A Boolean Expression-Based Approach for Maximum Incomplete Subcube Identification in Faulty Hypercubes

Hsing-Lung Chen, *Member*, *IEEE*, and Nian-Feng Tzeng, *Senior Member*, *IEEE*

Abstract—An incomplete hypercube possesses virtually every advantage of complete hypercubes, including simple deadlock-free routing, a small diameter, bounded link traffic density, a good support of parallel algorithms, and so on. It is natural to reconfigure a faulty hypercube into a maximum incomplete cube so as to lower potential performance degradation, because a hypercube so reconfigured often results in a much larger system than what is attainable according to any conventional reconfiguration scheme which identifies only complete subcubes. A maximum incomplete subcube involves one maximum complete subcube, plus certain smaller complete subcubes, and, thus, may accommodate multiple jobs of different sizes simultaneously, delivering a higher performance level. This paper proposes an efficient approach for identifying all the maximum incomplete subcubes present in a faulty hypercube. The proposed approach is on the basis of manipulating Boolean expressions, with the search space reduced considerably by taking advantage of the basic properties of faulty hypercubes during expression manipulation. It is distributed, in that every healthy node executes the same identification algorithm independently, at the same time. It is confirmed by fault simulation that our approach indeed gives rise to significantly larger reconfigured systems and requires short execution times.

Index Terms—Boolean expressions, distributed algorithms, faulty hypercubes, incomplete subcubes, reconfiguration.

1 INTRODUCTION

THE hypercube has received considerable attention, and many hypercube-based machines have been available [1], [2], [3], [4], [5]. For a large hypercube system, the probability of faults arising is nonnegligible, and, over its mission duration, the system might involve one or several faults, making it necessary to consider the fault-tolerant issue in system design. It is possible for the hypercube to achieve fault-tolerance without employing spares by reconfiguring itself to a smaller sized system after faults occur, operating in a gracefully degraded mode. Our focus in this paper lies in such a fault-tolerant approach.

Several schemes have been proposed for reconfiguring a hypercube with faults [6], [7], [8], [9], [10]. All these schemes assume that only *complete* subcubes are permitted and attempt to find the complete healthy subcubes with the maximum dimension in a faulty hypercube. They are often unable to maintain as many workable nodes as desired, after reconfiguration, due to the strong restriction on allowable system sizes. For example, a reconfigured hypercube in the presence of just one fault contains merely half of its original nodes, discarding many healthy nodes simply to meet the system size constraint, and possibly resulting in a severe performance loss (as system performance tends to be in proportion to the number of nodes).

This paper deals with an approach for identifying maximum *incomplete* subcubes in a faulty hypercube, retaining as many healthy nodes as possible in order to keep performance degradation minimal. Unlike a complete one, an incomplete cube can be of any arbitrary size. A maximum incomplete subcube usually involves one maximum complete subcube, plus certain smaller sized subcubes, making it possible to finish a given batch of jobs faster than its complete counterpart alone, by supporting simultaneous execution of multiple jobs of different sizes. Moreover, an incomplete subcube may carry out a given job faster by assigning more nodes to execute the job cooperatively, in view of the fact that the communication time for sending a message to a remote node is only slightly larger than that to a neighboring node on a contemporary hypercube, which supports circuit switching or wormhole routing. This is because the communication time in such a hypercube is distance-insensitive, if no conflicts arise [18].

Simple and deadlock-free algorithms for routing and for broadcasting messages in an incomplete cube have been introduced [11]. These algorithms are essentially similar to those for complete cubes [13], and, in fact, a unified set of algorithms can be used for delivering messages in both complete and incomplete cubes alike. A recent study on incomplete cube systems revealed that no heavily loaded point existed in such a system [12], [20], suggesting that a faulty hypercube can be reconfigured into an incomplete cube without creating any congestion point in it. The maximum incomplete subcubes available in a faulty hypercube are found, through extensive fault simulation, to be

- *H.-L. Chen is with the Department of Electronic Engineering, National Taiwan University of Science and Technology, 43 Keelung Rd., Sec. 4, Taipei, Taiwan, R.O.C. E-mail: hlchen@et.ntust.edu.tw.*
- *N.-F. Tzeng is with The Center for Advanced Computer Studies, University of Southwestern Louisiana, P.O. Box 44330, Lafayette, LA 70504-4330. E-mail: tzeng@cacs.usl.edu.*

Manuscript received 5 Jan. 1996; revised 18 Aug. 1997.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 100082.

much larger than the maximum complete subcubes, as a result of high flexibility in allowable system sizes. In fact, for any random fault pattern in a hypercube, a maximum incomplete subcube found in the hypercube has an average size roughly two times as large as that of a maximum complete subcube [19].

Our proposed identification approach uncovers all maximum incomplete subcubes by taking advantage of the basic properties of faulty hypercubes to reduce search space considerably. For a given node and a set of faults in a faulty hypercube, one can quickly derive all reject regions, which consist of those nodes *impossible* to be a part of any fault-free subcube containing the given node, and then arrives at a Boolean expression that specifies the collection of all healthy complete subcubes containing the given node, efficiently and systematically. Next, the Boolean expression so obtained is carefully manipulated to explore all the incomplete subcubes composed of these healthy complete subcubes. Every fault-free node is involved in the identification process concurrently and independently. In order to facilitate reconfiguration and avoid unnecessary traffic, the addresses of all maximum incomplete subcubes, after their size is determined in a distributed manner, are sent nonredundantly to the host by nodes elected via a distributed procedure. The average time taken for each involved node to carry out our identification algorithm is short, according to the simulation results.

It should be noted that our reconfigured system may drop some healthy nodes so as to preserve the incomplete hypercube topology, to be defined in Section 2. If a system attempts to maintain all the healthy nodes without assuring the incomplete hypercube topology, it is impossible to have simple deterministic routing and broadcasting algorithms. In fact, for a hypercube with all the healthy nodes retained after faults occur without reconfiguration, node-to-node routing and broadcasting are often complicated and inefficient, realized on the basis of “sidetracking” [15], “backtracking” [16], or duplicated transmission [17], which do not guarantee the important property of deadlock-freeness. Moreover, traffic density over a link under this situation is not bounded by a small constant, and a message is not necessarily routed through a shortest path (as in an incomplete hypercube), possibly causing excessive communication delay.

Various experiments have been carried out on the *i*PSC/860 to study allocating tasks into incomplete hypercubes and to contrast the performance difference on an incomplete hypercube and on a complete counterpart. Among the algorithms considered, matrix multiplication requires virtually the same allocation on an incomplete hypercube as on a complete hypercube, using an identical node program; whereas *FFT* involves slightly different allocation on an incomplete hypercube than on a complete hypercube. Our experimental results [19] demonstrate that for matrix multiplication, an incomplete system of size 24 may outperform a complete system of size 16 by more than 45 percent; this is somewhat expected as matrix multiplication is computation-intensive and the performance level is proportional to the system size. In the other extreme, such as *FFT* algorithms, in which communication is intensive, the incomplete system still can achieve a performance gain of, typically, six to 12 percent.

An algorithm for maximum incomplete subcube determination in a faulty hypercube has been discussed recently in [22]. The earlier algorithm may determine only *some* maximum incomplete subcubes present in a faulty hypercube and is a centralized scheme, with the algorithm executed by one single processor, say the host. The approach proposed here, in contrast, identifies *all* maximum incomplete subcubes in existence and is a distributed method, with every fault-free node involved in the identification process. The rest of this article is organized as follows: Section 2 provides notation and background relevant to subsequent presentation. The methodology of deriving the expression specifying subcubes which contain a given node is described in Section 3. Pertinent features that help arrive at the general expression for incomplete subcubes are presented in Section 4. Section 5 introduces the process of finding incomplete subcubes involving a given node, through Boolean expression manipulation. Distributed identification of all maximum incomplete subcubes in a faulty hypercube is provided in Section 6. The results of our reconfiguration approach are given in Section 7.

2 PRELIMINARY

Let H_n denote an n-dimensional hypercube, which consists of 2^n nodes, each labeled by an n-bit string, $l_{n-1}l_{n-1}\cdots l_1 l_0$, where bit l_i corresponds to dimension i. A link joins two nodes whose labels differ in exactly one bit position. A k-dimensional subcube in H_n can be addressed uniquely by a string of n symbols over set {0, 1, ∗}, where ∗ is a *don’t care* symbol, such that there are exactly k ∗'s in the string. For example, a two-dimensional subcube involving nodes 00011, 00111, 01011, and 01111 in H_5 is addressed by 0∗∗11, or, equivalently, $0{*}^2\,1^2$, where α^c denotes c consecutive α's.

An n-dimensional incomplete hypercube with M nodes, $2^{n-1} \le M < 2^n$, represented by I_n^M, is defined recursively as follows: I_n^M comprises two components, H_{n-1} and $I_k^{M-2^{n-1}}$ $(k = \lceil \log_2(M - 2^{n-1}) \rceil)$, with nodes in H_{n-1} numbered from 0 to $2^{n-1}-1$ and nodes in $I_k^{M-2^{n-1}}$ numbered from 2^{n-1} to M–1; a link exists between a node A in H_{n-1} and another node B in $I_k^{M-2^{n-1}}$, if and only if the addresses of nodes A and B differ in exactly one bit. I_n^M comprises a set of complete cubes of dimensions $n-1$ and below, and no two constituent cubes are of the same size. Incomplete hypercube I_5^{26}, depicted in Fig. 1, for example, comprises H_4 and I_4^{10}, which, in turn, contains H_3 and $I_2^2 = H_1$. Let the binary representation of M be $< 1x_{n-2}x_{n-3}\cdots x_i \cdots x_1 x_0 >$, then it is clear that I_n^M contains, in addition to H_{n-1}, H_i’s, for all $x_i = 1,\ 0 \le i \le n-2$. H_i is a constituent cube of I_n^M if and only if bit x_i in the binary representation of M equals 1.

