same time bound $O((M/N)\log(M/N) + (M/N)\log^2 N)$ as [Sheu et al., 1992], the proposed algorithm can tolerate $\lfloor n/2 \rfloor$ more faulty nodes than Sheu et al.'s algorithm which can tolerate at most $n-1$ faulty nodes. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Fault tolerance; Hypercube; Parallel algorithm; Parallel sorting

1. Introduction

Sorting is one of the most fundamental operations in the computer science community; the hypercube is one of the most versatile and popular networks due to its low diameter, good connectivity, and symmetry [8, 12]. Without providing fault-tolerant capacity, many efficient sorting algorithms [1, 4, 6, 7, 9, 11, 13, 16, 17] have been designed on hypercubes.

* Corresponding author. Tel.: +886-2-27376771; fax: +886-2-27376777.
E-mail addresses: ywchen@email.au.edu.tw, cyw533@ms23.hinet.net (Y.-W. Chen), klchung@cs.ntust.edu.tw (K.-L. Chung).
[1] Supported by NSC87-2213-E011-001/003.

PII: S0304-3975(00)00390-X

Considering faulty hypercubes, it is an important issue to design an efficient fault-tolerant sorting algorithm. Previously, using the concept of minimum number of cut dimensions, Sheu et al. [15] presented the first fault-tolerant sorting algorithm to sort M elements on an n-dimensional faulty hypercube H_n with $f \leqslant n-1$ faulty nodes in $O((M/N)\log(M/N) + M/N\log^2 N)$ time, where $M \geqslant N = 2^n$.

Consider an H_n with $\lfloor 3n/2 \rfloor - 1$ faulty nodes. Employing a newly proposed partition strategy and the light-occupied dimension (LOD) concept, this paper improves Sheu et al.'s sorting algorithm [15] to sort these M unsorted elements on the faulty H_n. With the same time bound $O((M/N)\log(M/N) + (M/N)\log^2 N)$ as [15], the proposed algorithm can tolerate $\lfloor n/2 \rfloor$ more faulty nodes than Sheu et al.'s algorithm which can tolerate at most $n-1$ faulty nodes. The fault-tolerance improvement of this paper is about 50%.

2. Preliminaries

This section consists of three subsections. Section 2.1 describes some notations used in hypercubes and the adopted fault model. Section 2.2 introduces the concept of Sheu et al.'s algorithm [15]. Section 2.3 introduces the concept of the LOD.

2.1. Adopted fault model

An H_n has 2^n nodes and $n2^{n-1}$ edges. Each node in H_n is labeled as $b = b_n b_{n-1} \ldots b_2 b_1$ for $b_j \in \{0, 1\}$ and $1 \leqslant j \leqslant n$, where j denotes the corresponding dimension. In what follows, without confusion, the base of any one binary string is 2. Fig. 1 illustrates an H_4. Two nodes are linked via an edge if and only if their binary strings differ in exactly one binary digit. For example, node 0 (= 0000) is adjacent to nodes 1 (= 0001), 2 (= 0010), 4, and 8 along dimensions 1, 2, 3, and 4, respectively.

H_n can be partitioned into 2^{n-k} SH_k's, where each SH_k is a k-dimensional subcube, spanned by the same k dimensions. Note that in [15], Sheu et al. presented an interesting relabeling method. In their fault-tolerant sorting algorithm, each SH_k can independently relabel its own 2^k nodes.

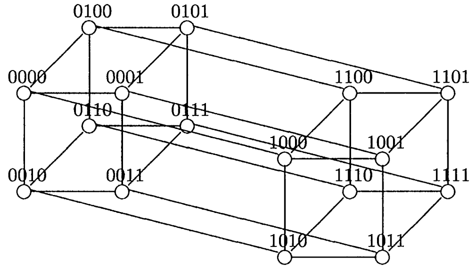

Fig. 1. An H_4.

For example, H_4 can be partitioned into four SH_2's labeled by $0*0*$, $0*1*$, $1*0*$, and $1*1*$, respectively. Two SH_2's are adjacent if their ternary representations differ in exactly one symbol. For simplicity, the four SH_2's labeled by $0*0*$, $0*1*$, $1*0*$, and $1*1*$ are denoted by 0-SH_2, 1-SH_2, 2-SH_2, and 3-SH_2, respectively; each node labeled by $0b_30b_1$ in 0-SH_2, $b_3, b_1 \in \{0, 1\}$, is called node b_3b_1 in 0-SH_2 or node b $(= b_3b_1)$ in 0-SH_2. Using Sheu et al.'s relabeling scheme, the four nodes $\{00, 01, 10, 11\}$ in 0-SH_2 (1-SH_2) can be independently relabeled as $\{10, 11, 00, 01\}$ in 0-SH_2 ($\{11, 10, 01, 00\}$ in 1-SH_2), for example.

