

Efficient Prefix Computation on Faulty Hypercubes

YU-WEI CHEN AND KUO-LIANG CHUNG⁺

Department of Computer and Information Science

Aletheia University

Tamsui, Taipei, Taiwan 251, R.O.C.

E-mail: ywchen@email.au.edu.tw

⁺*Department of Information Management*

Institute of Computer Science and Information Engineering

National Taiwan University of Science and Technology

Taipei, Taiwan 106, R.O.C.

E-mail: klchung@cs.ntust.edu.tw

Consider an n -dimensional SIMD hypercube H_n with $\lfloor 3n/2 \rfloor - 1$ faulty nodes. Given 2^n operands, this paper presents an efficient algorithm for prefix computation on the faulty H_n . Employing the newly proposed delay-update technique and the subcube partition scheme, the proposed algorithm takes $n+5\log n+7$ steps, and it tolerates $\lfloor n/2 \rfloor$ more faulty nodes than does Raghavendra and Sridhar's algorithm [4] although 11 extra steps are needed.

Keywords: complexity analysis, fault tolerance, parallel algorithm, prefix computation, SIMD hypercube

1. INTRODUCTION

Prefix computation is one of the most fundamental operations in computer science. Let \odot denote a binary associative operator. Given an ordered data set $\langle a_0, a_1, \dots, a_{t-1} \rangle$, $t > 0$, computing $ps_j = a_0 \odot a_1 \odot \dots \odot a_j$ for $0 \leq j < t$ is referred to as the prefix computation. For example, assuming that \odot denotes an addition operator, and that the given data set is $\langle 4, 2, 1, 7 \rangle$, then the corresponding prefix sums are $ps_0 = 4$, $ps_1 = 6$, $ps_2 = 7$, and $ps_3 = 14$. Throughout this paper, calculating prefix sums is used to represent prefix computation. Lacking the fault-tolerant capability, some efficient algorithms [3, 5] for prefix computation have been designed based on SIMD hypercubes.

For a hypercube multiprocessor composed of a large amount of processors, it is impossible to ensure that every processor will work normally. Because some nodes of a hypercube may be faulty and prefix computation arises in a wide variety of applications [3], we are motivated to design an efficient fault-tolerant algorithm for prefix computation.

With $f \leq n - 1$ faulty nodes and 2^n operands, using the free dimension concept [10], Raghavendra and Sridhar [4] presented the first fault-tolerant algorithm for prefix com-

Received October 11, 1999; revised February 22, 2000; accepted May 3, 2000.

Communicated by Wen-Lian Hsu.

*This research was supported in part by the National Science Council, under contract number NSC87-2213-E011-001.

putation in $n+5\log n - 4$ steps on an n -dimensional SIMD hypercube, say H_n . In the SIMD hypercube, all the nodes can exchange messages with their neighbors along a specific dimension in each step. In [4], it is assumed that each step denotes one communication step and a few basic arithmetic operations. Throughout this paper, we make the same assumption.

Consider an SIMD H_n with $\lfloor 3n/2 \rfloor - 1$ faulty nodes. Given 2^n operands, this paper presents an improved algorithm with higher fault-tolerance capability for prefix computation on the faulty H_n . Employing the newly proposed the delay-update technique and the subcube partition scheme, the proposed algorithm takes $n+5\log n+7$ steps, and it tolerates $\lfloor n/2 \rfloor$ more faulty nodes than does Raghavendra and Sridhar's algorithm [4] although 11 extra steps are needed.

The remainder of this paper is organized as follows. Section 2 introduces some preliminaries. Section 3 presents the subcube-partitioning strategy and gives the fault-distribution analyses required in the proposed algorithm. Section 4 presents our data allocation strategy based on the relabelling technique. Section 5 presents the proposed algorithm and gives complexity analyses. Finally, some concluding remarks are given in Section 6.

2. PRELIMINARIES

This section consists of three subsections. Subsection 2.1 introduces some notations of hypercubes and the fault model used. Subsection 2.2 describes one parallel prefix-computation algorithm on the fault-free H_n [5]. Subsection 2.3 reviews the concept of the degree of occupancy [7-9].

2.1 Notations and Fault Model Used

An n -dimensional H_n has 2^n nodes and $n2^{n-1}$ edges. Each node in H_n is labeled by $b_n b_{n-1} \dots b_2 b_1$, $b_j \in \{0, 1\}$ for $1 \leq j \leq n$, where j denotes the corresponding dimension. Two nodes are connected via an edge if and only if their binary strings differ in exactly one bit. For example, node $b_n b_{n-1} \dots b_{j+1} b_j b_{j-1} \dots b_2 b_1$ and node $b_n b_{n-1} \dots b_{j+1} \bar{b}_j b_{j-1} \dots b_2 b_1$ are adjacent along dimension j . Fig. 1 illustrates an H_4 and its four dimensions.

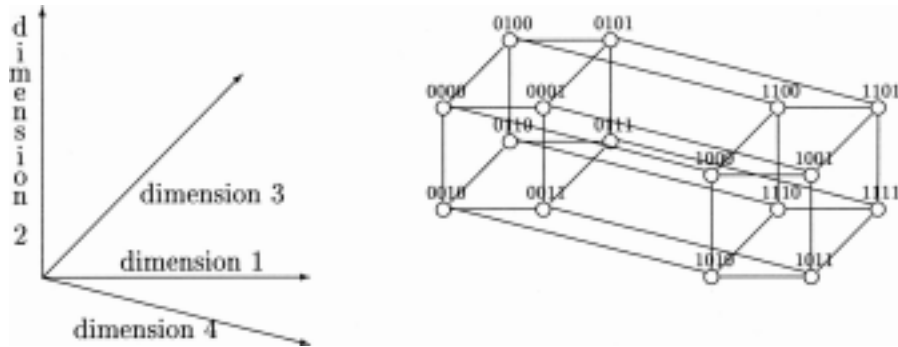


Fig. 1. An H_4 .

H_n can be partitioned into 2^{n-k} k -dimensional subcubes spanned by the same k dimensions, where a k -dimensional subcube is denoted by SH_k . To avoid confusion, we use \bar{H}_{n-k} to denote an $(n-k)$ -dimensional hypercube formed by these 2^{n-k} SH_k 's, where each SH_k is viewed as a supernode. For example, H_4 can be shrunk to \bar{H}_2 , whose four nodes, i.e., four SH_2 's, are labeled $0*0*$, $0*1*$, $1*0*$, and $1*1*$. Two SH_2 's are adjacent if their ternary representations differ in exactly one symbol. For simplicity, the four SH_2 's labeled " $0*0*$ ", " $0*1*$ ", " $1*0*$ ", and " $1*1*$ " are denoted as $0-SH_2$, $1-SH_2$, $2-SH_2$, and $3-SH_2$, respectively; each node labeled $0b_30b_1$ 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 $0-SH_2$.

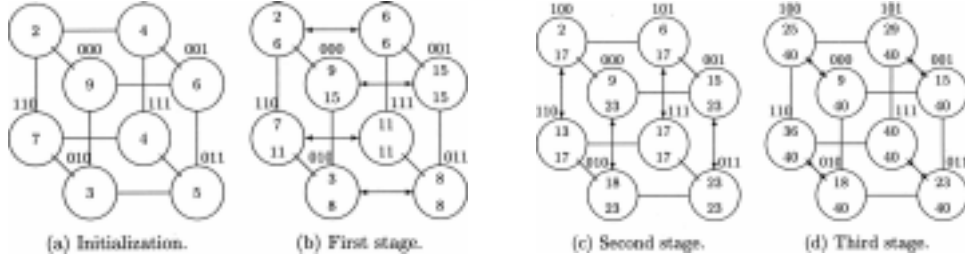
The fault model used in this paper follows the total fault model [2]. In this fault model, it is assumed that the functions for computation and communication in the faulty node are all lost, where such faulty nodes are called truly faulty (TF) nodes. In addition, we assume that when some TF nodes occur in a hypercube, a long time, e.g., several days, is needed for the other nodes to become faulty or to repair these TF nodes. That is, the number of TF nodes will be fixed for a long time. Consequently, the steps required in the preprocessing procedures, such as the relabeling process presented in Section 4, for finding the LOD's (see Subsection 3.1), and partitioning a hypercube into some subcubes (see Sections 3 and 5) should not be included in the steps required in the proposed algorithm for prefix computation.

For exposition, we classify all nodes in a faulty H_n into three types. The first type consists of only TF nodes. The second type consists of only virtual faulty (VF) nodes, which are essentially fault-free nodes, but each of them is totally surrounded by TF nodes. For example, suppose the set of TF nodes shown in Fig. 1 is $\{0001, 0010, 0100, 1000\}$. Node 0000 is a VF node since it is totally surrounded by the four TF nodes. The third type, called fault-free (FF) nodes, consists of the other nodes which neither belong to the first type nor belong to the second type. It is assumed that TF and VF nodes are known in advance.

2.2 Prefix Computation on Fault-Free H_n

Given an ordered data set with 2^n operands, say a_k for $0 \leq k \leq 2^n - 1$, the operand a_k is assigned to node k in H_n . The parallel algorithm without the fault-tolerant capacity [5] for computing prefix sums on H_n is conceptually reviewed as follows. In the first stage, each node exchanges its own operand with its adjacent node along dimension 1. Then, each node keeps the local sum of its own operand and the received operand and computes its corresponding prefix sum. In the i -th stage, $2 \leq i \leq n$, each node exchanges the kept local sum with its adjacent node along dimension i . Then, each node computes the new local sum of its own local sum and the received local sum sent from its adjacent node along dimension i and computes its corresponding prefix sum. After n stages, each node has obtained its corresponding prefix sum and the total sum for 2^n operands. As a result, the parallel algorithm can compute prefix sums on H_n in n steps.