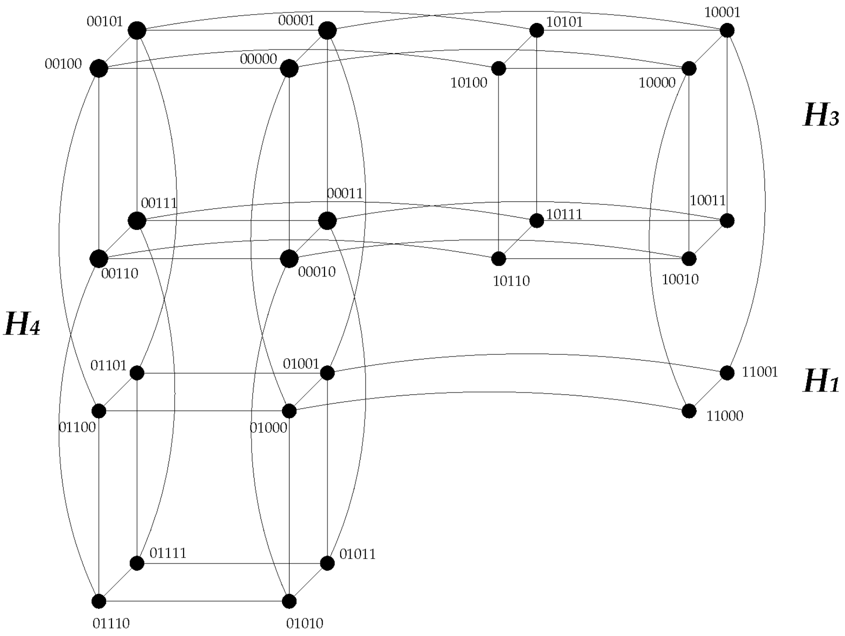

Fig. 1. An incomplete hypercube with 26 nodes, I_5^{26}.

Since there is only one binary representation for any given number M, the set of constituent cubes in an incomplete hypercube with size M is uniquely defined.

In a faulty complete hypercube, one or multiple fault-free incomplete subcubes exist. Each such fault-free incomplete subcube is contained in exactly one smallest complete subcube, called the *minimum cover subcube*. For fault-free incomplete subcube I_s^{Υ}, $\Upsilon > 2^{s-1}$, its minimum cover subcube is of dimension s, i.e., H_s, and nodes which are in the minimum cover subcube, but are outside I_s^{Υ}, form a subcube (either a complete or an incomplete one), called the *conjugate subcube*. I_s^{Υ} and its conjugate subcube together form a complete subcube of dimension s, and the conjugate subcube involves at least one fault (as otherwise, a fault-free H_s exists). Every fault-free incomplete subcube I_s^{Υ}, $\Upsilon > 2^{s-1}$, has one and only one conjugate subcube, denoted by $CS^{2^s-\Upsilon}$. For example, the conjugate subcube of I_5^{26} shown in Fig. 1 is CS^6, a three-dimensional, incomplete subcube comprising H_2 and H_1.

The operations on subcubes in a hypercube can be performed elegantly, following a way similar to Boolean algebra, if a subcube is represented as a *minterm*, i.e., product of Boolean variables, obtained from the address of the subcube by replacing bit position i with b_i (or $\overline{b_i}$), if position i is 1 (or 0), and then dropping all $*$'s. For example, subcube $*0{*}{*}1$ is represented by $\overline{b}_3 b_0$. The union of the three subcubes: $*0{*}{*}1$, $*01{*}0$, and $*00{*}0$, in H_5 is given by $\overline{b}_3 b_0 + \overline{b}_3 b_2 \overline{b}_0 + \overline{b}_3 \overline{b}_2 \overline{b}_0 = \overline{b}_3 b_0 + \overline{b}_3 \overline{b}_0 = \overline{b}_3$, which is $*0*^3$. As becomes clear later, the use of a Boolean expression to specify the union of subcubes greatly facilitates our identification procedure. Note that a *null* expression denotes the whole hypercube.

From the expression for a union of subcubes in a hypercube, one can get the addresses of all cube nodes *outside* the union of subcubes directly, with the aid of DeMorgan's theorem. Consider the union of three subcubes in H_5: $\overline{b}_4 + \overline{b}_3 + b_3 \overline{b}_2 \overline{b}_1 = W$, which, in fact, represents I_5^{26}, illustrated in Fig. 1. The collection of all the nodes outside W is expressed by $\overline{W} = (b_4)(b_3)(\overline{b}_3 + b_2 + b_1)$, which is simplified to $b_4 b_3 (b_2 + b_1)$, designating an incomplete subcube of size six (i.e., the conjugate subcube of I_5^{26}, as expected; the general expression for an incomplete subcube in a faulty hypercube will be introduced in Section 4).

3 Expression with Respect to a Given Node

Consider a faulty hypercube H_n and a given node A in H_n. It is possible to identify systematically every fault-free subcube which involves the given node A. In other words, we can arrive at the expression characterizing the set of $P = \{S_i \mid S_i$ is a fault-free subcube in H_n and S_i involves node $A\}$ quickly and systematically, by determining "regions" which never contribute to any fault-free subcube containing the given node A [10] , [21]. Each fault results in one such region, called a *reject region*, which is the smallest subcube involving both the fault and the antipodal node of A, and its address is given by performing operation $\otimes$ on the labels of the faulty node and the antipodal node, where $\otimes$ is the bit operation defined as follows: It yields 0 (or 1) if the two corresponding bits are "0" (or "1"), and it is $*$ if the two corresponding bits differ. As an example, let H_5 involve two faulty nodes 11101 and 11010, and the given node A be 01101, then the antipodal node of A is 10010 and the two reject regions are, respectively, 11101 ⊗ 10010 = $1*^4$ and 11010 ⊗ 10010 = 1*010. Any fault-free subcube involving the given node (i.e., node A) in H_5 can never contain any node inside the two reject regions (specified by the two faulty nodes 11101 and 11010) [21]. Assume that R denotes the collection of all reject regions in H_n, one corresponding to a fault. It has been shown [10], [21], by taking advantage of interesting properties of faulty hypercubes, that all the cubes nodes in H_n can be partitioned into two *disjoint sets* with respect to node A: R and P. As a result, the fact that knowing R can directly get P holds valid for any faulty hypercube, and it is fundamentally important to our identification process.

From the labels of all faulty nodes and a given node, we may quickly obtain the addresses of all reject regions (simply by performing the ⊗ operations). A reject region which lies entirely inside another reject region is then removed to avoid redundancy. Each reject region left after removing redundancy is represented by a minterm. Expression R is the summation of all minterms, or, equivalently, is given in the form of sum-of-product. From R, we immediately arrive at $P = \overline{R}$, which is in the form of product-of-sum (from DeMorgan's theorem). Since P involves all the fault-free subcubes containing the given node, we can identify the largest incomplete subcube from P, based on certain pertinent features described below.

4 Pertinent Features of Incomplete Subcubes

Consider an s-dimensional incomplete subcube with size M in H_n, I_s^M. Let $\Omega(M, j)$ be the value obtained by resetting all the bits in the binary representation of M except for the jth run of "1", with the first run of "1" starting from the most significant bit. Given M = <1 1 1 0 0 1 1 0 1>, for example, $\Omega(M, 1)$ is <1 1 1 0 0 0 0 0 0> and $\Omega(M, 2)$ equals <0 0 0 0 0 1 1 0 0>. It is clear that I_s^M consists of incomplete subcubes $I^{\Omega(M,1)}$, $I^{\Omega(M,2)}$, $I^{\Omega(M,3)}$, and so on (note that the subscripts are dropped from the subcube denotations for simplicity without confusion, since they are defined by their sizes). The next theorem reveals an interesting property of $I^{\Omega(M,1)}$, and its proof can be found in the Appendix.

Theorem 1. *Incomplete subcube $I^{\Omega(M,1)}$ of dimension s in H_n involves exactly η $(s-1)$-dimensional complete subcubes, which together constitute $I^{\Omega(M,1)}$, where η is the number of nonzero bits in the binary representation of $\Omega(M, 1)$.*

Theorem 1 indicates that incomplete subcube $I^{\Omega(M,1)}$ can be uniquely expressed by the collection of the η $(s-1)$-dimensional subcubes $y_1 y_2 \cdots y_\alpha \overline{z}_\nu$, for all $1 \le \nu \le \eta$, i.e., $y_1 y_2 \cdots y_\alpha \left(\overline{z}_1 + \overline{z}_2 + \cdots + \overline{z}_\nu + \cdots + \overline{z}_\eta\right)$. Note that these η complete subcubes are overlapped. Let Π and Σ denote the product and the summation of Boolean variables, then the preceding expression becomes $\prod_{i=1}^{\alpha} y_i (\sum_{j=1}^{\eta} \overline{z}_j)$.

Based on its recursive construction nature, I_s^M is composed of $I^{\Omega(M,1)}$ and $I^{\Omega(M,\ge 2)}$, which denotes the incomplete subcube comprising $\left\{ I^{\Omega(M,\delta)} \middle| \delta = 2, 3, \cdots \right\}$. The minterm representation of any node in incomplete subcube $I^{\Omega(M,\ge 2)}$ must involve Boolean variables $y_1, y_2, \cdots$, and y_α (since it is in MC) as well as Boolean variables $z_1, z_2, \cdots$, and z_η (for, otherwise, it would be in $I^{\Omega(M,1)}$, whose minterm involves the factor of $\sum_{j=1}^{\eta} \overline{z}_j$). In other words, the minterm representation of $I^{\Omega(M,\ge 2)}$ contains variables $y_1, y_2, \cdots, y_{\alpha,} z_1, z_2, \cdots$, and z_η (perhaps plus some other Boolean variables). Since the minterm representation of $CS(1)$ is $y_1 y_2 \cdots y_\alpha z_1 z_2 \cdots z_\eta$, $I^{\Omega(M,\ge 2)}$ must fall completely in $CS(1)$, as stated in the lemma below.