This paper uses the same fault model as that in [15]. Throughout this paper, for short, the faulty nodes are called TF nodes; the fault-free nodes are called FF nodes.

2.2. The sketch of Sheu et al.'s sorting algorithm

Initially, applying the concept of the minimum number of cut dimensions [15], H_n with $f \leqslant n-1$ TF nodes is partitioned into 2^{f-1} SH_{n-f+1}'s such that each SH_{n-f+1} contains at most one TF node. In the extreme case, we have $f = n-1$. In this case, each SH_2 containing one TF node independently does the relabeling operation to relabel its own four nodes such that the single TF node is relabeled as node 0 $(=00)$ while the other three FF nodes are relabeled as nodes 1 $(=01)$, 2 $(=10)$, and 3 $(=11)$.

We are given M unsorted elements, $M \gg N = 2^n$. In Sheu et al.'s data assignment scheme, no element is assigned to node 0 in each SH_2, although node 0 may be FF; these M elements are evenly assigned to the FF nodes labeled as 1, 2, and 3 in each SH_2 such that each of such FF nodes holds M/N' $(= M/(3N/4))$ elements.

Then, using the sequential heapsort algorithm each FF node sorts its M/N' elements in ascending (descending) order if its label is even (odd). Sheu et al. have shown that the bitonic sorting algorithm [7, 14] can correctly work on each k-SH_2 with only one TF node labeled by 0 for $0 \leqslant k \leqslant 2^{n-2}-1$. Afterward, the $3M/N'$ elements in k-SH_2 can be sorted in ascending (descending) order if k is even (odd).

Suppose each SH_2 is viewed as a supernode. These 2^{n-2} SH_2's form an $\bar{H}_{n-2}$. Finally, $\bar{H}_{n-2}$ performs a bitonic-like sorting algorithm among these SH_2's such that the M elements are sorted on $\bar{H}_{n-2}$. Consequently, with $O((M/N)\log(M/N)+(M/N)\log^2 N)$ $(= O((M/N')\log(M/N') + (M/N')\log^2 N)))$ time, Sheu et al.'s fault-tolerant sorting algorithm can sort M unsorted elements on H_n with $n-1$ TF nodes. The readers are recommended to refer to paper [15].

2.3. LOD concept

Previously, Yang and Raghavendra [18–20] presented a concept called the degree of occupancy. The degree of occupancy of dimension d in H_n is k, if there exist exactly k links, each link connecting two TF nodes along dimension d. If $k \leqslant 1$, we call dimension d a light-occupied dimension (LOD). For example, suppose the TF nodes of H_4 as shown in Fig. 1 are $\{0001, 1000, 1001, 1010, 1011\}$. The degrees of occupancy of dimensions 1, 2, 3, and 4 are 2, 2, 0, and 1, respectively. Thus, dimensions 3 and

4 are LOD's. Based on the LOD concept, Yang and Raghavendra [19, 20] have shown the property.

Lemma 1 (Yang and Raghavendra [19, 20]). *Given* $f \leqslant \lceil 3n/2 \rceil$ *TF nodes, there exists at least one LOD in* H_n.

In addition, Yang and Raghavendra [19] also presented a distributed algorithm in $O(n)$ time for finding an LOD in H_n with $\lceil 3n/2 \rceil$ TF nodes. After performing this LOD-finding algorithm, each FF node knows the found LOD.

3. New partition strategy based on LOD concept

To improve Sheu et al.'s sorting algorithm in order to have higher fault-tolerant capacity, our newly proposed partition strategy wants the partitioned hypercube to be one of the following two configurations.

The first partition configuration (PC1): H_n is partitioned into 2^{n-2} SH_2's forming $\bar{H}^*_{n-2}$ such that besides at most one SH_2 containing more than one TF node, each of the other SH_2's contains at most one TF node. Section 4 will explain why Sheu et al.'s sorting algorithm [15] can be applied on $\bar{H}^*_{n-2}$ directly.