As an example, as shown in Fig. 2a, an ordered data set $\langle 9, 6, 3, 5, 2, 4, 7, 4 \rangle$ is assigned to H_3 . The simulations for the first, second, and third stages are shown in Figs. 2b, 2c, and 2d, respectively. The value on the upper (respectively, lower) side in each node denotes the corresponding prefix sum (respectively, local sum).

Fig. 2. Computing prefix sums on fault-free H_3 .

2.3 The Degree of Occupancy

Previously, Yang and Raghavendra [7, 8] presented the concept of the degree of occupancy. The degree of occupancy along dimension d in H_n is k if there exist exactly k links, with each link connecting two TF nodes, across dimension d . If $k \leq 1$, for convenience, we call dimension d a lightly-occupied dimension (LOD). For example, suppose the set of TF nodes, as shown in Fig. 1, is $\{0000, 0010, 0011, 0101, 1001, 1101\}$. There is only one link connecting two TF nodes across each dimension. Thus, each dimension is an LOD. For such an LOD without considering the case of VF nodes, Yang and Raghavendra [7, 8] presented the following interesting property.

Lemma 2.1: [7, 8] Given $f \leq \lceil 3n/2 \rceil$ TF nodes, there exists at least one LOD in H_n .

Yang and Raghavendra [7] also presented a distributed algorithm in $O(n)$ steps for finding an LOD in H_n with $\lceil 3n/2 \rceil$ TF nodes. After performing the LOD-finding algorithm [7], each FF node knows the LOD.

Previously, Yang, Tien, and Raghavendra [9] used the LOD concept to partition a faulty H_n into $2^{n-3} SH_3$'s for embedding a ring into the faulty H_n . Although their result can be directly applied to partition a faulty H_n into $2^{n-2} SH_2$'s, they did not further analyze the fault-distribution among these $2^{n-2} SH_2$'s. Because each SH_2 is a basic subcube in our fault-tolerant algorithm for prefix computation, the next section will analyze the fault-distribution among these $2^{n-2} SH_2$'s in order to analyze in detail the time complexity required in the proposed algorithm.

3. PARTITIONING STRATEGY AND FAULT-DISTRIBUTION ANALYSES

This section consists of two subsections. Subsection 3.1 presents a strategy to partition H_n with $\lceil 3n/2 \rceil - 1$ TF nodes into $2^{n-2} SH_2$'s such that there exist at most two SH_2 's, with each SH_2 containing more than one TF node, and with each of the other SH_2 's containing at most one TF node. In subsection 3.2, we investigate the fault-distribution for these $2^{n-2} SH_2$'s and give some interesting characteristics.

3.1 Partitioning a Faulty H_n Into SH_2 's

From Lemma 2.1, there exists at least one LOD in H_n with $f \leq \lfloor 3n/2 \rfloor - 1 < \lceil 3n/2 \rceil$ TF nodes. Applying the LOD-finding algorithm [7] 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} , where each node in \bar{H}_{n-1} is SH_1 . 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} . \bar{H}_{n-1} contains $f' \leq \lfloor 3n/2 \rfloor - 1$ faulty SH_1 's when each SH_1 has at most one TF node; it contains $f' \leq \lfloor 3n/2 \rfloor - 2 (= \lfloor 3n/2 \rfloor - 1 - 1)$ faulty SH_1 's when only one SH_1 consists of two TF nodes.

From Lemma 2.1, changing n to $n - 1$, if the number of faulty SH_1 's in \bar{H}_{n-1} is less than or equal to $\lceil 3(n-1)/2 \rceil$, i.e., $\lfloor 3n/2 \rfloor - 1$, then there still exists at least one LOD in \bar{H}_{n-1} . Applying the LOD-finding algorithm [7] 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} , where each node in \bar{H}_{n-2} is SH_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 .

Using the LOD concept, we have presented how to partition H_n with $\lfloor 3n/2 \rfloor - 1$ TF nodes into 2^{n-2} SH_2 's. In the next subsection, we will analyze the distribution of TF nodes among these 2^{n-2} SH_2 's.

3.2 Analyses of Fault-Distribution Among SH_2 's

Since one SH_2 is composed of two SH_1 's, the distribution of all the TF nodes in SH_2 is as outlined in Table 1, where N_1 , N_2 , and N_3 denote the number of TF nodes in one SH_1 , the other SH_1 , and that SH_2 , respectively. As indicated in Table 1, the existence of the case for $N_1 = 2$ and $N_2 = 2$ is impossible; otherwise, there would exist two SH_1 's, i.e., with each SH_1 consisting of two TF nodes, and this would contradict the LOD definition. The symbol 'X' is used to denote this impossibility.

Table 1. Fault-distribution in SH_2 .

N_3		N_1		
		2	1	0
N_2	2	X	3	2
	1	3	2	1
	0	2	1	0

Having investigated the fault—distribution among these 2^{n-2} SH_2 's in \bar{H}_{n-2} , \bar{H}_{n-2} should be one of the following three mutually exclusive cases.

Case 1. Each of these 2^{n-2} SH_2 's in \bar{H}_{n-2} contains at most one TF node.

Case 2. Only one SH_2 in \bar{H}_{n-2} contains three TF nodes, and each of the other $2^{n-2} - 1$ SH_2 's contains at most one TF node.

Case 3. There exist at most two SH_2 's in \bar{H}_{n-2} , with each SH_2 containing two TF nodes, and with each of the other SH_2 's containing at most one TF node. One SH_2 containing two TF nodes must consist of one FF SH_1 and one faulty SH_1 composed of two TF nodes. Thus, the SH_2 must contain two connected TF nodes along dimension d_1 . The other SH_2 containing two TF nodes must consist of two faulty SH_1 's, each containing one TF node. Examining SH_2 , two subcases are as follows:

Case 3a. The other SH_2 contains two connected TF nodes along dimension d_2 .

Case 3b. The other SH_2 contains two disconnected TF nodes.

Theorem 3.1: Given $f \leq \lfloor 3n/2 \rfloor - 1$ TF nodes, H_n can be partitioned into 2^{n-2} SH_2 's which form an \bar{H}_{n-2} such that \bar{H}_{n-2} can be classified into three mutually exclusive cases as described above.

However, the case where VF nodes occur was not considered in the above analyses. Because one VF node cannot communicate with any neighboring FF node, no operand can be assigned to the VF node. Therefore, the case where VF nodes occur is now considered. In the above three mutually exclusive cases, based on the definition of a VF node, it is easy to know that at most one VF node may occur in Case 2 or 3b. When a VF node occurs in Case 2, there exists one SH_2 consisting of one VF node and three TF nodes. When the VF node occurs in Case 3b, there exist one SH_2 containing two connected TF nodes and one SH_2 containing one VF node and two disconnected TF nodes; each of the other SH_2 's contains at most one TF node.

Corollary 3.2: Given $f \leq \lfloor 3n/2 \rfloor - 1$ TF nodes, there exists at most one VF node in H_n .

4. DATA ALLOCATION

This section consists of two subsections. Subsection 4.1 presents a relabeling technique used to make the two SH_2 's, each containing two TF nodes, in Case 3b either adjacent or far enough away from each other. This can lead to communication reduction in the proposed algorithm. Subsection 4.2 presents a data allocation strategy for assigning 2^n input operands to the newly relabeled H_n . We will only focus on the allocation of 2^n operands for Case 3b. After presenting the allocation for Case 3b, we will explain why Case 3b can cover the other cases, including the cases where a VF node occurs.

Without loss of generality in constructing \bar{H}_{n-2} , we still let the two LOD's be $d_1 = 1$ and $d_2 = 2$. In Case 3b, let the SH_2 containing two connected (respectively, disconnected) TF nodes be w - SH_2 (respectively, p - SH_2), where $w = w_n w_{n-1} \dots w_{r+1} w_r w_{r-1} \dots w_4 w_3$ (respectively, $p = p_n p_{n-1} \dots p_{r+1} p_r p_{r-1} \dots p_4 p_3$) and w_i (respectively, p_i) $\in \{0, 1\}$ for $3 \leq i \leq n$. Because the two disconnected FF nodes in p - SH_2 can not communicate directly with each other, we can choose any one FF node to hold the four operands which will be assigned to p - SH_2 ; for simplicity, we choose the FF node with the smaller label, say $b_2 b_1$, $b_2, b_1 \in \{0, 1\}$. The other FF node $\bar{b}_2 \bar{b}_1$ in p - SH_2 is disabled. If node $b_2 b_1$ in one SH_2 which is adjacent to node $b_2 b_1$ in p - SH_2 is TF, then p - SH_2 cannot communicate with this adjacent SH_2 . From the LOD definition, node $b_2 b_1$ in p - SH_2 does have at least one FF neighboring node in an adjacent SH_2 . The q - SH_2 , $q = \min_r \{t = p_n p_{n-1} \dots p_{r+1} p_r p_{r-1} \dots p_4 p_3 \mid \text{node } b_2 b_1 \text{ in } t\text{-}SH_2 \text{ is FF}\}$, is se-

lected to help p - SH_2 communicate with any adjacent SH_2 .