Lemma 1. *Incomplete subcube $I^{\Omega(M,\ge 2)}$ is contained entirely in $CS(1)$.*

In general, we may obtain the result with respect to incomplete subcube $I^{\Omega(M,\ge l)}$ for any $l, l \ge 2$, following a similar argument as above: $I^{\Omega(M,\ge l)}$ is contained entirely in $CS(l-1)$, where $CS(l-1)$ denotes the conjugate subcube of incomplete subcube $\Xi_l = (I^{\Omega(M,1)} \cup I^{\Omega(M,2)} \cup \cdots \cup I^{\Omega(M,l-1)})$. This general result ensures that a specific incomplete subcube in $CS(l-1)$ together with Ξ_l forms I_s^M, and it provides the basis of our expression for I_s^M. Consider the previous example of M = <1 1 1 0 0 1 1 0 1>: $CS(1)$ is a six-dimensional complete subcube, in which a specific incomplete subcube of size 13 (= <1 1 0 1>) together with $I^{\Omega(M,1)}$ forms I_9^M. It should be noted that the specific constituent incomplete subcube of I_9^M is the *largest* one in $CS(1)$, which involves one or multiple faults, since *any* incomplete subcube in $CS(1)$, together with $I^{\Omega(M,1)}$, can form I_9^M. Let $I^{\Omega(M,1)}$ and $I^{\Omega(M,\ge 2)}$ be expressed respectively by $\prod_{i=1}^{\alpha_1} y_{i,1} (\sum_{j=1}^{\eta_1} \overline{z}_{j,1})$ and $\prod_{i=1}^{\alpha_1} y_{i,1} \prod_{j=1}^{\eta_1} z_{j,1} \Phi_1$, then the expression for I_s^M is given by

$$\prod_{i=1}^{\alpha_1} y_{i,1} (\sum_{j=1}^{\eta_1} \overline{z}_{j,1} + \prod_{j=1}^{\eta_1} z_{j,1} \Phi_1).$$

According to the rule of $E + \overline{E}G = E + G$, the preceding expression is equal to

$$\prod_{i=1}^{\alpha_1} y_{i,1} (\sum_{j=1}^{\eta_1} \overline{z}_{j,1} + \Phi_1),$$

since $\sum_{j=1}^{\eta_1} \overline{z}_{j,1}$ and $\prod_{j=1}^{\eta_1} z_{j,1}$ are complement with each other. Incomplete subcube $I^{\Omega(M,2)}$ in $CS(1)$ can be identified in a similar way as finding $I^{\Omega(M,1)}$ in MC, and the remaining part $I^{\Omega(M,\ge 3)}$ is contained totally in $CS(2)$, giving rise to

$\Phi_1 = \prod_{i=1}^{\alpha_2} y_{i,2}(\sum_{j=1}^{\eta_2} \bar{z}_{j,2} + \Phi_2)$, where α_2 is the number of zero bits between the first run of "1" and the second run of "1" in the binary representation of M, η_2 is the number of nonzero bits in $\Omega(M,2)$, $y_{i,2}$ and $\bar{z}_{j,2}$ are Boolean variables b_is or $\bar{b}_i$s, and Φ_2 is determined by $I^{\Omega(M,\geq 3)}$ inside $CS(2)$.

This process repeats until $\Omega(M, t+1) = 0$, for a certain t, and, at that time, all the runs of "1" in M have been exhausted, resulting in the target incomplete subcube I_s^M. The general expression for I_s^M is, thus,

$$\prod_{i=1}^{\alpha_1} y_{i,1}(\sum_{j=1}^{\eta_1} \bar{z}_{j,1} + \prod_{i=1}^{\alpha_2} y_{i,2}(\sum_{j=1}^{\eta_2} \bar{z}_{j,2} + \cdots + \prod_{i=1}^{\alpha_t} y_{i,t}(\sum_{j=1}^{\eta_t} \bar{z}_{j,t})\cdots)). \quad (1)$$

As an instance, the general expression for incomplete subcube I_5^{26} in H_n is $y_{1,1}y_{2,1}\cdots y_{n-5,1}(\bar{z}_{1,1} + \bar{z}_{2,1} + y_{1,2}(\bar{z}_{1,2}))$, because, from 26 = <1 1 0 1 0>, $\alpha_1 = n - 5$, $\eta_1 = 2$, $\alpha_2 = 1$, and $\eta_2 = 1$. For n = 5, this general expression is reduced to $\bar{z}_{1,1} + \bar{z}_{2,1} + y_{1,2}(\bar{z}_{1,2})$, which agrees with W given in Section 2 for representing I_5^{26}, by setting $\bar{z}_{1,1} = \bar{b}_4, \bar{z}_{2,1} = \bar{b}_3, y_{1,2} = \bar{b}_2$, and $\bar{z}_{1,2} = \bar{b}_1$, as expected. Similarly, the expression for its conjugate subcube, CS_3^6, is $y_{1,1}y_{2,1}(\bar{z}_{1,1} + \bar{z}_{2,1})$ for $\alpha_1 = 5 - 3$ and $\eta_1 = 2$, in agreement with $\overline{W}$ given in Section 2. Note that a reconfigured incomplete hypercube often requires renumbering the constituent nodes logically, and that is why variables $y_{i,l}$ and $\bar{z}_{j,l}$ in the general expression can be any Boolean values b_v or $\bar{b}_v$, $0 \leq v < n$.

5 Finding Incomplete Subcubes Involving a Given Node

A *proper* incomplete subcube in a faulty hypercube refers to a fault-free incomplete subcube which is not contained entirely in any other fault-free subcube. This section deals with identifying all the proper incomplete subcubes from P, which equals $\overline{R}$ (where R is the summation of all reject regions represented in their minterms, as described in Section 3), because nonproper incomplete subcubes cannot be the largest ones. While a complete subcube can be represented by a minterm, an incomplete subcube, in general, can be expressed by the form given in (1), called a *comterm* (standing for a compound term, denoted by C_i). It is easy to see from (1) that a comterm is reduced to a minterm if $\alpha_i = 0$, for all $i \geq 2$, and $\eta_j = 0$, for all $j \geq 1$. A comterm is thus an extended expression for subcubes, whether complete or incomplete ones.

In order to identify all fault-free subcubes, we convert P (in the product-of-sum form) to its sum-of-comterm equivalence, which signifies the collection of all the proper incomplete subcubes. During the conversion process, the distributive law is applied to maxterms (sumterms) repeatedly until all expressions obtained are in the form of comterms given by (1). Applying the distributive law to the first maxterm of $x(y + z)(a + b)$ results in $xy(a + b) + xz(a + b)$, where terms $xy(a + b)$ and $xz(a + b)$ are called the *genterms* (meaning generalized terms, denoted by G_i, which comprise the products of minterms and maxterms).

5.1 Conversion Process

The conversion process involves two steps:

1) determining an appropriate maxterm from P and applying the distributive law to the determined maxterm after it is added to P, and
2) extracting the largest common component present in all the maxterms.

For a given P, the appropriate maxterm determined by and to be added to P contains *all the variables which appear in one or multiple maxterms* of P. Consider genterm $x(y + z)(a + b)$ given above, represented by G_1, as an example. In Step 1, the maxterm determined is $(y + z + a + b)$, which is added to G_1, and the distributive law is applied to the added maxterm in $x(y + z + a + b)(y + z)(a + b)$, giving rise to $xy(y + z)(a + b) + xz(y + z)(a + b) + xa(y + z)(a + b) + xb(y + z)(a + b)$, which becomes $xy(a + b) + xz(a + b) + xa(y + z) + xb(y + z)$ after unnecessary terms are dropped; for example, term $xyz(a + b)$ is dropped, since it is contained in $xy(a + b)$. The reason for Step 1 to be involved is because comterms, unlike minterms or maxterms, do not possess the commutative property, and all the distinct comterms specified by P have to be derived, so that all the proper incomplete subcubes referred to by P are explored. Without Step 1, some comterms could be left out, possibly making certain proper incomplete subcubes unexplored, and, thus, rendering it unable to identify *all* maximum incomplete subcubes in existence. Example G_1 without Step 1 yields only two comterms: $xy(a + b)$ and $xz(a + b)$, with the other two $xa(y + z)$ and $xb(y + z)$ left out.

The purpose of Step 2 is to guarantee producing comterms which denote the largest incomplete subcubes possible. Take $G_2 = x(a + y)(a + b + c)(a + b + d)$ as an instance. All the three maxterms have common component "a," which is extracted, leading to $x(a + y(b + c)(b + d))$, where a new genterm $y(b + c)(b + d)$ is created. Step 2) is applied again, resulting in comterm $x(a + y(b + cd))$, which refers to a largest incomplete subcube. For any given P, Steps 1 and 2 apply repeatedly until every expression obtained is a comterm, and each expression then refers to one incomplete subcube.

Consider a given node 000110 in H_6 with four faults: 010011, 101111, 100101, and 110110. The antipodal node of 000110 is 111001, and the four reject regions are 010011 ⊗ 111001, 101111 ⊗ 111001, 100101 ⊗ 111001, and 110110 ⊗ 111001, yielding $R = b_4\bar{b}_2b_0 + b_5b_3b_0 + b_5\bar{b}_1b_0 + b_5b_4$. Expression P for the given node is

$$P = \left(\bar{b}_4 + b_2 + \bar{b}_0\right)\left(\bar{b}_5 + \bar{b}_3 + \bar{b}_0\right)\left(\bar{b}_5 + b_1 + \bar{b}_0\right)\left(\bar{b}_5 + \bar{b}_4\right). \quad (2)$$

This expression is converted into its sum-of-comterm equivalence by applying the above two steps repeatedly, as depicted in Fig. 2, where the result is the collection of the incomplete subcube specified by comterms given at the leaves of the tree.

At the first level of conversion, the distributive law is applied to $(\bar{b}_0 + b_1 + b_2 + \bar{b}_3 + \bar{b}_4 + \bar{b}_5)$ according to conversion Step 1, yielding six genterms, each due to one variable, as indicated by an arrow with the variable shown next to it.