The second partition configuration (PC2): H_n is partitioned into 2^{n-3} SH_3's forming $\bar{H}^+_{n-3}$ such that besides at most three SH_3's where each SH_3 contains two connected TF nodes, each of the other SH_3's contains at most one TF node. Section 5 will describe how to modify Sheu et al.'s sorting algorithm [15] in order to apply the modified version on $\bar{H}^+_{n-3}$.

In the rest of this section, we present how to classify H_n with $f \leqslant \lfloor 3n/2 \rfloor - 1$ into one of the two configurations PC1 and PC2. From Lemma 1, there exists at least one LOD in H_n with $f \leqslant \lfloor 3n/2 \rfloor - 1 < \lceil 3n/2 \rceil$ TF nodes. Applying the LOD-finding algorithm [19] on H_n, that LOD, say d_1, can be found and known by each FF node. The faulty H_n can be shrunk along dimension d_1 to $\bar{H}_{n-1}$ whose nodes are SH_1's. According to the LOD definition, at most one SH_1 may consist of two TF nodes; each of the other SH_1's contains at most one TF node. For exposition, if an SH_1 contains one or more TF nodes, it is called a faulty SH_1; otherwise, it is called an FF SH_1. Let f' be the number of faulty SH_1's in $\bar{H}_{n-1}$. If $\bar{H}_{n-1}$ contains $f' = f$ faulty SH_1's, each SH_1 has at most one TF node; if $\bar{H}_{n-1}$ contains $f' = f - 1$ faulty SH_1's, at most one SH_1 consists of two TF nodes along dimension d_1.

From Lemma 1, changing n to $n-1$, if the number of faulty SH_1's in $\bar{H}_{n-1}$ is no more than $\lceil 3(n-1)/2 \rceil$ $(= \lfloor 3n/2 \rfloor - 1)$, there still exists at least one LOD in $\bar{H}_{n-1}$. Applying the LOD-finding algorithm [19] on $\bar{H}_{n-1}$, another LOD, say d_2, can be found and known by each FF node. The $\bar{H}_{n-1}$ can be further shrunk along dimension d_2 to $\bar{H}_{n-2}$. From the LOD definition, at most one SH_2 may consist of two faulty SH_1's; each of the other SH_2's contains at most one faulty SH_1.

Based on the above partitioning process, we can further analyze how H_n with $f \leqslant \lfloor 3n/2 \rfloor - 1$ can be classified into PC1 or PC2.

Theorem 2. *An H_n with $f \leqslant \lfloor 3n/2 \rfloor - 1$ TF nodes can be classified into one of two configurations PC1 and PC2.*

Proof. For convenience, call an LOD type-0 or type-1 according to its degree of occupancy. First find a type-0 LOD, if any, then shrink and find a second LOD. This easily gives a PC1.

Otherwise, there is no type-0 LOD, and therefore there will never be any type-0 LOD for any of the later "shrunk" hypercubes. In this case, the first LOD, i.e., d_1, and the second, i.e., d_2, are both of type-1. Since $\lfloor 3n/2 \rfloor - 1 - 2 \leqslant \lceil 3(n-2)/2 \rceil$, there must be a third successive type-1 LOD, say d_3. The first type-1 LOD identifies exactly one edge e in dimension d_1; the second LOD identifies exactly one f, either an edge in dimension d_2 or a diagonal pair in $d_1 \times d_2$; and the third LOD gives exactly one g, an edge in d_3, or a diagonal pair in $d_1 \times d_3$ or $d_2 \times d_3$, or an antipodal pair in $d_1 \times d_2 \times d_3$.

If e, f, and g are all in the same SH_3, there are 3 distinct possibilities and they all yield a PC1. If e, f, and g are in two distinct SH_3's, then a partition along d_1, d_2 or d_3, according to whether the solitary element is e, f, or g, respectively, yields a PC1. If e, f, and g are in three distinct SH_3's, then a partition along d_2 will give a PC1 (by separating f and g) unless g is an edge in d_3 or a diagonal pair in $d_1 \times d_3$. In this case a partition along d_1 will give a PC1 (by separating e, f, and g) unless f and g are both edges. This final remaining case, that each of e, f, and g is an edge, gives a PC2. Because e, f, and g are all exactly one configuration although they have 1, 2, and 3 possible configurations, respectively, any combination among e, f, and g is disjoint each other. Consequently, along d_1, d_2, and d_3 (if any), H_n with $\lfloor 3n/2 \rfloor - 1$ TF nodes can be classified into one of the two configurations PC1 and PC2. □

4. The first partition configuration (PC1): $\bar{H}^*_{n-2}$

For PC1, this section presents how to modify the sorting algorithm [15] slightly in order to achieve higher fault-tolerant capacity.