4.1 The Relabeling Process

To ensure that the faulty nodes in the hypercube will affect as few communication steps during prefix computation as few as possible, we need to relabel all the 2^{n-2} SH_2 's in \bar{H}_{n-2} . Using a relabeling technique, the object here is to relabel p - SH_2 and w - SH_2 as either 0 - SH_2 and 1 - SH_2 or 0 - SH_2 and w'' - SH_2 for $w'' \geq n/2$, respectively. The latter implies that if H_n should be partitioned into $2^{n-\log_{n+1}} SH_{\log_{n+1}}$'s, then p - SH_2 and w - SH_2 can be assigned to different $SH_{\log_{n+1}}$'s. In the previous fault-tolerant algorithm for prefix computation [4], H_n was partitioned into $2^{n-\log_n} SH_{\log_n}$'s.

To relabel \bar{H}_{n-2} , two relabeling functions are defined first. The first relabeling function $f_{x,y}(\bar{H}_{n-2})$, $3 \leq x, y \leq n$, is a bit permutation function. Applying $f_{x,y}(\bar{H}_{n-2})$, the bit in the x -th (respectively, y -th) dimension in each SH_2 in \bar{H}_{n-2} is extracted out and put into the leftmost (respectively, rightmost) dimension, while the remaining $n-4$ bits are packed. For example, applying $f_{4,6}(\bar{H}_5)$, each node in \bar{H}_5 with label $b_7b_6b_5b_4b_3$, $b_i \in \{0,1\}$ and $3 \leq i \leq 7$, is relabeled as $b_4b_7b_5b_3b_6$. If $x = y$, applying $f_{x,y}(\bar{H}_{n-2})$, the bit in the y -th dimension in each SH_2 is moved to the rightmost dimension while the remaining $n-3$ bits are packed. For example, applying $f_{4,4}(\bar{H}_5)$, each node in \bar{H}_5 with label $b_7b_6b_5b_4b_3$ is relabeled as $b_7b_6b_5b_3b_4$.

The second relabeling function is $f_z(\bar{H}_{n-2})$, where $z = z_n z_{n-1} \dots z_4 z_3$ and $z_i \in \{0,1\}$ for $3 \leq i \leq n$. Applying $f_z(\bar{H}_{n-2})$, each b - SH_2 , where $b = b_n b_{n-1} \dots b_4 b_3$, in \bar{H}_{n-2} is relabeled by performing a bitwise exclusive-or (XOR) operation: $b_n b_{n-1} \dots b_4 b_3 \oplus z_n z_{n-1} \dots z_4 z_3$.

Now, we will describe how to use the above two relabeling functions can be used to relabel these SH_2 's properly. If $q \neq w$, then there exists a dimension l in w , $3 \leq r \neq l \leq n$, such that $w_l \neq q_l = p_l$. If $q = w$, then l must be equal to r . We first apply the relabeling function $f_{l,r}(\bar{H}_{n-2})$ to each SH_2 and assume that the original b - SH_2 is relabeled as b' - SH_2 . Therefore, p - SH_2 , q - SH_2 , and w - SH_2 are relabeled as p' - SH_2 , q' - SH_2 , and w' - SH_2 , respectively. Then, we apply the function $f_p(\bar{H}_{n-2})$ to each relabeled SH_2 and assume that b' - SH_2 is relabeled as b'' - SH_2 . That is, p' - SH_2 , q' - SH_2 , and w' - SH_2 are relabeled as p'' - SH_2 ($= 0$ - SH_2), q'' - SH_2 ($= 1$ - SH_2), and w'' - SH_2 , respectively.

In Fig. 3 (a), the set of TF nodes and the set of two found LOD's in H_4 are $\{0000, 0001, 0101, 0110, 1101\}$ and $\{1, 2\}$, respectively. The black and white circles denote the TF and FF nodes, respectively; the double circle denotes the disabled FF node. Then, H_4 is partitioned into four SH_2 's, $\langle 0$ - $SH_2 = w$ - $SH_2, 1$ - $SH_2 = p$ - $SH_2, 2$ - $SH_2, 3$ - $SH_2 = q$ - $SH_2 \rangle$. The FF node 00 ($= 0100$) in 1 - SH_2 is chosen to hold four operands while node 11 ($= 0111$) is disabled. There exists an FF node 00 ($= 1100$) in 3 - SH_2 that is adjacent to the node 00 in 1 - SH_2 . After applying $f_{l=3,r=4}(\bar{H}_2)$, $\langle 0$ - $SH_2, 1$ - $SH_2, 2$ - $SH_2, 3$ - $SH_2 \rangle$ are relabeled as $\langle 0$ - $SH_2, 2$ - $SH_2, 1$ - $SH_2, 3$ - $SH_2 \rangle$. After applying $f_2(\bar{H}_2)$ to each relabeled SH_2 , $\langle 0$ - $SH_2, 2$ - $SH_2, 1$ - $SH_2, 3$ - $SH_2 \rangle$ are relabeled as $\langle 2$ - $SH_2, 0$ - $SH_2, 3$ - $SH_2, 1$ - $SH_2 \rangle$. The relabeled H_4 is illustrated in Fig. 3 (b).

Lemma 4.1: All 2^{n-2} SH_2 's in \bar{H}_{n-2} can be relabeled such that p - SH_2 and w - SH_2 are relabeled as either 0 - SH_2 and 1 - SH_2 or as 0 - SH_2 and w'' - SH_2 for $w'' \geq n/2$, respectively, when $n \geq 4$.

Proof: If $q = w$, after applying $f_{i,r}(\bar{H}_{n-2})$, it is easy to see that p - SH_2 and w - SH_2 are relabeled as 0 - SH_2 and 1 - SH_2 , respectively. If $q \neq w$, then after applying $f_{i,r}(\bar{H}_{n-2})$ and $f_p(\bar{H}_{n-2})$, the leftmost dimensions of p'' and w'' are 0 and 1 , respectively. Because the weight of the leftmost dimension is 2^{n-3} , the distance between p'' and w'' is at least 2^{n-3} . When $n \geq 4$, 2^{n-3} is larger than or equal to $n/2$. This completes the proof.

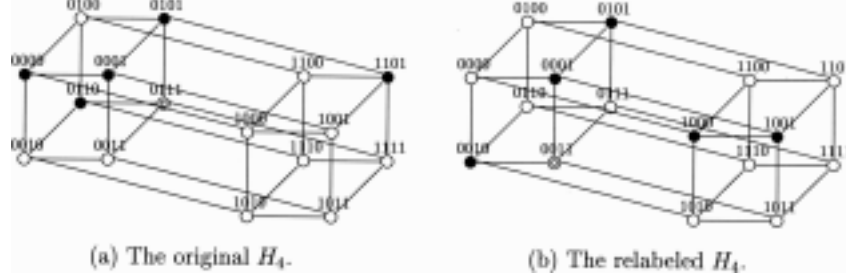


Fig. 3. The relabeled faulty H_4 .

Lemma 4.1 guarantees that using the above two labeling functions, p - SH_2 with two disconnected TF nodes and w - SH_2 with two connected TF nodes can be relabeled such that they are either neighboring or separated by a distance greater than or equal to $n/2$.

4.2 Allocation of 2^n Operands

Having relabelled the faulty \bar{H}_{n-2} , we will now present how the given 2^n operands can be allocated for Case 3b without considering the case in which a VF node occurs.

The first four operands, a_0, a_1, a_2 , and a_3 , are assigned to the selected FF node b_2b_1 ($= \underbrace{00 \dots 0}_{n-2} b_2 b_1$) in 0 - SH_2 . If 1 - SH_2 consists of four FF nodes, then the four operands, a_4, a_5, a_6 , and a_7 , are assigned to the four FF nodes, namely, $00(= \underbrace{0 \dots 0}_{n-2} 00)01$, 10 , and 11 , respectively. If 1 - SH_2 contains one TF node, then its neighboring node along dimension 1 has to take over the operand in the TF node. If 1 - SH_2 contains two connected TF nodes, say nodes 00 and 01 , then the two operands a_4 and a_5 and the two operands a_6 and a_7 are assigned to the two FF nodes 10 and 11 , respectively. Using the same data allocation as in 1 - SH_2 , the four operands, say $a_{4j}, a_{4j+1}, a_{4j+2}$, and a_{4j+3} for $2 \leq j \leq 2^{n-2} - 1$, are assigned to j - SH_2 . Fig. 4 illustrates an example in which 16 operands, a_i for $0 \leq i \leq 15$, are assigned to the faulty H_4 , i.e., \bar{H}_2 , in Fig. 3 (b) and the 16 operands are assigned to the FF nodes denoted by white circles.

It is clear that Case 3b covers Case 1, so the above data allocation strategy is valid for Case 1. For Case 2, we relabel the \bar{H}_{n-2} such that the SH_2 containing three TF nodes is relabeled as 0 - SH_2 . The first four operands $a_i, 0 \leq i \leq 3$, are assigned to the only FF node in 0 - SH_2 and the remaining $2^n - 4$ operands are assigned to the other SH_2 's based on the same argument. For Case 3a, we also use the same relabeling process as in Case 3b such that the two SH_2 's, each with two connected TF nodes, are relabeled such that they are adjacent or separated by a distance greater than or equal to $n/2$. Consequently, the

relabeling process and the data allocation strategy for Case 3b indeed cover those for the other cases. Therefore, we omit the details for Cases 1, 2, and 3a.