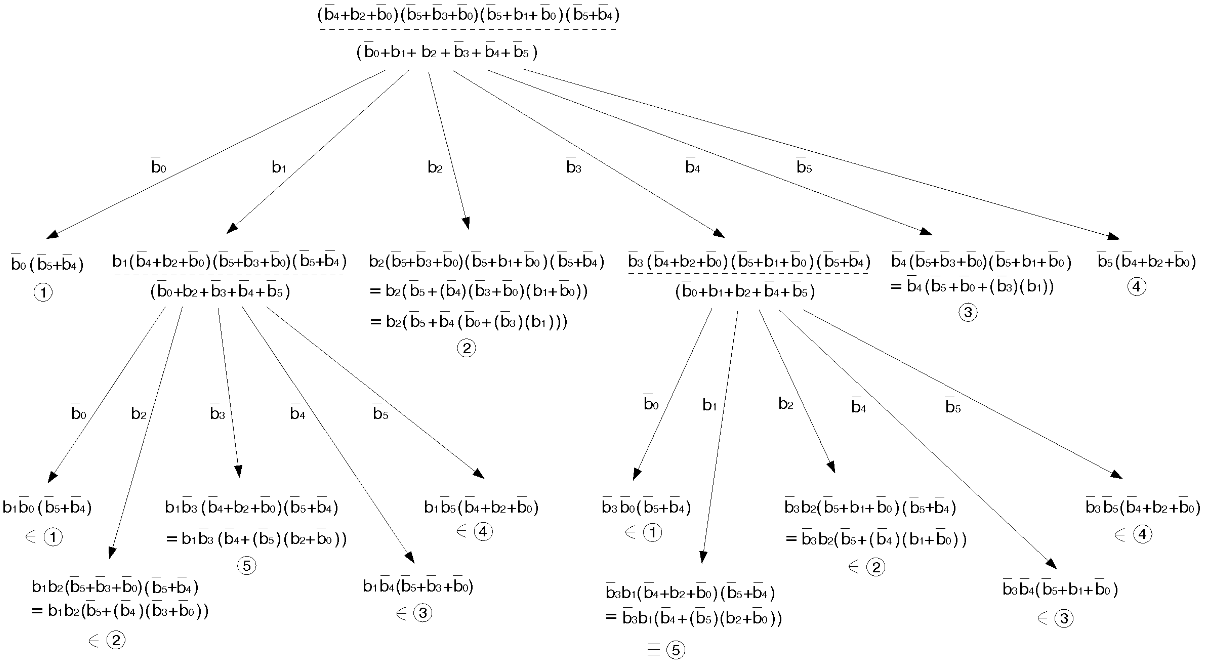
$(\bar{b}_4+b_2+\bar{b}_0)(\bar{b}_5+\bar{b}_3+\bar{b}_0)(\bar{b}_5+b_1+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$(\bar{b}_0+b_1+b_2+\bar{b}_3+\bar{b}_4+\bar{b}_5)$
$\bar{b}_0$
b_1
b_2
$\bar{b}_3$
$\bar{b}_4$
$\bar{b}_5$
$\bar{b}_0(\bar{b}_5+\bar{b}_4)$
①
$b_1(\bar{b}_4+b_2+\bar{b}_0)(\bar{b}_5+\bar{b}_3+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$(\bar{b}_0+b_2+\bar{b}_3+\bar{b}_4+\bar{b}_5)$
$b_2(\bar{b}_5+\bar{b}_3+\bar{b}_0)(\bar{b}_5+b_1+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$=b_2(\bar{b}_5+(\bar{b}_4)(\bar{b}_3+\bar{b}_0)(b_1+\bar{b}_0))$
$=b_2(\bar{b}_5+\bar{b}_4(\bar{b}_0+(\bar{b}_3)(b_1)))$
②
$\bar{b}_3(\bar{b}_4+b_2+\bar{b}_0)(\bar{b}_5+b_1+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$(\bar{b}_0+b_1+b_2+\bar{b}_4+\bar{b}_5)$
$\bar{b}_4(\bar{b}_5+\bar{b}_3+\bar{b}_0)(\bar{b}_5+b_1+\bar{b}_0)$
$=\bar{b}_4(\bar{b}_5+\bar{b}_0+(\bar{b}_3)(b_1))$
③
$\bar{b}_5(\bar{b}_4+b_2+\bar{b}_0)$
④
$\bar{b}_0$
b_2
$\bar{b}_3$
$\bar{b}_4$
$\bar{b}_5$
$b_1\bar{b}_0(\bar{b}_5+\bar{b}_4)$
$\in$ ①
$b_1b_2(\bar{b}_5+\bar{b}_3+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$=b_1b_2(\bar{b}_5+(\bar{b}_4)(\bar{b}_3+\bar{b}_0))$
$\in$ ②
$b_1\bar{b}_3(\bar{b}_4+b_2+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$=b_1\bar{b}_3(\bar{b}_4+(\bar{b}_5)(b_2+\bar{b}_0))$
⑤
$b_1\bar{b}_4(\bar{b}_5+\bar{b}_3+\bar{b}_0)$
$\in$ ③
$b_1\bar{b}_5(\bar{b}_4+b_2+\bar{b}_0)$
$\in$ ④
$\bar{b}_0$
b_1
b_2
$\bar{b}_4$
$\bar{b}_5$
$\bar{b}_3\bar{b}_0(\bar{b}_5+\bar{b}_4)$
$\in$ ①
$\bar{b}_3b_1(\bar{b}_4+b_2+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$=\bar{b}_3b_1(\bar{b}_4+(\bar{b}_5)(b_2+\bar{b}_0))$
$\equiv$ ⑤
$\bar{b}_3b_2(\bar{b}_5+b_1+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$=\bar{b}_3b_2(\bar{b}_5+(\bar{b}_4)(b_1+\bar{b}_0))$
$\in$ ②
$\bar{b}_3\bar{b}_4(\bar{b}_5+b_1+\bar{b}_0)$
$\in$ ③
$\bar{b}_3\bar{b}_5(\bar{b}_4+b_2+\bar{b}_0)$
$\in$ ④

Fig. 2. Converting a product-of-sum expression into its sum-of-comterm equivalence.

Conversion Step 2 is then applied to each created genterm which has a common variable present in all its maxterms. The genterm created due to b_2, i.e.,

$$b_2(\bar{b}_5+\bar{b}_3+\bar{b}_0)(\bar{b}_5+b_1+\bar{b}_0)(\bar{b}_5+\bar{b}_4),$$

for example, has variable $\bar{b}_5$ in common and, thus, becomes $b_2(\bar{b}_5+\bar{b}_4(\bar{b}_3+\bar{b}_0)(b_1+\bar{b}_0))$ after the common variable is extracted. This extraction process repeats for genterm $\bar{b}_4(\bar{b}_3+\bar{b}_0)(b_1+\bar{b}_0)$, as all its maxterms involve common variable $\bar{b}_0$, leading to $b_2(\bar{b}_5+\bar{b}_4(\bar{b}_0+(\bar{b}_3)(b_1)))$, as illustrated in Fig. 2. Similarly, Step 2 is applied to the genterm produced due to $\bar{b}_4$. After common variables are extracted, genterms produced due to b_2 and $\bar{b}_4$ are in the form of comterms as desired.

For the genterms created due to variables b_1 and $\bar{b}_3$, they are not in the form of comterms yet, so Step 1 is followed to distribute, respectively, $(\bar{b}_0+b_2+\bar{b}_3+\bar{b}_4+\bar{b}_5)$ and $(\bar{b}_0+b_1+b_2+\bar{b}_4+\bar{b}_5)$ over them, as depicted in the second level of Fig. 2. After this distribution, some newly produced genterms require Step 2 to extract common components, arriving at the desired comterm form. In this case, the expressions obtained after Step 2 are all in the form of comterms and the conversion process finishes.

5.2 Simplification Criteria

In the course of conversion shown in Fig. 2, many unnecessary comterms are produced, suggesting that simplification could be incorporated to save conversion effort (where a comterm is unnecessary if it is identical, or belongs, to another comterm). Our goal is to avoid unnecessary comterms from being created in the conversion process so that the comterms produced at the end all correspond to proper incomplete subcubes, involving much less effort. The subsequent two lemmas provide the basis of our simplification criteria to be included in the conversion process.

LEMMA 2. *Let expressions E_i and E_j be generated from E_f due to variables b_i and b_j, respectively. If all maxterms in E_i involve a common component and so do all maxterms in E_j, then incomplete subcubes specified by E_i and E_j, denoted, respectively, as IS_i and IS_j, satisfy $IS_i \not\subset IS_j$ and $IS_j \not\subset IS_i$.*

PROOF. Assume the genterm in E_f being G_f (note that E_f generates other expressions via distribution only if it is not in the comterm form and involves a genterm). Since distribution through Step 1 is done over the G_f part only, leaving the rest unchanged, the following discussion focuses merely on the results in E_i and E_j contributed by G_f. Let the results be denoted, respectively, by $b_i(C_i + G_i)$ and $b_j(C_j + G_j)$ after Step 2 is applied, where C_i and C_j are common components extracted, and G_i and G_j are genterms obtained after extraction. Two cases are discussed separately below, depending on the numbers of variables contained in C_i and C_j.

1) Both C_i and C_j contain two or more variables.

In this case, b_iC_i and b_jC_j would satisfy $b_iC_i \not\subset b_jC_j$ and $b_jC_j \not\subset b_iC_i$, regardless of what are involved in C_i and C_j. This means $IS_i \not\subset IS_j$ and $IS_j \not\subset IS_i$.

2) Either C_i or C_j contains exactly one variable.

Let us examine the situation that C_i contains exactly one variable, say b_k, and C_j contains one or more variables. In this situation, if $b_k \neq b_j$ or C_j doesn't contain b_i, then we have $b_ib_k \not\subset b_jC_j$ and $b_jC_j \not\subset b_ib_k$. On the other hand, if $b_k = b_j$ and C_j contains b_i, G_i doesn't