In PC1, each SH_2 in $\bar{H}^*_{n-2}$ is relabeled such that not only the SH_2 containing more than one TF node is labeled as 0-SH_2, but also the only TF node in any one SH_2 is relabeled as 00 in that SH_2 with the other three FF nodes being relabeled as 01, 10, and 11. We then set 0-SH_2 to be dead and to do nothing. Excluding the dead 0-SH_2, these M unsorted elements are evenly assigned to the FF nodes 01, 10 and 11 in i-SH_2 for $1 \leqslant i \leqslant 2^{n-2} - 1$ such that each of such FF nodes holds M/N' $(= M/(3N/4 - 3))$ elements, where $N' = 3N/4 - 3$.

After performing the data assignment, each node applies a sequential sorting algorithm, e.g., heapsort or quicksort, on its own M/N' elements. Then, applying the bitonic sorting algorithm [7, 14], four nodes in each SH_2 do the bitonic sorting, where if any FF node does any operation with a TF node, such an FF node will do nothing. Thus, except 0-SH_2, all the other SH_2's can successfully perform the bitonic sorting algorithm [7, 14] simultaneously.

From the supernode viewpoint, $\bar{H}^*_{n-2}$ contains only one dead SH_2, i.e., 0-SH_2. Then, the fault-tolerant sorting algorithm [15] is applied to $\bar{H}^*_{n-2}$ containing only one dead SH_2. The only difference is that if any SH_2 does any operation with the dead 0-SH_2, such an SH_2 will do nothing. For the completeness of the context, the formal algorithm is listed below. For simplicity, let the two LOD's be $d_1=1$ and $d_2=2$. Each k-SH_2 has an adjacent $k^{(j)}$-SH_2, where $k=k_nk_{n-1}\dots k_{j+1}k_jk_{j-1}\dots k_4k_3$ and $k^{(j)}=k_nk_{n-1}\dots k_{j+1}\bar{k_j}k_{j-1}\dots k_4k_3$ for $k_j\in\{0,1\}$ and $3\leqslant j\leqslant n$.

Algorithm FTSA_1 /* Fault-Tolerant Sorting Algorithm 1 */

Step 1: (*relabeling process*) Each SH_2 in $\bar{H}^*_{n-2}$ is relabeled such that not only the SH_2 containing more than one TF node is relabeled as 0-SH_2, but also the only TF node in any one SH_2 is relabeled as 00 in SH_2 with the other three FF nodes being relabeled as 01, 10, and 11.

Step 2: (*data assignment*) The given M elements are evenly assigned to the FF nodes 01, 10, and 11 in k-SH_2 for $1\leqslant k\leqslant 2^{n-2}-1$. Thus, each FF node used receives M/N' elements, where $N'=3N/4-3$.

Step 3: (*bitonic sorting on each SH_2*) Each node in SH_2 applies the sequential sorting algorithm on its own M/N' elements, such that the M/N' elements in node 10 (node 01 or 11) in SH_2 are sorted in ascending (descending) order. Then, four nodes in each SH_2 do the bitonic sorting [7, 14]. Here, if one FF node does any operation with another TF node, such that an FF node will do nothing. Finally, the $3M/N'$ elements in each k-SH_2 are sorted in ascending (descending) order if $k_3=0$ ($k_3=1$).

Step 4: (*bitonic sorting among SH_2's*)

For $i=3$ to n do: /* The two LOD's are $d_1=1$ and $d_2=2$. */

4.1: For $j=i$ down to 3 do:

4.1.1: (*sending a half data to corresponding adjacent node*) If $k_j=0$ ($k_j=1$) then FF nodes 01, 10, and 11 in k-SH_2 sends the first (last) $M/(2N')$ sorted elements to its corresponding adjacent nodes 01, 10, and 11 in $k^{(j)}$-SH_2, respectively.