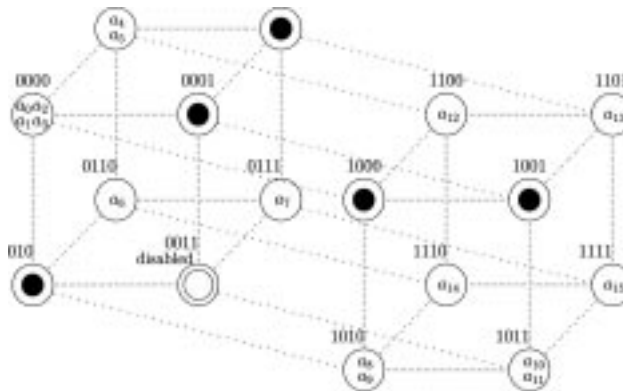


Fig. 4. Assigning 16 operands to 10 FF nodes in a faulty H_4 .

However, we have to consider Case 2 and Case 3b when a VF node occurs. When only one VF node occurs in Case 3b, $0-SH_2$ consists of one VF, one FF, and two disconnected TF nodes. The first four operands are all assigned to the only FF node in $0-SH_2$. Thus, the above data assignment strategy is still available for Case 3b when one VF node occurs.

When only one VF node occurs in Case 2, we evenly assign a_i for $0 \leq i \leq 7$ to the FF nodes in $1-SH_2$ because $0-SH_2$ consists of three TF nodes and one VF node, and it is dead. For example, if $1-SH_2$ consists of four FF nodes, then $\{a_0, a_1\}$, $\{a_2, a_3\}$, $\{a_4, a_5\}$, and $\{a_6, a_7\}$ are assigned to the four FF nodes, namely, 00 ($= \underbrace{00 \dots 000}_{n-2}$) 01 , 10 , and 11 , respectively. If $1-SH_2$ contains one TF node, say node 01 , then the operands $\{a_i \mid 0 \leq i \leq 2\}$, $\{a_i \mid 3 \leq i \leq 5\}$, and $\{a_i \mid 6 \leq i \leq 7\}$ are assigned to nodes 00 , 10 , and 11 , respectively.

5. THE PROPOSED ALGORITHM

This section presents the proposed algorithm for prefix computation on the faulty H_n with $\lfloor 3n/2 \rfloor - 1$ TF nodes. Conceptually, the proposed algorithm consists of three phases, namely, local computation in each subcube, global computation among subcubes, and local update in each subcube. In Phase_1, each subcube simultaneously performs its own prefix sums and the sum of the operands assigned on the same subcube such that each FF node in the same subcube keeps the corresponding prefix sum and the sum. At the end of Phase_1, each FF node in the same subcube lacks the same updated value needed to compute the final prefix sum. In Phase_2, the updated values are obtained. At the end of Phase_2, exactly one FF node in each subcube keeps the updated value for that subcube. In Phase_3, a variant of Phase_1 is used to broadcast the updated value to the other FF nodes in each subcube.

There are two main differences between the proposed algorithm and the one in [4]:

(1) instead of using the free-dimension concept, our algorithm uses the LOD concept and can tolerate $\lfloor n/2 \rfloor$ more TF nodes than can that of [4]; (2) our algorithm uses a novel delay-update technology although this requires a few extra steps. The three phases of the proposed algorithm and the related complexity analyses are described in the following three subsections.

5.1 Phase_1: Local Computation in Each Subcube

To increase the number of TF nodes allowable in H_n from $n - 1$ [4] to $\lfloor 3n/2 \rfloor - 1$ in this paper, instead of partitioning H_n into $2^{n-\log n}$ $SH_{\log n}$'s [4], H_n is partitioned into $2^{n-\log n-1}$ $SH_{\log n+1}$'s in Phase_1, where each $SH_{\log n+1}$ is spanned by dimensions $\log n+1$, $\log n$, $\log n-1$, ..., 2, and 1. In Phase_1, each x - $SH_{\log n+1}$ in $\bar{H}_{n-\log n-1}$, $0 \leq x \leq 2^{n-\log n-1} - 1$, wants to compute its own prefix sums in parallel. In what follows, we will present this phase in more detail.

5.1.1 Performing prefix sums and local sum in each SH_2

First, each FF node holding two or more operands computes its prefix sums sequentially and then provides its own local sum as an operand to be computed with the final prefix sums of the other FF nodes. For example, letting $a_i = i$ in Fig. 4, FF nodes 0, 4, 6, 7, 10, 11, 12, 13, 14, and 15 first computes their prefix sums and provide their local sums 6 (= 0+1+2+3), 9 (= 4+5), 6, 7, 17 (= 8+9), 21 (= 10+11), 12, 13, 14, and 15, respectively, as the operands to be used by the other processors.

Then, each SH_2 spanned by dimensions 1 and 2, i.e., two found LOD's, wants to compute its prefix sums and the local sum. To analyze the steps required in each SH_2 , all 2^{n-2} SH_2 's in Case 3b are classified into four types. Type-1 consists of all FF SH_2 's. Type-2 consists of the SH_2 's, each containing only one TF node. Type-3 consists of only one SH_2 with two connected TF nodes. Type-4 consists of only one SH_2 with two disconnected TF nodes.

The communication patterns for computing prefix sums in SH_2 's are shown in Figs. 5 (a), (b), (c), and (d), respectively, where the solid black circles and the double circles denote the TF nodes and the disabled FF or VF nodes, respectively. The curve denotes the communication between two FF nodes. The numbers 1, 2, and 3 on the curves denote that two FF nodes communicate each other in steps 1, 2, and 3, respectively. From Fig. 5, it is easy to see that all the SH_2 's can compute their own prefix sums and local sums in three steps, simultaneously.

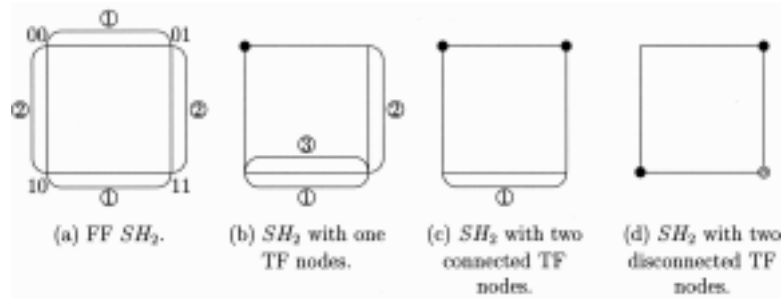


Fig. 5. The four communication patterns for SH_2 's.

Proposition 5.1: Each SH_2 in \bar{H}_{n-2} with $f \leq \lfloor 3n/2 \rfloor - 1$ TF nodes can simultaneously com-

compute its own prefix sums and local sum in three steps.

5.1.2 Exchange operation between two adjacent SH_2 's

We will now define one exchange operation which will be used often later.

Definition 5.1: One **exchange operation** consists of a few communication and computation operations. In the communication operations, two adjacent SH_2 's, say $x-SH_2$ and $y-SH_2$, exchange their two local sums such that each FF node in $x-SH_2$ (respectively, $y-SH_2$) receives the local sum of $y-SH_2$ (respectively, $x-SH_2$). Then, in the computation operations, each FF node in $x-SH_2$ and $y-SH_2$ computes the new local sum of its own local sum and the received local sum; if $x > y$ (respectively, $x < y$), then each FF node in $x-SH_2$ (respectively, $y-SH_2$) computes its corresponding prefix sum by adding the received local sum to its held prefix sum.

In fact, the related exchange operations dominate the number of steps required in the remaining prefix computation on $2^{n-\log n-1} \overline{SH}_{\log n-1}$'s ($= SH_{\log n+1}$'s) of Phase_1. The details will be given in Subsection 5.1.3.

In this subsection, we will analyze the number of steps required in one exchange operation between two adjacent SH_2 's. We only focus on the analyses for Case 3b for the following reasons. It is clear that Case 3b covers Case 1 because each SH_2 in Case 1 is either Type-1 or Type-2 (see Fig. 5). In Case 2, the SH_2 with three TF nodes can be treated as the one with two disconnected TF nodes and one disabled FF node in Case 3b. Thus, it is clear that Case 3b covers Case 2. In Case 3a, there exists one SH_2 (respectively, the other SH_2) with two connected TF nodes along dimension 1 (respectively, 2). If we disable one FF node in the SH_2 with two connected TF nodes along dimension 2, the SH_2 can be treated as the one with two disconnected TF nodes and one disabled FF node in Case 3b. Thus, it is clear that Case 3b covers Case 3a.

Because all the SH_2 's in Case 3b are classified into four types, totally, there are eight communication pairs between any two adjacent SH_2 's, namely, $\langle \text{Type-1, Type-1} \rangle$, $\langle \text{Type-2, Type-1} \rangle$, $\langle \text{Type-3, Type-1} \rangle$, $\langle \text{Type-4, Type-1} \rangle$, $\langle \text{Type-2, Type-2} \rangle$, $\langle \text{Type-3, Type-2} \rangle$, $\langle \text{Type-4, Type-2} \rangle$, and $\langle \text{Type-4, Type-3} \rangle$. Fig. 6 illustrates the eight communication patterns between two SH_2 's involved in the prefix computation.

We will first consider the four pairs $\langle \text{Type-}j, \text{Type-1} \rangle$ for $1 \leq j \leq 4$. Let two adjacent SH_2 's, $x-SH_2 \in \text{Type-}j$ and $y-SH_2 \in \text{Type-1}$. As shown in Figs. 6 (a), (b), (c), and (d), it is easy to check that $x-SH_2$ and $y-SH_2$ do the exchange operation in steps one, two, three, and four if $j = 1, 2, 3$, and 4, respectively.