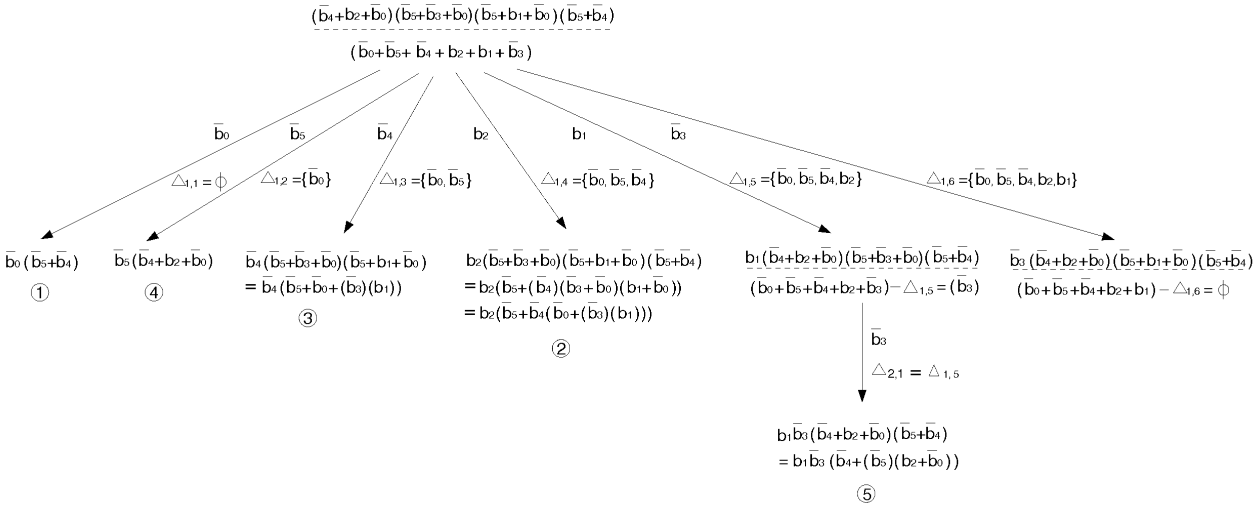
$(\bar{b}_4+b_2+\bar{b}_0)(\bar{b}_5+\bar{b}_3+\bar{b}_0)(\bar{b}_5+b_1+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$(\bar{b}_0+\bar{b}_5+\bar{b}_4+b_2+b_1+\bar{b}_3)$
$\bar{b}_0$
$\bar{b}_5$
$\bar{b}_4$
b_2
b_1
$\bar{b}_3$
$\triangle_{1,1}=\phi$
$\triangle_{1,2}=\{\bar{b}_0\}$
$\triangle_{1,3}=\{\bar{b}_0,\bar{b}_5\}$
$\triangle_{1,4}=\{\bar{b}_0,\bar{b}_5,\bar{b}_4\}$
$\triangle_{1,5}=\{\bar{b}_0,\bar{b}_5,\bar{b}_4,b_2\}$
$\triangle_{1,6}=\{\bar{b}_0,\bar{b}_5,\bar{b}_4,b_2,b_1\}$
$\bar{b}_0(\bar{b}_5+\bar{b}_4)$
①
$\bar{b}_5(\bar{b}_4+b_2+\bar{b}_0)$
④
$\bar{b}_4(\bar{b}_5+\bar{b}_3+\bar{b}_0)(\bar{b}_5+b_1+\bar{b}_0)$
$=\bar{b}_4(\bar{b}_5+\bar{b}_0+(\bar{b}_3)(b_1))$
③
$b_2(\bar{b}_5+\bar{b}_3+\bar{b}_0)(\bar{b}_5+b_1+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$=b_2(\bar{b}_5+(\bar{b}_4)(\bar{b}_3+\bar{b}_0)(b_1+\bar{b}_0))$
$=b_2(\bar{b}_5+\bar{b}_4(\bar{b}_0+(\bar{b}_3)(b_1)))$
②
$b_1(\bar{b}_4+b_2+\bar{b}_0)(\bar{b}_5+\bar{b}_3+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$(\bar{b}_0+\bar{b}_5+\bar{b}_4+b_2+\bar{b}_3)-\triangle_{1,5}=(\bar{b}_3)$
$\bar{b}_3(\bar{b}_4+b_2+\bar{b}_0)(\bar{b}_5+b_1+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$(\bar{b}_0+\bar{b}_5+\bar{b}_4+b_2+b_1)-\triangle_{1,6}=\phi$
$\bar{b}_3$
$\triangle_{2,1}=\triangle_{1,5}$
$b_1\bar{b}_3(\bar{b}_4+b_2+\bar{b}_0)(\bar{b}_5+\bar{b}_4)$
$=b_1\bar{b}_3(\bar{b}_4+(\bar{b}_5)(b_2+\bar{b}_0))$
⑤

Fig. 3. Conversion with the simplification criteria incorporated.

contain b_j and G_j doesn't contain b_i, leading to $b_iG_i \not\subset b_jG_j$ and $b_jG_j \not\subset b_iG_i$. In this situation, the relation of $IS_i \not\subset ISj$ and $IS_j \not\subset IS_i$ always holds. The other situation that C_j contains exactly one variable can be proved similarly. □

This lemma indicates that all expressions derived from E_f have to be treated further, if they may follow conversion Step 2 to extract common components; none of them can be discarded immediately. In Fig. 2, for example, expressions derived due to b_2 and $\bar{b}_4$ in the first level are treated until they are in the comterm form, referring to two proper incomplete subcubes.

LEMMA 3. *Let expressions E_i and E_j be generated from E_f due to variables b_i and b_j, respectively. If all maxterms in E_j share no common component, then the incomplete subcube specified by the expression produced subsequently from E_j due to b_i is contained entirely in the incomplete subcube specified by E_i.*

PROOF. As before, our focus is limited to the results caused by distribution over genterm G_f in E_f, leaving out the unchanged parts. Suppose that, after distribution, E_i involves genterm b_iG_i and E_j involves genterm b_jG_j, where G_i (or G_j) is G_f with every maxterm containing b_i (or b_j) dropped. Since all maxterms in G_j share no common component and E_j is not in the comterm form yet, conversion Step 1 is applied again, producing an expression, say E_k, due to variable b_i. It is clear that E_k involves $b_jb_iG_k$, where genterm G_k is G_j, with every maxterm containing b_i discarded. As G_k is G_f with every maxterm containing either b_j or b_i dropped, G_i involves more maxterms than G_k, implying that $b_jb_iG_k \subset b_iG_i$. In other words, the expression produced subsequently from E_j due to b_i, i.e., E_k, is contained in E_i. □

This lemma reveals an interesting attribute that tells which expressions to be generated are contained in prior expressions and, thereby, can be dropped, often resulting in significantly less conversion effort. To this end, a set of "discarded" variables is kept for each expression generated by distribution, where the set involves all the Boolean variables that have been distributed (over the same expression) so far. Take the first level of Fig. 3 as an example. There are six variables to be distributed over the given (same) expression, and the set of discarded variables for expression generated due to b_1 is $\{\bar{b}_0, \bar{b}_5, \bar{b}_4, b_2\}$, as depicted in the figure. The set of discarded variables is forwarded to the next level in order to prevent distributing them over the expression generated (due to variable b_1 in this example), because, according to Lemma 3, if all the maxterms in this generated expression share no common component, any expression produced subsequently by distributing a discarded variable is unnecessary. Distribution over the expression generated due to variable b_1 in level 1 is thus carried out for variable $\bar{b}_3$ only, excluding all discarded variables as shown in level 2 of Fig. 3. It should be noted that, if all maxterms in a generated expression share a common component, the set of discarded variables is flushed and distribution, if needed, is performed for every variable existing in maxterms.

From Lemmas 2 and 3, we arrive at the simplification criteria to be incorporated in our conversion process.

1) The distribution sequence for an expression, say, E, follows that the variable which yields an expression with a common variable present in all its maxterms is treated first (recall that, if a variable appears in all maxterms, the variable is extracted); in case there are multiple choices, the one that appears in more maxterms of E is selected earlier; and
2) The set of discarded variables is produced for each expression generated at level i and the set is employed by the expression generated to avoid unnecessary distribution at level $i + 1$, provided that all the maxterms in the expression share no common component.

Making use of these simplification criteria, we obtain the conversion result of expression P for the faulty H_6 given by (2), as illustrated in Fig. 3, where the set of discarded variables for the *j*th expression generated in level *i* is denoted by $\Delta_{i,j}$ (with expressions in a level numbered rightwards). In the first level, variables $\bar{b}_0, \bar{b}_5, \bar{b}_4$, and b_2 all yield expressions with common variables present in all their maxterms, but $\bar{b}_0$ and $\bar{b}_5$ are distributed earlier, since they appear in the largest number of maxterms of *P*. The set of discarded variables for expression $b_1(\bar{b}_4 + b_2 + \bar{b}_0)(\bar{b}_5 + \bar{b}_3 + \bar{b}_0)(\bar{b}_5 + \bar{b}_4), \Delta_{1,5}$, is used during level 2 distribution by the expression, as shown in the figure. If the set of discarded variables for an expression involves all maxterm variables, the expression requires no distribution, like the last expression in level 1 of Fig. 3. The conversion process with the aid of the simplification criteria produces only necessary comterms (i.e., proper incomplete subcubes, see Fig. 3) as desired, saving lots of effort, and the collection of comterms produced is identical to that depicted in Fig. 2 after those unnecessary comterms are removed.

An algorithm for generating the sum-of-comterm (SOC) equivalence of a given product-of-sum expression *P* is provided below, where a stack is employed to keep expressions produced in the course of conversion, as well as their corresponding sets of discarded variables. The stack holds only expression *P* initially, and both the set of discarded variables for *P* and SOC are initialized with ∅. After the algorithm ends, all comterms produced are stored in SOC, which is then scanned to find the largest size.

Algorithm A. (generate a sum-of-comterm equivalence):

```
While (the stack is not empty)
  {
        Pop an expression $E_i$, together with its corresponding set of discarded variables, from the stack;
        Let $G_i$ be the genterm involved in $E_i$;
        While (all maxterms in $G_i$ share a common variable)
          {
            Extract the common variable according to conversion Step 2;
            Flush the set of discarded variables;
          }
        If ($G_i$ is in the comterm form, i.e., involves no more than one maxterm)
            Append the comterm derived from $E_i$ to SOC;
        else {
                Apply the distributive law to each variable in maxterms of $G_i$, excluding discarded variables;
                Determine the sets of discarded variables for expressions newly generated;
                Push all generated expressions and their corresponding sets of discarded variables into the stack.
             }
  }
```

Since our conversion process explores all incomplete subcubes which are specified by expression *P* and which contain a given node, Algorithm A is insured to identify all proper incomplete subcubes with respect to the given node, as the simplification criteria remove only unnecessary expressions. The size of an incomplete subcube specified by a comterm can be directly obtained from (1), as follows: Let the size be represented as an *n*-bit string, then the bit string (starting from the leftmost bit) consists of α_1 0's, followed by η_1 1's, followed by α_2 0's, followed by η_2 1's, and so on, until the comterm is exhausted; if the string formed is of length less than *n*, say *l*, then $(n - l)$ 0's are appended to the string. For example, comterm $\bar{b}_5(\bar{b}_4 + b_2 + \bar{b}_0)$ in Fig. 3 is of size 28 (= 011100_2, as $\alpha_1 = 1$ and $\eta_1 = 3$, with two appended 0's), whereas comterm $b_2(\bar{b}_5 + \bar{b}_4(\bar{b}_0 + (\bar{b}_3)(b_1)))$ is of size 21 (= 010101_2, as $\alpha_i = \eta_j = 1$, for all $1 \le i, j \le 3$). After examing all five comterms produced in Fig. 3, we find that the size of the largest incomplete subcube characterized by *P*, given in (2), is 28.