4.1.2: (*comparing data and sending the compared data back*) If $k_{i+1}=k_j$ ($k_{i+1}\neq k_j$), where $k_{n+1}=0$, then each FF node holding data in k-SH_2 compares its own elements with the received elements and keeps the smaller (larger) element in each comparison, and sends the larger (smaller) elements for all comparisons to its corresponding adjacent node.

4.1.3: (*merging two ordered subsequences*) Each node 10 (01 or 11) merges the two ordered subsequences in ascending (descending) order.
4.1.4: (*bitonic sorting on each SH_2*) Four nodes in each SH_2 do the bitonic sorting [7, 14]. The $3M/N'$ elements in k-SH_2 are sorted in ascending (descending) order if $k_{j-1} = k_{i+1}$ ($k_{j-1} \neq k_{i+1}$), where $k_2 = 0$.

The time required in Algorithm FTSA_1 [15] is analyzed again as follows. Let symbols $t_{s/r}$ (t_c) denotes the sending and receiving time (the time for comparing a pair of elements). In Algorithm FTSA_1, since Steps 1 and 2 are preprocessing steps, we only focus on the time required in Steps 3 and 4.

In Step 3, the time required in the sequential sorting algorithm is bounded by $((M/N' - 1)\log(M/N') + 1)t_c$. The bitonic sorting algorithm [13] in each SH_2 needs 3 ($= 2 \cdot 3/2$) loops [3], each with time $(M/N')t_{s/r} + (3M/2N' - 1)t_c$. Because two adjacent SH_2's contains at most two TF nodes except 0-SH_2, the distance between two corresponding relabeled nodes in Step 4.1.1 is at most 3. Thus, the time in the worst case required in Steps 4.1.1, 4.1.2, and 4.1.3 are $3(M/2N')t_{s/r}$, $3(M/2N')t_{s/r} + (M/2N' - 1)t_c$, and $(M/N' - 1)t_c$, respectively. In Step 4.1.4, the bitonic sorting needs 3 loops, each with time $(M/N')t_{s/r} + (3M/2N' - 1)t_c$. Totally, there are $(n-2)(n-1)/2$ loops [3] in Steps 4. Consequently, the total time is

$$\begin{aligned} T &= ((M/N' - 1)\log(M/N') + 1)t_c + 3((M/N')t_{s/r} + (3M/2N' - 1)t_c) \\ &\quad + (n-2)(n-1)/2(3(M/N')t_{s/r} + (M/2N' - 1)t_c + (M/N' - 1)t_c \\ &\quad + 3((M/N')t_{s/r} + (3M/2N' - 1)t_c)) \\ &= \mathrm{O}(M/N' \log(M/N') + n^2(M/N')) \\ &= \mathrm{O}(M/N \log(M/N) + (M/N)\log^2 N). \end{aligned}$$

As a result, we have the following theorem.

Theorem 3. *For PC1, with* $\mathrm{O}((M/N)\log(M/N) + (M/N)\log^2 N)$ $(= \mathrm{O}((M/(3N/4 - 3))\log(M/(3N/4 - 3)) + (M/(3N/4 - 3))\log^2 N))$ *time for* $M \gg N = 2^n$, *M elements can be sorted on* $\bar{H}^*_{n-2}$, *with at most* $\lfloor 3n/2 \rfloor - 1$ *TF nodes.*

From Theorem 3, we know that with the same time complexity as [15] for PC1, the proposed algorithm can tolerate $\lfloor n/2 \rfloor$ more TF nodes than [15]. However, Algorithm FTSA_1 cannot be applied to the faulty $\bar{H}^+_{n-3}$ for PC2 directly. In the next section, we further handle PC2.

5. The second configuration (PC2): $\bar{H}^+_{n-3}$

We now present the idea of the proposed improved sorting algorithm on $\bar{H}^+_{n-3}$ with $f \leqslant \lfloor 3n/2 \rfloor - 1$ TF nodes for PC2. Initially, each SH_3 in $\bar{H}^+_{n-3}$ is relabeled such that the two connected TF nodes in any one SH_3 are relabeled as 000 and 001 while the other six FF nodes are relabeled as 010, 011, 100, 101, 110 and 111. The M unsorted elements are evenly assigned to the FF nodes 010, 011, 100, 101, 110 and 111 in each SH_3 such that each of such FF nodes holds M/N' $(=M/(3N/4))$ elements, where $N=2^n$. To present our improved sorting algorithm on $\bar{H}^+_{n-3}$, we need the following lemma.