Proposition 5.2: With at most three steps, one exchange operation for prefix computation between two adjacent SH_2 's, $x-SH_2 \in \text{Type-}j$ for $1 \leq j \leq 4$ and $y-SH_2 \in \text{Type-1}$, can be done.

As shown in Fig. 6 (e), it is easy to check that Proposition 5.3 is correct.

Proposition 5.3: With at most three steps, one exchange operation for prefix computation between two adjacent SH_2 's, $x-SH_2 \in \text{Type-2}$ and $y-SH_2 \in \text{Type-2}$, can be done.

As shown in Fig. 6 (f) (respectively, (g)), it is easy to check that $x-SH_2 \in \text{Type-3}$ (re-

spectively, Type-4) and $y-SH_2 \in \text{Type-2}$ do the exchange operation in four steps.

Proposition 5.4: With at most four steps, one exchange operation for prefix computation between two adjacent SH_2 's, $x-SH_2 \in \text{Type-3}$ or Type-4 and $y-SH_2 \in \text{Type-2}$, can be done.

Specifically, as shown in Fig. 6 (g), when node 00 is TF in $y-SH_2$, $x-SH_2$ and $y-SH_2$ cannot communicate with each other. Thus, we have the following fact. Here, we will not consider this situation because two such SH_2 's will temporarily do nothing.

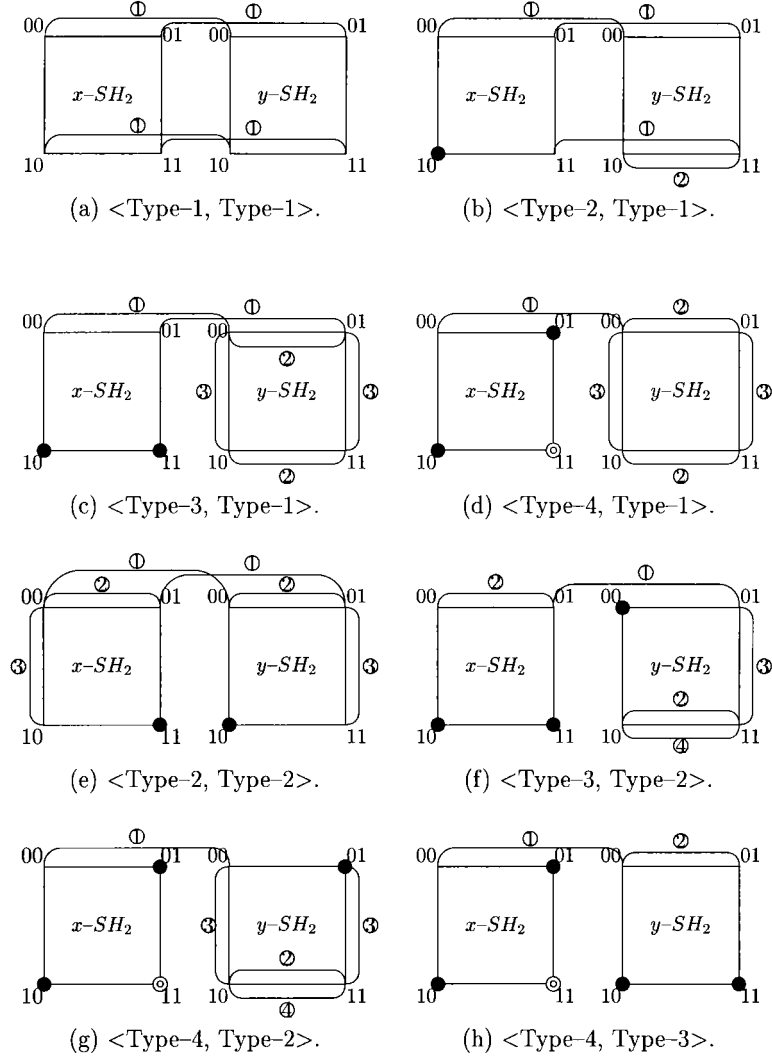


Fig. 6. The eight communication patterns between two SH_2 's for prefix computation.

Fact 5.1: Two adjacent SH_2 's, $x-SH_2 \in \text{Type-4}$ and $y-SH_2 \in \text{Type-2}$, may not communicate with each other.

As shown in Fig. 6 (h), by the same arguments, it is easy to check that the following proposition is correct.

Proposition 5.5: In at most two steps, one exchange operation for prefix computation between two adjacent SH_2 's, $x-SH_2 \in \text{Type-4}$ and $y-SH_2 \in \text{Type-3}$, can be done.

5.1.3 Performing the remaining prefix computation of Phase_1

After analyzing the communication steps required in one exchange operation between two SH_2 's for prefix computation, all the SH_2 's, each SH_2 being viewed as a super-node, can work together to perform the remaining prefix computation of Phase_1 on $2^{n-\log n-1}$ $\overline{SH}_{\log n-1}$'s in parallel.

From Propositions 5.2, 5.3, 5.4, and 5.5, it is known that the required steps in an exchange operation for each of the two pairs $\langle \text{Type-3}, \text{Type-2} \rangle$ and $\langle \text{Type-4}, \text{Type-2} \rangle$ is 4 while the steps required for the other pairs is at most 3. It is clear that if the two pairs $\langle \text{Type-3}, \text{Type-2} \rangle$ and $\langle \text{Type-4}, \text{Type-2} \rangle$ are used so heavy in the remaining prefix computation, it will make the total steps required in Phase_1 inefficient.

Our delay-update strategy is sketched as follows. While a large number of exchange operations among communication pairs are being performed, the concerning exchange operations for the two pairs $\langle \text{Type-3}, \text{Type-2} \rangle$ and $\langle \text{Type-4}, \text{Type-2} \rangle$ will be delayed in order to speedup the time required. Finally, a recovery strategy will be used to obtain the desired results.

By Lemma 4.1, all 2^{n-2} SH_2 's in \overline{H}_{n-2} can be relabeled such that $\langle p-SH_2$ and $w-SH_2 \rangle$ are relabeled as either $\langle 0-SH_2$ and $1-SH_2$ in $0-\overline{SH}_{\log n-1}$ or $\langle 0-SH_2$ in $0-\overline{SH}_{\log n-1}$ and $w''-SH_2$ in $j-\overline{SH}_{\log n-1}$ for $w'' \geq n/2$ and $j \neq 0$, respectively. We will first focus on the prefix computation for the first relabeling case via a brief example. Because only $0-\overline{SH}_{\log n-1}$ contains the two pairs $\langle \text{Type-3}, \text{Type-2} \rangle$ and $\langle \text{Type-4}, \text{Type-2} \rangle$, the time required in $0-\overline{SH}_{\log n-1}$ dominates the time complexity when compared to the other $\overline{SH}_{\log n-1}$.

For simplicity, letting $0-\overline{SH}_{\log n-1} = 0-\overline{SH}_4$, Fig. 7 illustrates the simulation of the remaining prefix computation of Phase_1 on $0-\overline{SH}_4$ for the first relabeling case. This simulation consists of four stages for performing exchange operations and two stages for performing update operations, which will be defined later. In Fig. 7, each square denotes an SH_2 . The black (respectively, double) circle in each SH_2 (respectively, $0-SH_2$) denotes the TF (respectively, disabled FF) node. The square associated with the symbol 'D' denotes a delayed SH_2 while the other squares denote the active SH_2 's. The delayed SH_2 will temporarily do nothing until the update stages begin. During the first four stages, the active SH_2 's will continually perform the related exchange operations. Two adjacent SH_2 's associated with two cross arrows denote that the two adjacent SH_2 's perform the exchange operation. For example, $11-SH_2$ with its adjacent $10-SH_2$, $9-SH_2$, $15-SH_2$, and $3-SH_2$ perform the exchange operations in Stage_1, Stage_2, Stage_3, and Stage_4, respectively. Two adjacent SH_2 's associated with one arrow denote that the two adjacent SH_2 's perform the update operation. For example, $11-SH_2$ with its adjacent $9-SH_2$ and $10-SH_2$ perform the update operations in Stage_5 and Stage_6 (i.e., the first

and second update stages), respectively.

In Fig. 7, Stage₁, $2x$ - SH_2 and $(2x+1)$ - SH_2 for $0 \leq x \leq 7$ perform the exchange operations. Each FF node in $2x$ - SH_2 and $(2x+1)$ - SH_2 obtains its correct prefix sum and the new local sum $\sum_{i=8x}^{8x+7} a_i$ of 2^3 operands a_i for $8x \leq i \leq 8x+7$. At the end of Stage₁, by Fact 5.1, 0 - SH_2 is set to be delayed. Let each FF node v in $2x$ - SH_2 and $(2x+1)$ - SH_2 have the variable $v.ps_1$ to store its temporal prefix sum, which will be used in the second update stage.