5.3 Time Complexity

The worst case time complexity of Algorithm A can be derived by finding out how many expressions (i.e., E_i's) at most are examined during conversion. All expressions examined constitute a tree structure (called an *expansion tree*, see Fig. 3), with an expression corresponding to a tree node and leaf nodes denoting incomplete subcubes obtained. While Algorithm A is implemented using a stack (for more efficiency), its time complexity can be derived from a recursive standpoint. Let $\Lambda_m(n)$ denote the worst case time complexity of Algorithm A applied to H_n with *m* faults. Initially, expression *P* contains up to *m* maxterms, one corresponding to a fault, and each maxterm involves at most $n - 1$ (distinct) variables for an *n*-dimensional hypercube. When the distributive law is first applied, it produces *n* expressions, according to conversion Step 1. Since expression *P* involves up to $m(n - 1)$ variables, and, applying the distributive law goes through *P* exactly *n* times, the time complexity of this step is $O(m \times n^2)$.

Now, every one of these produced *n* expressions corresponds to an $(n - 1)$-dimension hypercube, which is specified by the distributed variable. It is apparent that such an $(n - 1)$-dimension hypercube contains no more than *m* faults. There are two possibilities to every such produced expression: applying conversion Step 2, or applying Algorithm A. They both examine the maximum healthy subcubes inside H_{n-1} and are applied exclusively (see Fig. 3). When a common variable is identified by conversion Step 2, it is extracted from all maxterms, leaving the maxterms corresponding to H_{n-2}; otherwise, Algorithm A is applied. If Algorithm A is applied, its time complexity is $\Lambda_m(n-1)$, neglecting the effect of the use of discarded variables (which tends to largely reduce complexity). As applying conversion Step 2 leads to time complexity less than $\Lambda_m(n-1)$, the time complexity of examining each H_{n-1} is $\Lambda_m(n-1)$. This suggests the recurrence inequality of $\Lambda_m(n) \le O(m \times n^2) + n \times \Lambda_m(n-1)$, with $\Lambda_m(2) = 1$. Following the exponential generating functions, we arrive at $\Lambda_m(n) \le O(m \times n!)$. The time complexity of Algorithm A is thus upper bounded by $O(m \times n!)$.

6 Distributed Identification of All Maximum Incomplete Subcubes

Algorithm A lends itself perfectly to distributed identification by being executed at every healthy node independently, with the executing node itself treated as the given node. It is assumed that the set of faulty nodes is uncovered in a distributed manner by fault-free nodes, following the diagnostic algorithm introduced by Armstrong and Gray [14]. After fault diagnosis is done, the address of a faulty node is broadcast to all nodes by a healthy neighbor, and every healthy node keeps this broadcast information.

On receiving the addresses of all faulty nodes, each healthy node identifies all the largest incomplete subcubes involving the node itself, using Algorithm A. For a faulty H_n, there are, in general, $O(2^n)$ nodes participating in the identification process, and information calculated at each participant about the largest incomplete subcubes has to be taken into account in reaching the decision on maximum incomplete subcubes globally, i.e., the size of the maximum incomplete subcubes in the whole system is determined based on the largest incomplete subcube size found at each participating node.

We assume that the hypercube system has one host (also known as the service node or system manager), which is in charge of reconfiguration after receiving the comterms denoting maximum incomplete subcubes (called the *maximum comterms* for short), and which has a direct connection to each cube node, like the Intel *i*PSC/860 [3] and *n*-CUBE 2 [4]. A distributed approach to determining the size of maximum incomplete subcubes is introduced next, followed by an algorithm which ensures sending all maximum comterms to the host nonredundantly. Cube nodes are partitioned into $n + 1$ levels, according to the number of nonzero bits in their labels. For example, node 0000 is in level 0, nodes 1000 and 0010 are in level 1, and so on. A level i node has neighbors in level $i - 1$ or $i + 1$.

6.1 Size Determination

Consider a fault-free node at level i, $0 < i \le n$, in H_n. The node, after carrying out Algorithm A, waits to receive the size of the largest incomplete subcube(s) determined at a healthy neighbor in level $i - 1$. A node is assumed to know the status of all its neighbors and ignores messages from its faulty neighbors. Once receiving messages from all its healthy level $i - 1$ neighbors, the node chooses the largest one among all received sizes and the size it found using Algorithm A, and sends the chosen largest size to all its neighbors in level $i + 1$ (where the level $i + 1$ neighbor of node 1^n is the host). Node 0^n receives no messages and sends the largest size it found to all its neighbors immediately after finishing Algorithm A.

In case a node in level i has no fault-free neighbor in level $i - 1$, it forwards the largest size it found immediately to all its neighbors in level $i + 1$. On the other hand, if a level i node has no healthy neighbor in level $i + 1$, it simply sends the largest size chosen to the host. In the last step of this determination, if the host receives multiple sizes, it selects the largest one as the size of the maximum comterm(s). The maximum size is then broadcast to all healthy nodes by the host, so that each node knows whether or not it contains the maximum comterm(s). For an n-dimensional hypercube with m faults, the total number of messages sent to the host depends on the distribution of these faults and is always less than $n \times m$, since the failure of node 1^n causes n messages to be sent to the host, and any other fault results in fewer than n messages directed to the host.

6.2 Gathering Maximum Comterms

All maximum comterms are forwarded to the host so as to facilitate reconfiguration. It is desirable that comterms are sent to the host nonredundantly, in an attempt to avoid unnecessary traffic and computation. The subsequent theorem provides the basis for nonredundant delivery of comterms to the host.

THEOREM 2. *All the healthy nodes which come out with an identical maximum comterm form a fault-free complete subcube.*

PROOF. The comterm is generally expressed as (1). Let the maximum comterm of interest be

$$C_{\max} = \prod_{i=1}^{\alpha_1} y_{i,1}(\sum_{j=1}^{\eta_1} \bar{z}_{j,1} + \prod_{i=1}^{\alpha_2} y_{i,2} (\sum_{j=1}^{\eta_2} \bar{z}_{j,2} + \cdots + \prod_{i=1}^{\alpha_t} y_{i,t}(\sum_{j=1}^{\eta_t} \bar{z}_{j,t})\cdots)).$$

Now, consider the specific complete subcube, S_β, addressed by

$$\prod_{i=1}^{\alpha_1} y_{i,1} \prod_{j=1}^{\eta_1} \bar{z}_{j,1} \prod_{i=1}^{\alpha_2} y_{i,2} \prod_{j=1}^{\eta_2} \bar{z}_{j,2} \cdots \prod_{i=1}^{\alpha_t} y_{i,t} \prod_{j=1}^{\eta_t} \bar{z}_{j,t}. \quad (3)$$

It is clear that any node in S_β, say, node O, is contained in the following complete subcubes: $\prod_{i=1}^{\alpha_1} y_{i,1}\bar{z}_{j,1}$, for all $1 \le j \le \eta_1$; $\prod_{i=1}^{\alpha_1} y_{i,1} \prod_{i=1}^{\alpha_2} y_{i,2}\bar{z}_{j,2}$, for all $1 \le j \le \eta_2$; $\cdots$, and $\prod_{i=1}^{\alpha_1} y_{i,1} \prod_{i=1}^{\alpha_2} y_{i,2} \cdots \prod_{i=1}^{\alpha_t} y_{i,t}\bar{z}_{j,t}$, for all $1 \le j \le \eta_t$. Consequently, Algorithm A, with respect to node O, recognizes these complete subcubes and probes the incomplete subcube composed of all these complete subcubes, which is the largest incomplete subcube containing node O, and which is specified by $C_{\max}$. This indicates that node O comes out with $C_{\max}$, for any node O in S_β.

On the other hand, for any node X outside subcube S_β, it is easy to show that node X is not in at least one of the constituent complete subcubes of the incomplete subcube defined by $C_{\max}$. Thus, the largest incomplete subcube explored is different from the one specified by $C_{\max}$. □

According to Theorem 2, we let one and only one node in S_β be responsible for sending the maximum comterm(s) to the host, preventing any redundancy. Among all the S_β nodes, there is exactly one node at the highest level and that node is designated as the responsible node. The following algorithm elects the responsible node(s) in a distributed way, on the basis of the fact stated in Theorem 2: Each node in level $i - 1$, $0 < i \le n$, sends the largest comterm(s) it found, if any, to all its neighbors in level i; if no such comterm is involved, it sends out a specific string, say the empty string. A level i node responds to messages from level $i - 1$ nodes individually: If a received message carries the same comterm(s) as what the node found, the node responds by sending the comterm(s) back; else, it responds with the specific (empty) string. If the receiving node in level i involves no maximum comterms, it responds to every message from level $i - 1$ with the empty string. A node in level $i - 1$ waits until all responses from level i arrive, and if any of the received response involves the same comterm(s) as what the node itself found, the node is not a responsible node for directing the comterm(s) to the host; otherwise, it is a responsible node.

According to the preceding algorithm, the single node at the highest level within subcube S_β is identified as the responsible node, and every such subcube has one responsible node determined. All the maximum comterm(s) are thus forwarded to the host nonredundantly. As an example, consider H_4 with faults at 0010, 0110, 1010, and 1110, as depicted in Fig. 4. The maximum incomplete subcube available is of size 12, and its comterm is given by $\overline{b}_1 + b_0$. From Theorem 2, the nodes which come out with the maximum comterm fall into a complete subcube S_β, whose minterm representation is $\overline{b}_1 b_0$ (from (3)), or equivalently, **01. This can be quickly validated by calculating expression P for each node in **01 and then following Algorithm A to get the largest comterm; any other healthy node outside **01 has P which covers fewer nodes. Exactly one node within **01 is at the highest level, i.e., node 1101. According to the above algorithm, node 0001 receives the maximum comterm back from node 1001 and node 0101, whereas nodes 1001 and 0101 receive the maximum comterm back from node 1101, indicating that they are not responsible nodes. Since node 1101 receives the empty string back from node 1111, it is the responsible node.