Lemma 4. *With* $((M/N'-1)\log(M/N')+1+6(3M/2N'-1))t_c+6(M/N')t_{s/r}$ *time, each* SH_3 *with two connected TF nodes can perform a bitonic sorting to sort its own* $6M/N'$ *elements.*

Proof. Initially, each SH_3 is relabeled such that the two connected TF nodes are relabeled as 000 and 001. Each node applies the sequential sorting algorithm on its own M/N' elements. The time required for the sequential sorting algorithm is bounded by $((M/N'-1)\log(M/N')+1)t_c$. Let SH_3 be divided into 0-SH_2 and 1-SH_2. Assume that elements in 0-SH_2 and 1-SH_2 are sorted in ascending and descending orders, respectively, after three steps of executing the bitonic sorting algorithm. When we merge two ordered subsequences in 0-SH_2 and 1-SH_2, initially, nodes 00 and 01 in 1-SH_2 also do nothing because nodes 00 and 01 in 0-SH_2 are TF. Because these M/N' elements in node 0 (1) in 1-SH_2 are larger than $2(M/N')$ elements, thus we know that the elements in 1-SH_2 must be larger than those in 0-SH_2 holding only $2(M/N')$ elements. Repeatedly, these $6M/N'$ elements can be sorted on each SH_3 with two connected TF nodes. The bitonic sorting [14] on each SH_3 totally needs 6 $(=3(3+1)/2)$ loops, each with time $((M/N')t_{s/r}+((3M/2N')-1)t_c)$. As a result, the total time is $((M/N'-1)\log(M/N')+1+6(3M/2N'-1))t_c+6(M/N')t_{s/r}$. □

From Lemma 4, each k-SH_3 for $0\leqslant k\leqslant 2^{n-3}-1$ performs the bitonic sorting algorithm [7, 14] to sort its own $6M/N'$ elements in ascending (descending) order if k is even (odd). From the supernode viewpoint, these 2^{n-3} SH_3's form $\bar{H}^+_{n-3}$. Therefore, we only modify Algorithm FTSA_1 slightly. Replacing the term SH_2, the initial value 3, and ultimate value 3 in the two loops of Step 4 in Algorithm FTSA_1 by SH_3, 4, and 4, respectively, the modified version of Algorithm FTSA_1 can be applied to the $\bar{H}^+_{n-3}$. The time required in this step is analyzed as follows. Because each SH_3 contains at most two connected TF nodes, the diameter in such an SH_3 is still 3. Thus, the distance between any two FF nodes used in two adjacent SH_3's is at most 7 $(=3+1+3)$. Thus, the time in the worst case for Steps 4.1.1 and 4.1.2 are $7(M/2N')t_{s/r}$ and $7(M/2N')t_{s/r}+(M/2N'-1)t_c$, respectively. In Step 4.1.3, the time is $(M/N')t_c$. In Step 4.1.4, the bitonic sorting performs 6 loops, each with time $((M/N')t_{s/r}+((3M/2N')-1)t_c)$. Step 4 needs $(n-3)(n-2)/2$ loops [3].

Consequently, the total time is

$$
\begin{aligned}
T=&((M/N'-1)\log(M/N')+1+6(3M/2N'-1))t_c+6(M/N')t_{s/r}\\
&+(n-3)(n-2)/2(7(M/2N')t_{s/r}+7(M/2N')t_{s/r}+(M/2N'-1)t_c\\
&+(M/N')t_c+6((M/N')t_{s/r}+((3M/2N')-1)t_c)
\end{aligned}
$$

$$= \mathrm{O}(M/N' \log(M/N') + n^2(M/N'))$$
$$= \mathrm{O}(M/N \log(M/N) + (M/N)\log^2 N).$$

As a result, we have the following theorem.

Theorem 5. *For PC2, with* $\mathrm{O}((M/N)\log(M/N) + (M/N)\log^2 N)$ *time*, $M \geqslant N = 2^n$, M *elements can be sorted on the faulty* $\bar{H}_{n-3}^{+}$.

Combining Theorem 3 and 5, we have the main result.