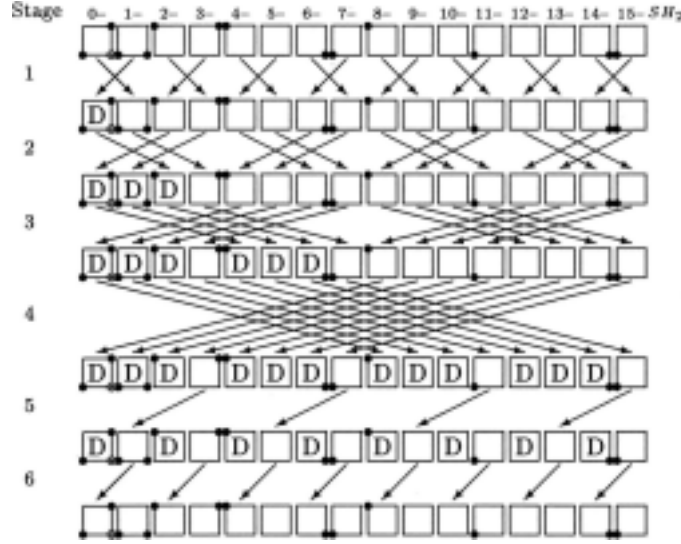


Fig. 7. The $0-\overline{SH}_4$ simulation of Phase₁ for the first relabeling case.

In Stage₂, $4x$ - SH_2 (respectively, $(4x+1)$ - SH_2) and $(4x+2)$ - SH_2 (respectively, $(4x+3)$ - SH_2) for $0 \leq x \leq 3$ perform the exchange operations. At the end of Stage₂, 1 - SH_2 is also set to be delayed. In addition, 2 - SH_2 should be set to be delayed because it cannot get the correct data from the delayed 0 - SH_2 . As a result, in the remaining prefix computation in this phase:

if any active SH_2 wants to perform the exchange operation with a delayed SH_2 , then that active SH_2 will be set to be delayed until it encounters the update stage, which will be defined later.

Except for the FF nodes in 0 - SH_2 and 2 - SH_2 , each FF node v in $4x$ - SH_2 , $(4x+1)$ - SH_2 , $(4x+2)$ - SH_2 , and $(4x+3)$ - SH_2 obtains its correct prefix sum stored in $v.ps_2$, which will be used in the first update stage, and the new local sum of 2^4 operands a_i for $16x \leq i \leq 16x+15$.

Let 'A' denote an active SH_2 . At the end of Stage₂, the active-delayed propagation pattern (ADPP) in $0-\overline{SH}_4$ is $(D^3A)^{2^0}(A^4)^{2^2-2^0}$ ($= DDDAAAAAAAAAAAAAAAA$) with respect to the status of these 16 SH_2 's from 0 - SH_2 to 15 - SH_2 . After Stage₃ and Stage₄ are completed as shown in Fig. 7, the corresponding ADPP's are $(D^3A)^{2^1}(A^4)^{2^2-2^1}$ and $(D^3A)^{2^2}(A^4)^{2^2-2^2}$ ($= (D^3A)^{2^2}$), respectively. In general, using the induction proving tech-

nique, it can be proven that the ADPP at the end of Stage $_i$, $2 \leq i \leq \log n - 1$, is $(D^3A)^{2^{i-2}}(A^4)^{2^{\log n-3} \cdot 2^{i-2}}$. Immediately, we have the following proposition.

Proposition 5.6: The ADPP for the first relabeling case in $0-\overline{SH}_{\log n-1}$ is $(D^3A)^{2^{\log n-3}}(=(D^3A)^{2^{\log n-3}}(A^4)^{2^{\log n-3} \cdot 2^{\log n-3}})$ at the end of Stage $_{(\log n - 1)}$.

Before returning to the example shown in Fig. 7, we will first define the update operation. Let each FF node have the variables tps_1 , tps_2 , and $tps_{(\log n - 1)}$ which can be used to store its temporal prefix sums obtained at the end of Stage $_1$, Stage $_2$, and Stage $_{(\log n - 1)}$, respectively, where the final prefix sum of Phase $_1$ is stored in $tps_{(\log n - 1)}$. From Proposition 5.6, each FF node v in the delayed $(4x+1)-SH_2$ (respectively, $2x-SH_2$) for $0 \leq x \leq 2^{\log n-3} - 1$ (respectively, $0 \leq x \leq 2^{\log n-2} - 1$) in $0-\overline{SH}_{\log n-1}$ has the correct prefix sum stored in $v.tps_2$ (respectively, $v.tps_1$) of these operands a_i for $16x \leq i \leq 16x+15$ (respectively, $8x \leq i \leq 8x+7$). Therefore, the FF node v in $(4x+1)-SH_2$ (respectively, $2x-SH_2$) can obtain its correct prefix sum $v.tps_{(\log n - 1)}$ by computing $v.tps_{(\log n - 1)} = v.tps_2 + \sum_{i=0}^{16x-1} a_i$ (respectively, $v.tps_{(\log n - 1)} = v.tps_1 + \sum_{i=0}^{8x-1} a_i$), where the value $\sum_{i=0}^{16x-1} a_i$ (respectively, $\sum_{i=0}^{8x-1} a_i$) is called the local-update value. Fortunately, if each $(4x+3)-SH_2$ (respectively, $(2x+1)-SH_2$) is active, the local-update value $\sum_{i=0}^{16x-1} a_i$ (respectively, $\sum_{i=0}^{8x-1} a_i$) can be obtained by each FF node v' in the active $(4x+3)-SH_2$ (respectively, $(2x+1)-SH_2$) simultaneously by computing $v'.tps_{(\log n - 1)} - v'.tps_2$ (respectively, $v'.tps_{(\log n - 1)} - v'.tps_1$). Thus, we have the following definition.

Definition 5.2: The first (respectively, second) update operation consists of the following operations: (1) the active $(4x+3)-SH_2$ (respectively, $(2x+1)-SH_2$) for $0 \leq x \leq 2^{\log n-3} - 1$ (respectively, $0 \leq x \leq 2^{\log n-2} - 1$) first computes the local-update value $\sum_{i=0}^{16x-1} a_i$ (respectively, $\sum_{i=0}^{8x-1} a_i$); (2) the active $(4x+3)-SH_2$ (respectively, $(2x+1)-SH_2$) sends the local-update value and the kept local sum $\sum_{i=0}^{2^{\log n+1}} a_i$ to the delayed $(4x+1)-SH_2$ (respectively, $2x-SH_2$); (3) the delayed $(4x+1)-SH_2$ (respectively, $2x-SH_2$) receives the local-update value and the local sum from the active $(4x+3)-SH_2$ (respectively, $(2x+1)-SH_2$); (4) the delayed $(4x+1)-SH_2$ (respectively, $2x-SH_2$) obtains its correct prefix sums via each FF node v by computing $v.tps_{(\log n - 1)} = v.tps_2 + \sum_{i=0}^{16x-1} a_i$ (respectively, $v.tps_{(\log n - 1)} = v.tps_1 + \sum_{i=0}^{8x-1} a_i$).

As shown in Fig. 7, the active $11-SH_2$ is used to update the delayed $9-SH_2$. First, each FF node v' in $11-SH_2$ computes the local-update value $\sum_{i=0}^{31} a_i = v'.tps_4 - v'.tps_2$. Then, $11-SH_2$ sends the local-update value $\sum_{i=0}^{31} a_i$ and its own local sum $\sum_{i=0}^{63} a_i$ to $9-SH_2$. Thus, each FF node v in $9-SH_2$ can obtain the correct prefix sum $v.tps_4$ by computing $v.tps_4 = v.tps_2 + \sum_{i=0}^{31} a_i$ and keeps the local sum $\sum_{i=0}^{63} a_i$ sent from $11-SH_2$. After the first update stage is performed, the delayed $9-SH_2$ becomes active, and generally, the ADPP at the end of Stage $_5$ becomes $(DA)^{2^3}$. Now, each active $(2x+1)-SH_2$ for $0 \leq x \leq 7$ has the correct prefix sums and local sum of Phase $_1$.

In the second update stage, i.e., Stage $_6$, the active $11-SH_2$ (respectively, $9-SH_2$) is used to update the delayed $10-SH_2$ (respectively, $8-SH_2$). Each FF node v' in $11-SH_2$ (respectively, $9-SH_2$) first computes the local-update value $\sum_{i=0}^{39} a_i = v'.tps_4 - v'.tps_1$

(respectively, $\sum_{i=0}^{31} a_i = v'.tps_4 - v'.tps_1$). Then, 11- SH_2 (respectively, 9- SH_2) sends the local-update value $\sum_{i=0}^{39} a_i$ (respectively, $\sum_{i=0}^{31} a_i$) and the local sum $\sum_{i=0}^{63} a_i$ to 10- SH_2 (respectively, 8- SH_2). Each FF node v in 10- SH_2 (respectively, 8- SH_2) can obtain its correct prefix sum $v.tps_4$ by computing $v.tps_4 = v.tps_1 + \sum_{i=0}^{39} a_i$ (respectively, $v.tps_4 = v.tps_1 + \sum_{i=0}^{31} a_i$) and keeps the local sum $\sum_{i=0}^{63} a_i$.

Based on the description of the two update stages, in general, we have the following two propositions.

Proposition 5.7: Using the active $(4x+3) - SH_2$ for $0 \leq x \leq 2^{\log n - 3} - 1$, the first update stage can make the resulting prefix sums and local sum of the delayed $(4x+1) - SH_2$ correct in $0 - \overline{SH}_{\log n - 1}$. Then, the corresponding ADPP becomes $(DA)^{2^{\log n - 2}}$.

Proposition 5.8: Using the active $(2x+1) - SH_2$ for $0 \leq x \leq 2^{\log n - 2} - 1$, the second update stage can make the resulting prefix sums and local sum of the delayed $2x - SH_2$ correct in $0 - \overline{SH}_{\log n - 1}$. Then, each FF node obtains the correct prefix sum and local sum of $0 - \overline{SH}_{\log n - 1}$. In addition, the corresponding ADPP becomes $(A)^{2^{\log n - 1}}$.