As another example, if H_4 in Fig. 4 involves only two faults, 0010 and 1110 (rather than four faults), the maximum subcubes available are then of size 13, specified, respectively, by $\overline{b}_1 + b_0 + \overline{b}_2(b_3)$ and $\overline{b}_1 + b_0 + b_2(\overline{b}_3)$. The nodes which come out with these two maximum comterms following Algorithm A are $b_3\overline{b}_2\overline{b}_1 b_0$ and $\overline{b}_3 b_2\overline{b}_1 b_0$, respectively, each of which is a zero-dimensional subcube. They are identified as the responsible nodes trivially.

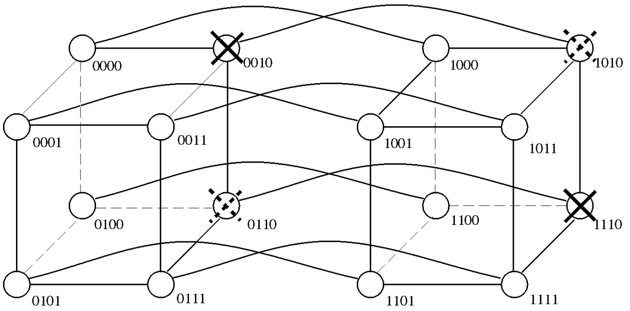

Fig. 4. A four-dimensional hypercube with faults.

7 Fault Simulation and Results

The results of our proposed identification strategy were collected using fault simulation for a 10-dimensional hypercube, in which the number of faulty nodes (m) ranges from two to 20. This simulation study was carried out on a SUN Sparc 10/20. For each given m, 5,000 uniformly distributed fault patterns were generated and reconfiguration was performed with respect to every fault pattern individually. The average size of maximum incomplete subcubes is plotted as a function of m in Fig. 5, with the mean size of largest complete subcubes included for comparison. It is observed that the gap between the average maximum incomplete subcube size and the mean maximum complete subcube size is the largest for $m = 1$, as might be expected. While the gap shrinks gradually as m increases, the former remains close to twice as large as the latter, for any m simulated. These results demonstrate that our strategy indeed gives rise to a significantly larger system than any prior scheme which identifies solely complete subcubes, suggesting the potential advantage of reconfiguring a faulty hypercube into a maximum incomplete one.

The average time spent for each involved node in carrying out the proposed reconfiguration algorithm is depicted in the same figure by dashed curves. Since only a SUN workstation was assigned to the whole simulation, the role

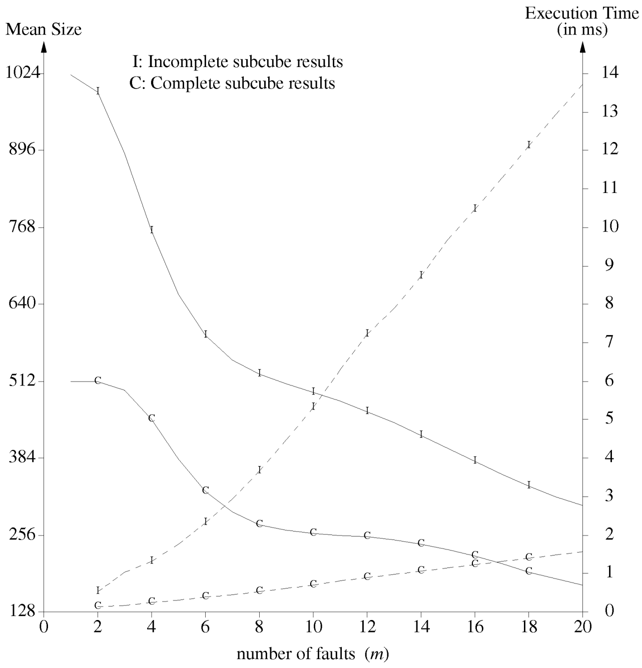

Fig. 5. Mean size and execution time versus number of faults, with solid (or dashed) curves for mean size (or execution time).

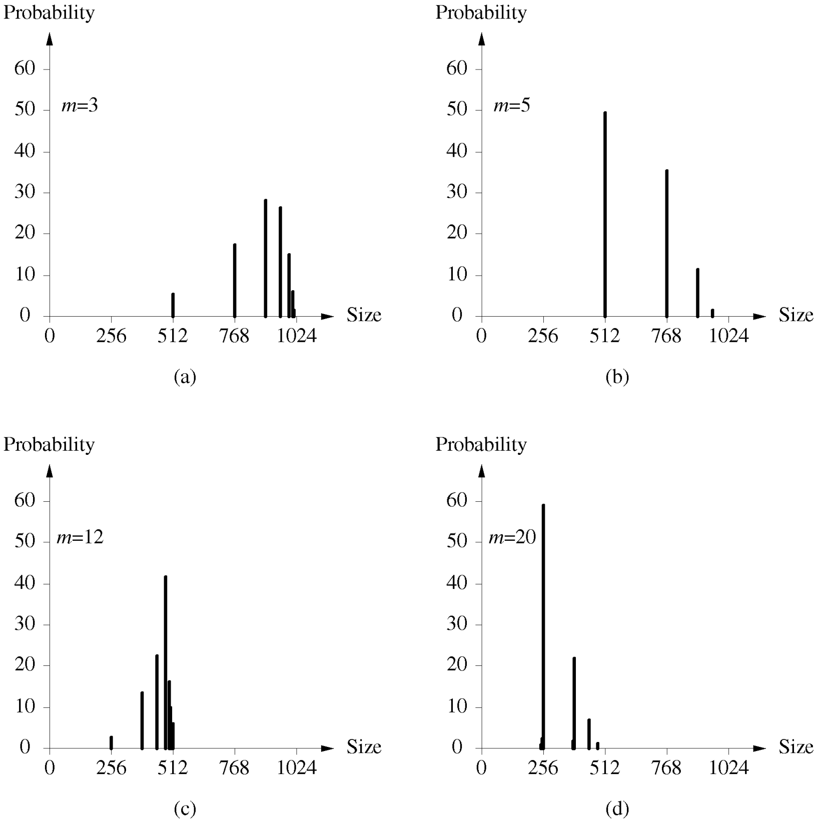

Fig. 6. Probabilities of the sizes of identified maximum incomplete subcubes.

of every healthy cube node was simulated in sequence, and the mean time was estimated by dividing the total time taken to simulate all the healthy nodes over the number of healthy nodes. This reveals the expected execution duration of Algorithm A. For a larger *m*, it takes a longer mean duration to finish the algorithm because more sumterms exist in expression *P* initially (recall that a fault results in one sumterm). To understand the worst case scenario, the largest execution time at each node for every *m* was also recorded, so as to determine the maximum time among all nodes for each *m*. For a given *m*, the maximum execution time is often several times as large as the mean execution time shown in Fig. 5. As an example, for $m = 2$, the maximum time is about triple the mean time, whereas for $m = 10$ or 20, the maximum time is roughly twice as large. The average time required for identifying largest complete subcubes following the approach described in [10] (which is the most efficient among all known techniques for complete subcube determination) is provided in Fig. 5, as well. Identifying all maximum incomplete subcubes takes anywhere from four- to nine-fold, as much as what is needed, for recognizing largest complete subcubes. Since reconfiguration is performed infrequently (only after faults occur), the longer time for reconfiguration into an incomplete system appears to be well spent, in view of its potential performance gain.

In order to offer an insight into the distributions of various sizes obtainable for a given number of faults in H_{10} under the proposed reconfiguration strategy, the probabilities of the sizes of identified maximum incomplete subcubes are shown in Fig. 6, in which the cases of probabilities lower than one percent are omitted. As can be seen from this figure, a larger number of faults makes it less likely to have a big incomplete subcube in existence. For example, when *m* equals three, the probability of finding a subcube with more than 900 nodes in H_{10} is 49 percent, whereas, for $m = 5$, that probability drops to less than two percent. If *m* is 20, the probability of getting a subcube with size 400 or larger is below 10 percent.

8 Concluding Remarks

A Boolean expression-based approach for identifying all maximum incomplete subcubes present in a faulty hypercube has been introduced. Every fault-free node is required to participate in the identification process by executing the same algorithm independently at the same time. The nodes responsible for sending the addresses of maximum incomplete subcubes to the host, after their size is determined in a distributed manner, are then elected through a distributed procedure. The host is ensured to receive the addresses of all maximum incomplete subcubes nonredundantly.

Extensive fault simulation indicates that, for a given faulty pattern, our proposed identification approach leads to approximately two times as large as the size obtained by an earlier scheme capable of recognizing only complete subcubes. While a system reconfigured following the proposed approach often does not contain all the healthy nodes (in order to ensure the incomplete hypercube topology), such a system enjoys essential properties like simple routing and broadcasting [11], bounded traffic density over links [12], [20] and efficient communication. If one intends to preserve all the healthy nodes (without any reconfiguration) after faults arise, complicated routing and broadcasting procedures are normally needed, and a message routed between a pair of nodes could encounter excessive communication delay. Further, allocating and executing jobs efficiently in this situation appear to be more difficult than on an incomplete hypercube.

Our discussion in this paper is limited to the cases where only cube nodes could fail. However, this approach can be extended to deal with the hypercube involving both node and link failures. Since a link joins two nodes, the failure of a link excludes one of the two end nodes, if they both are healthy, with an excluded node treated as a faulty node in calculating the reject regions. The excluded node is the one (of the two end nodes) which yields a smaller reject region. If any end node of a failed link is faulty, no additional node is excluded because the effect of the failed link is considered by a faulty node to which the link connects. With reject regions due to faulty nodes and links decided for a participating node, expression *P* can be obtained and Algorithm A is followed to get the largest incomplete subcube(s), as described in this paper. This distributed identification approach appears beneficial and practical for large hypercubes operating in a gracefully degraded mode.