Theorem 6. *With* $\mathrm{O}((M/N)\log(M/N) + (M/N)\log^2 N)$ *time*, $M \geqslant N = 2^n$, M *elements can be sorted on the faulty* H_n *with* $\lfloor 3n/2 \rfloor - 1$ *TF nodes.*

6. Conclusions

The significance of sorting is due to its popular use in science and engineering. Our main contribution is to show that with $\mathrm{O}((M/N)\log(M/N) + (M/N)\log^2 N)$ time, $M \geqslant N = 2^n$, M elements can be sorted on the faulty H_n with $\lfloor 3n/2 \rfloor - 1$ TF nodes. Our algorithm can tolerate $\lfloor n/2 \rfloor$ more faulty nodes than Sheu et al.'s algorithm [15] under the same time bound. The fault-tolerance improvement is about 50%. In addition, using some variants of the proposed partition strategy and a newly proposed delay-update scheme, we have presented an efficient fault-tolerant algorithm for prefix computation [5]. It is an interesting research issue to plug the fault-tolerance consideration into the randomized sorting networks [2, 10].

Acknowledgements

The authors would like to thank the anonymous referees and Editor Prof. Mike Paterson for their valuable comments that lead to the improved presentation of our paper. In addition, we would like to thank Prof. D. Frank Hsu for his valuable comments.

References

[1] B. Abali, F. Özgüner, A. Bataineh, Balanced parallel sort on hypercube multiprocessors, IEEE Trans. Parallel Distributed Systems 4 (1993) 572–581.
[2] M. Ajtai, J. Komlós, E. Szemerédi, An O($n$log$n$) sorting network, Proc. 1983 ACM Symp. on Theory of Computing, 1983, pp. 1–9.
[3] S.G. Akl, Parallel Sorting Algorithms, Academic Press, Inc., New York, 1985.
[4] K.E. Batcher, Sorting networks and their applications, Proc. AFIPS 1968 SJCC, 1968, pp. 307–314.
[5] Y.W. Chen, K.L. Chung, Efficient prefix computation on faulty hypercubes, J. Inform. Sci. Eng., 2000, to appear.
[6] G. Fox, M. Johnson, G. Lyzenga, S.O.J. Salmon, D. Walker, Solving Problems on Concurrent Processors, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[7] S.L. Johnsson, Combining parallel and sequential sorting on a boolean n-cube, Proc. 1984 Internat. Conf. on Parallel Processing, 1984, pp. 444–448.
[8] F.T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann Pub., CA., 1992 (Chapter 3).
[9] E.W. Mayr, C.G. Plaxton, Pipelined parallel prefix computations, and sorting on a pipelined hypercube, J. Parallel Distributed Comput. 17 (1993) 374–380.
[10] M.S. Paterson, Improved sorting networks with $O(\log N)$ depth, Algorithmica 5 (1990) 75–92.
[11] C.G. Plaxton, Load balancing, selection and sorting on the hypercube, Proc. 1989 ACM Symp. Parallel Algorithms and Architectures, 1989, pp. 64–73.
[12] Y. Saad, M.H. Schultz, Topological properties of hypercube, IEEE Trans. Comput. 37 (1988) 867–872.
[13] S.R. Seidel, W.L. George, Binsorting on hypercubes with d-port communication, Proc. Third Conf. Hypercube Concurrent Comput. and Appl., January 1988, pp. 1455–1461.
[14] S.R. Seidel, L.R. Ziegler, Sorting on hypercubes, Proc. 2nd Conf. on Hypercube Multiprocessors, 1987, pp. 285–291.
[15] J.P. Sheu, Y.S. Chen, C.Y. Chang, Fault-tolerant sorting algorithm on hypercube multicomputers, J. Parallel Distributed Comput. 16 (1992) 185–197.
[16] B. Wagar, Hyperquicksort, Proc. 2nd Conf. on Hypercube Multiprocessors, 1987, pp. 292–299.
[17] Y. Won, S. Sahni, A balanced bin sort for hypercube multicomputers, J. Supercomput. 2 (1988) 435–448.
[18] P.J. Yang, S.B. Tien, C.S. Raghavendra, Embedding of rings and chains onto faulty hypercubes, Tech. Rep., Dept. Elec. Eng., Univ. of South California, 1990.
[19] P.J. Yang, C.S. Raghavendra, Reconfiguration of binary trees in faulty hypercubes, Proc. 7th Internat. Parallel Processing Symp., 1993, pp. 401–405.
[20] P.J. Yang, C.S. Raghavendra, Embedding and reconfiguration of binary trees in faulty hypercubes, IEEE Trans. Parallel Distributed Systems 7 (1996) 237–245.