Totally, it takes $\log n - 1$ stages to perform exchange operations and two stages to perform update operations in the remaining prefix computation of Phase_1 in each $\overline{SH}_{\log n - 1}$ for the first relabeling case.

According to the above description, the formal algorithm for Phase_1 is listed below.

ALGORITHM PFRC /*Phase_1 for the First Relabeling Case*/

- Stage 0.** Each SH_2 in \overline{H}_{n-2} computes its own prefix sums and local sum as mentioned in Subsection 5.1.1. From Proposition 5.1, Stage_0 takes at most three steps.
- Stage 1.** Let each $b - SH_2$ in \overline{H}_{n-2} , $b = b_n b_{n-1} \dots b_{i+1} b_i b_{i-1} \dots b_4 b_3$ for b_i and $3 \leq i \leq n$, be adjacent to $b^{(i)} - SH_2$, $b^{(i)} = b_n b_{n-1} \dots b_{i+1} \bar{b}_i b_{i-1} \dots b_4 b_3$. Each $b - SH_2$ and $b^{(i)} - SH_2$ do the exchange operation as defined in Subsection 5.1.2. By Fact 5.1, let 0- SH_2 be set to be delayed. From Propositions 5.2, 5.3, and 5.5, Stage_1 takes at most three steps.
- Stage 2.** Each $b - SH_2$ and $b^{(4)} - SH_2$ perform the exchange operation. Afterwards, both 2- SH_2 and 1- SH_2 are set to be delayed. From Propositions 5.2, 5.3, and 5.4, Stage_2 takes at most four steps.
- Stage t for $3 \leq t \leq \log n - 1$.** Each $b - SH_2$ and $b^{(t+2)} - SH_2$ perform the exchange operation. If $b - SH_2$ is delayed, then $b^{(t+2)} - SH_2$ is set to be delayed. From Propositions 5.2 and 5.3, Stage_2 takes at most three steps.
- Stage ($\log n$):** The first update stage. From Definition 5.2 and Proposition 5.7, $b' - SH_2$, where $b' = b_n b_{n-1} \dots b_6 b_5 1 1$, and $b^{(4)} - SH_2$ perform the first update stage. From Propositions 5.2 and 5.3, except for 3- SH_2 updating 1- SH_2 , this update stage takes at most three steps. Checking Fig. 6 (f), it is easy to see that 3- SH_2 can update 1- SH_2 in two steps. Thus, this update stage takes three steps.
- Stage ($\log n + 1$):** The second update stage. From Proposition 5.8, each $b'' - SH_2$, $b'' = b_n b_{n-1} \dots b_6 b_5 b_4 1$, and $b^{(3)} - SH_2$ perform the second update stage. From Propositions 5.2, 5.3 and 5.5, this update stage takes at most three steps.

End.

Lemma 5.1: For the first relabeling case, Phase 1 takes $3\log n + 7$ ($= 3+3+4+3(\log n - 3)+3+3$) steps.

While each SH_2 is thought of as a supernode, any of the other $\overline{SH}_{\log n - 1}$ except for $0-\overline{SH}_{\log n - 1}$ has the same communication patterns as does the one without the fault-tolerant capability [5] for computing prefix sums. Thus, we have the following lemma.

Lemma 5.2: After Stage_($\log n - 1$) in Algorithm PFRC is finished, except for $0-\overline{SH}_{\log n - 1}$, any of the other $\overline{SH}_{\log n - 1}$ can correctly compute its own prefix sums and local sum of Phase₁.

From Proposition 5.7 and 5.8, we have the following lemma.

Lemma 5.3: After Algorithm PFRC is finished, $0-\overline{SH}_{\log n - 1}$ can correctly compute its corresponding prefix sums and local sum of Phase₁.

Now, we will discuss the prefix computation on each $\overline{SH}_{\log n - 1}$ for the second relabeling case. A brief example is given to illustrate the concept. Assuming that $w'' = 17$, Fig. 8 illustrates simulation of the remaining prefix computation of Phase₁ for $0-\overline{SH}_4$ and $1-\overline{SH}_4$ for the second relabeling ease. Like the first relabeling case, the simulations for both $0-\overline{SH}_4$ and $1-\overline{SH}_4$ are the two worst cases for all \overline{SH}_4 's.

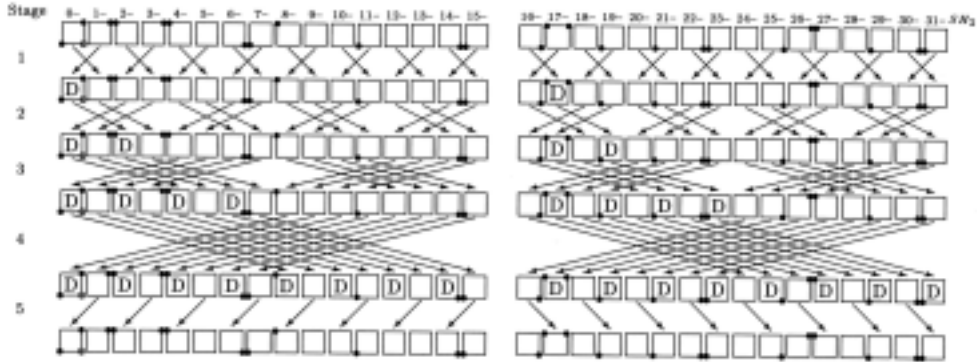


Fig. 8. The $0-\overline{SH}_4$ and $1-\overline{SH}_4$ simulations of Phase₁ for the second relabeling case.

In Stage₁, in general, each $b-SH_2$ in \overline{H}_{n-2} , $b = b_n b_{n-1} \dots b_4 b_3$ for $b_i \in \{0, 1\}$ and $3 \leq i \leq n$, and $b^{(3)}-SH_2$, $b^{(3)} = b_n b_{n-1} \dots b_4 \overline{b}_3$ perform the exchange operation. From Proposition 5.2, 5.3, 5.4, and 5.5, Stage₁ takes at most four steps. At the end of Stage₁, $0-SH_2$ and $w''-SH_2$ are set to be delayed. In Stage _{t} , $2 \leq t \leq \log n - 1$, each $b-SH_2$ and $b^{(t+2)}-SH_2$ do the exchange operation. If $b-SH_2$ is delayed, then $b^{(t+2)}-SH_2$ is set to be delayed. Thus, from Propositions 5.2 and 5.3, Stage _{t} takes at most three steps.

Proposition 5.9: For the second relabeling case, the ADPP in $0 - \overline{SH}_{\log n - 1}$ (respectively, $j - \overline{SH}_{\log n - 1}, j \neq 0$) containing an SH_2 with two disconnected (*respectively, connected*) TF nodes will become $(DA)^{2^{\log n - 2}}$ (respectively, either $(DA)^{2^{\log n - 2}}$ or $(AD)^{2^{\log n - 2}}$) at the end of Stage_1 ($\log n - 1$).

From Proposition 5.9, it is easy to see that Phase_1 for the second relabeling case only needs one update stage. The ADPP in $j - \overline{SH}_{\log n - 1}$ for this case is either $(DA)^{2^{\log n - 2}}$ or $(AD)^{2^{\log n - 2}}$. Based on the same argument as in Proposition 5.8, the SH_2 's with the ADPP with $(AD)^{2^{\log n - 2}}$ can be updated. From Proposition 5.8, the delayed SH_2 's in $0 - \overline{SH}_{\log n - 1}$ and $j - \overline{SH}_{\log n - 1}$ can be updated.

From Propositions 5.2 and 5.3, except for updating $0-SH_2$ and $w''-SH_2$, the update for the other SH_2 's takes at most three steps. Examining Figs. 6 (c), (d), (f), and (g), it is easy to see that $0-SH_2 \in \text{Type-4}$ and $w''-SH_2 \in \text{Type-3}$ can be updated in at most two steps. Consequently, it is easy to see that Phase_1 for the second relabeling case takes $3\log n + 4$ ($= 3 + 4 + 3(\log n - 2) + 3$) steps.

Lemma 5.4: Phase_1 for the second relabeling case takes $3\log n + 4$ steps.

In the last part of this subsection, we will show that when a VF node occurs, the proposed method for Phase_1 still works. When a VF node occurs in Case 3b, $0-SH_2$ consists of one VF, one FF, and two disconnected TF nodes. The only FF node in $0-SH_2$ is selected to compute the prefix sums. Thus, $0-SH_2$ with a VF node can be treated as $0-SH_2$ without a VF node. Consequently, Algorithm PFRC still works for Case 3b when one VF node occurs.

When a VF node occurs in Case 2, $0-SH_2$ consists of three TF nodes and one VF node. Thus, if any SH_2 performs the exchange operation with $0-SH_2$, then that SH_2 will be delayed. The ADPP in $0 - \overline{SH}_{\log n - 1}$ is $(DA)^{2^{\log n - 2}}$, and it is covered by the above second relabeling case. Consequently, if VF occurs, the proposed algorithm can still work. According to Lemmas 5.1 and 5.4, and the above, we have the following lemma.

Lemma 5.5: In $3\log n + 7$ (*respectively, $3\log n + 4$*) steps, Phase_1 for the first (*respectively, second*) relabeling case can compute the correct prefix sums and local sum in each $SH_{\log n + 1}$, i.e., $\overline{SH}_{\log n - 1}$, spanned by dimensions 1, 2, ..., and $\log n + 1$.