APPENDIX A
PROOF OF THEOREM 1

PROOF. It is easy to observe that the conjugate subcube of $I^{\Omega(M,1)}$ is a complete subcube of size $2^{s-\eta}$. The minimum cover subcube of $I^{\Omega(M,1)}$ is of dimension *s*, denoted by *MC*, and suppose that its minterm representation is $y_1 y_2 \cdots y_k \cdots y_\alpha$, where $\alpha = n - s$ and y_k, $1 \le k \le \alpha$, is Boolean variable b_i or $\overline{b_i}$, $0 \le i \le n-1$ (recall that the minterm for H_s involves *n* – *s* Boolean variables). Since the conjugate subcube of $I^{\Omega(M,1)}$ is a complete subcube of dimension $s - \eta$, its minterm representation would be $y_1 y_2 \cdots y_\alpha z_1 z_2 \cdots z_\nu \cdots z_\eta$, where $z_\nu, 1 \le \nu \le \eta$, is Boolean variable b_i or $\overline{b_i}$. The conjugate subcube of $I^{\Omega(M,1)}$ is denoted by *CS*(1) in order to relate it to the first run of "1" (note that, if $I^{\Omega(M,1)}$ itself is a complete cube, it would have multiple conjugate subcubes; however, *CS*(1) then refers to the one which involves the conjugate subcube of I_s^M).

Consider the following η $(s-1)$-dimensional subcubes in *MC*: $y_1 y_2 \cdots y_\alpha \overline{z}_\nu, 1 \le \nu \le \eta$. Each of these η subcubes is contained entirely in $I^{\Omega(M,1)}$, because it shares no common node with *CS*(1), the conjugate subcube, and if a node in *MC* does not belong to *CS*(1), it must fall inside $I^{\Omega(M,1)}$ (according to the definition of conjugate subcubes given in Section 2). Thus, $I^{\Omega(M,1)}$ involves η $(s-1)$-dimensional subcubes. We then show that none of other (*s* – 1)-dimensional subcubes in *MC* is contained in $I^{\Omega(M,1)}$. This is achieved according to the next two facts about all the other (*s* – 1)-dimensional subcubes in *MC*, denoted as either $y_1 y_2 \cdots y_\alpha z_\nu, 1 \le \nu \le \eta$, or $y_1 y_2 \cdots y_\alpha x$, x being a Boolean variable *nomem* $\{\overline{z}_\nu \text{ or } z_\nu \mid 1 \le \nu \le \eta\}$. First, each subcube $y_1 y_2 \cdots y_\alpha z_\nu$ contains the conjugate subcube, *CS*(1) (from its minterm representation) and, therefore, does not fall completely inside $I^{\Omega(M,1)}$. Second, each subcube $y_1 y_2 \cdots y_\alpha x$ and *CS*(1) have a nonempty common subcube $y_1 y_2 \cdots y_\alpha z_1 z_2 \cdots z_\eta x$, and it is, thus, not contained wholly in $I^{\Omega(M,1)}$.

We then argue by contradiction that the union of these η $(s-1)$-dimensional subcubes $y_1 y_2 \cdots y_\alpha \overline{z}_\nu$, for all $1 \le \nu \le \eta$, is equal to $I^{\Omega(M,1)}$. Assume that there is one node *B* in $I^{\Omega(M,1)}$, but it is not contained in any of these η subcubes. The minterm representation of node *B* should be given by $y_1 y_2 \cdots y_\alpha z_1 z_2 \cdots z_\eta w_1 w_2 \cdots w_p \cdots w_{s-\eta}$, where w_p, $1 \le p \le s - \eta$, is a Boolean variable b_i or $\overline{b_i}$ (because only the representation of $y_1 y_2 \cdots y_\alpha z_1 z_2 \cdots z_\eta$ denotes nodes outside these η subcubes). It is obvious that node *B* belongs to *CS*(1) (whose minterm representation is $y_1 y_2 \cdots y_\alpha z_1 z_2 \cdots z_\eta$, a contradiction due to our above assumption. □

ACKNOWLEDGMENTS

Hsing-Lung Chen's research was supported by the National Science Council of the Republic of China under Contracts NSC81-0408-E011-511 and NSC83-0408-E011-011. Nian-Feng Tzeng's research was supported in part by the U.S. National Science Foundation under Grants MIP-9201308 and CCR-9300075 and by the State of Louisiana under Contract LEQSF(1994-96)-RD-A-39. A preliminary version of this paper was presented at the Eighth International Parallel Processing Symposium, April 1994.

REFERENCES

[1] C.L. Seitz, "The Cosmic Cube," *Comm. ACM*, vol. 28, no. 1, pp. 22-33, Jan. 1985.

[2] J.C. Peterson et al., "The Mark III Hypercube-Ensemble Concurrent Computer," *Proc. 1985 Int'l Conf. Parallel Processing*, pp. 71-73, Aug. 1985.

[3] Intel Corporation, *iPSC/2 and iPSC/860 User's Guide.* Intel Corporation, June 1990.

[4] NCUBE Corporation, *n-CUBE 2 Processor Manual.* NCUBE Corporation, 1990.

[5] W.D. Hillis, *The Connection Machine.* Cambridge, Mass.: The MIT Press, 1985.

[6] B. Becker and H.-U. Simon, "How Robust is the *n*-Cube?," *Proc. IEEE 27th Symp. Foundations of Computer Science*, pp. 283-291, Oct. 1986.

[7] F. Ozguner and C. Aykanat, "A Reconfiguration Algorithm for Fault Tolerance in a Hypercube Multiprocessor," *Information Processing Letters*, vol. 29, pp. 247-254, Nov. 1988.

[8] M.A. Sridhar and C.S. Raghavendra, “On Finding Maximal Subcubes in Residual Hypercubes,” *Proc. Second IEEE Symp. Parallel and Distributed Processing*, pp. 870-873, Dec. 1990.

[9] S. Latifi, “Distributed Subcube Identification Algorithms for Reliable Hypercubes,” *Information Processing Letters*, vol. 38, pp. 315-321, June 1991.

[10] H.-L. Chen and N.-F. Tzeng, “Quick Determination of Subcubes in a Faulty Hypercube,” *Proc. 21st Int’l Conf. Parallel Processing*, vol. III, pp. 338-345, Aug. 1992.

[11] H.P. Katseff, “Incomplete Hypercubes,” *IEEE Trans. Computers*, vol. 37, no. 5, pp. 604-608, May 1988.

[12] N.-F. Tzeng and H. Kumar, “Analysis of Link Traffic in Incomplete Hypercubes,” *Proc. Fifth IEEE Symp. Parallel and Distributed Processing*, pp. 312-319, Dec. 1993.

[13] H. Sullivan and T.R. Bashkow, “A Large Scale Homogeneous, Fully Distributed Parallel Machine, vol. I,” *Proc. Fourth Symp. Computer Architecture*, pp. 105-117, Mar. 1977.

[14] J.R. Armstrong and F.G. Gray, “Fault Diagnosis in a Boolean *n*-Cube Array of Microprocessors,” *IEEE Trans. Computers*, vol. 30, no. 8, pp. 587-590, Aug. 1981.

[15] J.M. Gordon and Q.F. Stout, “Hypercube Message Routing in the Presence of Faults,” *Proc. Third Conf. Hypercube Concurrent Computers and Applications*, vol. I, pp. 318-327, Jan. 1988.

[16] M.-S. Chen and K.G. Shin, “Adaptive Fault-Tolerant Routing in Hypercube Multicomputers,” *IEEE Trans. Computers*, vol. 39, no. 12, pp. 1,406-1,416, Dec. 1990.

[17] P. Ramanathan and K.G. Shin, “Reliable Broadcast in Hypercube Multicomputers,” *IEEE Trans. Computers*, vol. 37, no. 12, pp. 1,654-1,657, Dec. 1988.

[18] L.M. Ni and P.K. McKinley, “A Survey of Wormhole Routing Techniques in Direct Networks,” *Computer*, vol. 26, no. 2, pp. 62-76, Feb. 1993.

[19] G. Lin and N.-F. Tzeng, “Effective Utilization of Hypercubes in the Presence of Faults,” *J. Parallel and Distributed Computing*, vol. 32, pp. 223-231, Feb. 1996.

[20] N.-F. Tzeng and H. Kumar, “Traffic Analysis and Simulation Performance of Incomplete Hypercubes,” *IEEE Trans. Parallel and Distributed Systems*, vol. 7, no. 7, pp. 740-754, July 1996.

[21] H.-L. Chen and N.-F. Tzeng, “Subcube Determination in Faulty Hypercubes,” *IEEE Trans. Computers*, vol. 46, no. 8, pp. 871-879, Aug. 1997.

[22] N.-F. Tzeng and G. Lin, “Efficient Determination of Maximum Incomplete Subcubes in Hypercubes with Faults,” *IEEE Trans. Computers*, vol. 45, no. 11, pp. 1,303-1,308, Nov. 1996.

Hsing-Lung Chen (S ’79-M ‘88) received the BS and MS degrees in computer science from National Chiao Tung University, Taiwan, in 1978 and 1980, respectively, and the PhD degree in electrical and computer engineering from the Illinois Institute of Technology, Chicago, in 1987.

From 1987-1989, he was an assistant professor in the Department of Mathematics and Computer Science at Clarkson University, Potsdam, New York. In 1989, he joined the Department of Electronic Engineering at the Taiwan Institute of Technology, Taipei, Taiwan, where he is currently a professor. His research interests include parallel processing, distributed computing, and database systems.

Dr. Chen is a member of the ACM and the IEEE.

Nian-Feng Tzeng (S ‘85-M ‘86-SM ‘92) received the BS degree in computer science from National Chiao Tung University, Taiwan, the MS degree in electrical engineering from National Taiwan University, Taiwan, and the PhD degree in computer science from the University of Illinois at Urbana-Champaign in 1978, 1980, and 1986, respectively.

He has been with the Center for Advanced Computer Studies at the University of Southwestern Louisiana (USL), Lafayette, since 1987. From 1986-1987, he was a member of the technical staff, AT&T Bell Laboratories, Columbus, Ohio. He is on the editorial board of *IEEE Transactions on Computers*, has served on program committees of several conferences, and is a distinguished visitor of the IEEE Computer Society. He was co-guest editor of a special issue of the *Journal of Parallel and Distributed Computing* on distributed shared memory systems, 1995, and the newsletter editor of the IEEE Technical Committee on Distributed Processing. His research interests include parallel and distributed processing, high-performance computer systems, high-speed networking, and fault-tolerant computing.

Dr. Tzeng is a member of Tau Beta Pi, a member of the ACM, a senior member of the IEEE, and the recipient of the outstanding paper award of the 10th International Conference on Distributed Computing Systems, 1990. He received the USL Foundation Distinguished Professor Award in 1997.