5.2 Phase_2: Global Computation Among Subcubes

After Phase_1 is finished, each $\overline{SH}_{\log n - 1}$ has computed its own prefix sums and local sum. Phase_2 performs global computation among subcubes.

In this phase, we initially construct $2n$ $SH_{n - \log n - 1}$'s, each of which is spanned by dimensions $\log n + 2, \log n + 3, \dots, n - 1$, and n . Thus, the $2^{n - \log n - 1}$ nodes in each $SH_{n - \log n - 1}$ are collected from different $2^{n - \log n - 1}$ $\overline{SH}_{\log n - 1}$'s defined and used in Phase_1. Specifically, if node x for $0 \leq x \leq 2^{n - \log n - 1} - 1$ in one $SH_{n - \log n - 1}$ is FF, then node x kept the local sum $\sum_{i=2nx}^{2n(x+1)-1} a_i$ derived at Phase_1.

Because at most $\lfloor 3n / 2 \rfloor - 1$ TF nodes and one disabled FF node are allowable in the

faulty H_n , according to the pigeonhole principle, there exist at least $\lceil n/2 \rceil$ FF $SH_{n-\log n-1}$'s. Among these $\lceil n/2 \rceil$ FF $SH_{n-\log n-1}$'s, we choose the FF $SH_{n-\log n-1}$ with the smallest label index, say $k-SH_{n-\log n-1}$ for $0 \leq k \leq 2n-1$, to perform the parallel algorithm without the fault-tolerant capability for prefix computation [5]. Each node in $k-SH_{n-\log n-1}$ obtains the correct prefix sums among these $2^{n-\log n-1}$ local sums, i.e., $\sum_{i=2nx}^{2n(x+1)-1} a_i$ for $0 \leq x \leq 2^{n-\log n-1} - 1$. At the end of Phase_2, node x in $k-SH_{n-\log n-1}$ keeps the corresponding prefix sum $\sum_{i=0}^{2n(x+1)-1} a_i$, which will be used in Phase_3 to update the final prefix sums.

Lemma 5.6: Phase_2 takes $n - \log n - 1$ steps.

5.3 Phase_3: The Local Update in Each Subcube

The same as in Phase_1, in Phase_3, H_n is partitioned into $2^{n-\log n-1}$ $SH_{\log n+1}$'s ($=\overline{SH}_{\log n-1}$'s). Recall that at the end of Phase_1, each $x-\overline{SH}_{\log n-1}$, $0 \leq x \leq 2^{n-\log n-1} - 1$, has computed its own prefix sums and local sum of these operands a_i for $2nx \leq i \leq 2n(x+1) - 1$. Each FF node in each $x-\overline{SH}_{\log n-1}$ can obtain the correct finally prefix sum by adding the update-value $\sum_{i=0}^{2nx-1} a_i$ to its own prefix sum(s). The local update for final prefix sums is performed in each subcube. In what follows, we will present this phase in detail.

Each $x-\overline{SH}_{\log n-1}$ contains node k , which also belongs to $k-SH_{n-\log n-1}$ in $\overline{H}_{\log n+1}$ defined and used in Phase_2. Node k holding $\sum_{i=0}^{2n(x+1)-1} a_i$ (respectively, $\sum_{i=2nx}^{2n(x+1)-1} a_i$) obtained in Phase_2 (respectively, Phase_1) in each $x-SH_{\log n+1}$ computes the update-value $\sum_{i=0}^{2nx-1} a_i = \sum_{i=0}^{2n(x+1)-1} a_i - \sum_{i=2nx}^{2n(x+1)-1} a_i$. Then, node k in each $x-\overline{SH}_{\log n-1}$ broadcasts the update-value $\sum_{i=0}^{2nx-1} a_i$ to the other FF nodes in $x-\overline{SH}_{\log n-1}$.

Although some fault-tolerant broadcasting methods on faulty hypercubes [1, 6] have been presented, they cannot be used in Phase_3 to broadcast the update-value in each $x-\overline{SH}_{\log n-1}$ to the other processors for the following two reasons: (1) the broadcasting methods [1, 6] cannot guarantee that they will be able to simultaneously work well in $2^{n-\log n-1}$ $\overline{SH}_{\log n-1}$ constructed from an SIMD H_n ; (2) too many TF nodes may fall on a specific $\overline{SH}_{\log n-1}$ such that the number of TF nodes in the subcube is larger than that allowable in the previous broadcasting methods [1, 6]. Consequently, instead of using the broadcasting methods [1, 6], we modify the method Phase_1 slightly and use it to perform the broadcasting process.

Let each FF node in each $x-\overline{SH}_{\log n-1}$ have the operand 0, while node k takes the update-value $\sum_{i=0}^{2nx-1} a_i$ as the operand. Removing the function of computing prefix sums and preserving the function of computing the local sum in Phase_1, each $x-\overline{SH}_{\log n-1}$ can compute the local sum, i.e., the update-value. Therefore, the broadcasting process is finished.

Finally, each FF node in $x-\overline{SH}_{\log n-1}$ obtains the correct prefix sum by adding the update-value to its own prefix sum obtained in Phase_1.

Next, the number of steps required in Phase_3 will be analyzed. We will first consider the first relabeling case. Because $0-\overline{SH}_{\log n-1}$ does not need to update its prefix sums, the two update stages can be omitted. From Propositions 5.2 and 5.3, each stage $j-\overline{SH}_{\log n-1}$, $j \neq 0$, takes at most three steps. Thus, Phase_3 for the first case takes $3\log n+1$ ($= 3+3+4+3(\log n - 3)$) steps. In the second relabeling case, it is easy to see

that it takes the same number of steps as in Phase_1.

Lemma 5.7: In $3\log n+1$ (respectively, $3\log n+4$) steps, Phase_3 for the first (respectively, second) relabeling case can update the final prefix sums in H_n .

From Lemmas 5.5, 5.6, and 5.7, we have the main result.

Theorem 5.8: In $n+5\log n+7$ steps, $n \geq 2$, a prefix computation with 2^n operands can be performed in H_n with $\lfloor 3n/2 \rfloor - 1$ TF nodes.

6. CONCLUSIONS

Based on 2^n operands and $n+5\log n+7$ steps as well as some new partitioning schemes and a delay-update technique, this paper have presented a fault-tolerant algorithm for prefix computation on H_n with $\lfloor 3n/2 \rfloor - 1$ TF nodes. Our main contribution is that although 11 extra steps are required, the proposed algorithm for prefix computation tolerates $\lfloor n/2 \rfloor$ more faulty nodes than does the algorithm in [4].

Prefix computation is the kernel of many applications, such as finding solutions tridiagonal systems and B-spline surface fitting. However, the I/O problem in the proposed fault-tolerant algorithm for prefix computation is still a bottleneck from the practical viewpoint. How to alleviate the I/O problem in the proposed algorithm is a future research topic.

REFERENCES

1. P. Fraigniaud, "Asymptotically optimal broadcasting and gossiping in faulty hypercubes," *IEEE Transactions on Computers*, Vol. 41, No. 11, 1992, pp. 1410-1419.
2. J. Håstad, T. Leighton, and M. Newman, "Reconfiguring an hypercube in the presence of faults," *ACM Symposium on Theory of Computing* 1987, pp. 274-284.
3. S. Lakshmivarahan and S. K. Dhall, *Parallel Computing Using The Prefix Problem*, Oxford University Press, 1994.
4. C. S. Raghavendra and M. A. Sridhar, "Prefix computation on a faulty hypercube," 1993 *International Conference on Parallel Processing*, 1993, pp. III 280-283.
5. S. Ranka and S. Sahni, *Hypercube Algorithms with Applications to Image Processing and Pattern Recognition*, Chap. 2, Springer-Verlag, NY, 1990.
6. A. Sengupta and C. S. Raghavendra, "All-to-all broadcast and matrix multiplication in faulty SIMD hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 6, 1998, pp. 550-559.
7. P. J. Yang and C. S. Raghavendra, "Reconfiguration of binary trees in faulty hypercubes," 7th *International Parallel Processing Symposium*, 1993, pp. 401-405.
8. P. J. Yang and C. S. Raghavendra, "Embedding and reconfiguration of binary trees in faulty hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 3, 1996, pp. 237-245.
9. P. J. Yang, S. B. Tien, and C. S. Raghavendra, "Embedding of rings and chains onto

faulty hypercubes,” Technical Report, Department of Electrical Engineering, University of South California., 1990.

10. P. J. Yang, S. B. Tien, and C. S. Raghavendra, “Embedding of multidimensional meshes on to faulty hypercubes,” 1991 *International Conference on Parallel Processing*, 1991, pp. 571-574.



Yu-Wei Chen (陳育威) received the B.S. and Ph.D. degree from the Department of Information Management, National Taiwan University of Science and Technology, R.O.C., in 1993 and 1999, respectively. In 1994, he began work on a Ph.D. degree after taking first year courses for the M.S. degree. He is now an assistant professor at Aletheia University. His research interests include parallel and distributed computing, fault-tolerant computing, and networks.



Kuo-Liang Chung (鍾國亮) received the B.S., M.S., and Ph.D. degrees in Computer Science and Information engineering from National Taiwan University, R.O.C. He is now a professor in the Department of Information Management and the Institute of Information Engineering, National Taiwan University of Science and Technology. His current research interests include image processing, compression, computer graphics, video processing, and theoretical computer science. He is a member of IEEE.