Subcube Determination in Faulty Hypercubes

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

Abstract—A hypercube may operate in a gracefully degraded manner, after faults arise, by supporting the execution of parallel algorithms in smaller fault-free subcubes. In order to reduce execution slowdown in a hypercube with given faults, it is essential to identify the maximum healthy subcubes in the faulty hypercube because the time for executing a parallel algorithm tends to depend on the dimension of the assigned subcube. This paper describes an efficient procedure capable of determining all maximum fault-free subcubes in a faulty hypercube. The procedure is a distributed one, since every healthy node next to a failed component performs the same procedure independently and concurrently. Based on interesting properties of faulty hypercubes, this procedure exhibits empirically polynomial time complexity with respect to the system dimension and the number of faults, for a practical range of dimensions. It compares favorably with prior methods when the number of faults is in the order of the system dimension. This procedure can deal with node failures and link failures uniformly and equally efficiently.

Index Terms—Faulty hypercubes, prime subcubes, product-of-sums, reconfiguration, sum-of-products, time complexity.

1 Introduction

The binary hypercube is a powerful and cost-effective topology for interconnecting a large number of computing nodes to constitute a message-passing parallel system for versatile applications. Hypercube systems have received considerable attention, and many such machines have been prototyped [1], [2] or marketed [3], [4], [5]. Most parallel programs developed for the hypercube can be executed on various system sizes [6], but they experience certain slowdowns on a small sized system. The extent of execution slowdown tends to grow as the system size decreases.

For a large system, the probability of faults arising is nonnegligible, and, over its mission duration, the system might involve one or several faults. In order to maintain the hypercube topology in the presence of faults, researchers have proposed the addition of spare nodes to hypercube designs [7], [8]. For these fault-tolerant hypercube designs, the system size remains unchanged after reconfiguration in response to an operational failure by replacing the failed component with a spare.

It is also possible for the hypercube to achieve fault-tolerance without employing spares by reconfiguring itself to a smaller sized system after faults occur, following the graceful degradation strategy. Under this strategy, performance degradation is kept to a minimum if the reconfigured system is a fault-free subcube with the maximum dimension possible. Several algorithms have been proposed for identifying the maximum healthy subcube in a hypercube with faults. In particular, an algorithm for maximum fault-free subcube identification was introduced by Ozguner and Aykanat [10]. This algorithm is centralized in nature, as it has to be run on a single processor (like the host or resource manager). Another centralized algorithm was presented in [11], with its time complexity being $O(nh^2)$ for an n-dimensional hypercube with h healthy nodes.

Distributed procedures for locating subcubes in a faulty hypercube are favorable, when compared with centralized counterparts, as they are more efficient and may prevent the host from becoming a bottleneck. A distributed procedure for finding the sizes of fault-free subcubes was given in [12]. However, the procedure is not guaranteed to always discover the maximum size of fault-free subcubes, and it doesn't identify the addresses of fault-free subcubes (which are needed for reconfiguration). Other distributed subcube identification algorithms have been developed recently [13] for an n-dimensional hypercube in which the number of faulty nodes is bounded above by $O(n)$. Such a heuristic algorithm is executed by each healthy node, and its time complexity at every involved node in an n-dimensional hypercube with m faulty nodes is $O(n^2m^2)$. These algorithms, however, could fail to identify the maximum fault-free subcube and are unsuitable for hypercubes with an accumulation of massive faults.

In this paper, we present a fast procedure for determining all maximum healthy subcubes in a faulty hypercube. This procedure is distributed in that the same algorithm is carried out by selected healthy nodes independently at the same time. It exhibits empirically polynomial time complexity with respect to the hypercube dimension and the number of faults, for hypercube systems with practical sizes. This is made possible by employing the novel concept of "reject regions" to eliminate unnecessary searches. For a given node and a set of faults, we can quickly determine all reject regions, which consist of those nodes *impossible* to be a part of any fault-free subcube containing the given node, and then arrive at an expression that specifies the collection of all healthy subcubes containing the given node efficiently and systematically. The expression so obtained is carefully manipulated with the aid of two simplification criteria to derive the addresses of healthy subcubes. This procedure is readily extended to deal with hypercubes with failed links, involving the same order of time complexity.

- *H.-L. Chen is with the Department of Electronic Engineering, National Taiwan Institute of Technology, Taipei, Taiwan, Republic of China.*
- *N.-F. Tzeng is with the Center for Advanced Computer Studies, University of Southwestern Louisiana, P.O. Box 44330, Lafayette, LA 70504. E-mail: tzeng@cacs.usl.edu.*

Manuscript received 15 Jan. 1993; revised 5 Sept. 1993.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number 104662.0.

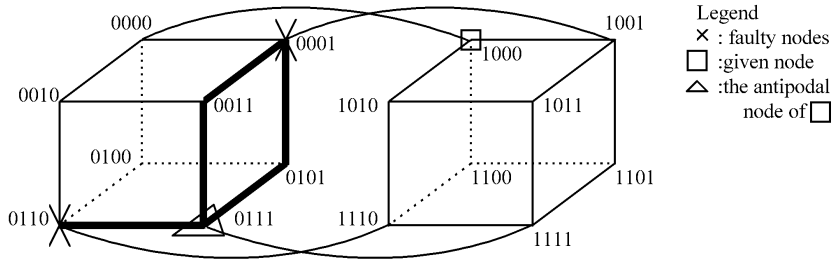

Fig. 1. A four-dimensional hypercube, H_4. (Some links are omitted for clarity, and reject regions are shown by bold lines).

This paper is organized as follows. Notation and relevant background are provided in Section 2. Section 3 addresses the methodology of finding subcubes containing a given node. Our subcube determination procedure is described in Section 4, and its extension is presented in Section 5. Section 6 concludes the paper.

2 Notation and Background

An n-dimensional hypercube, denoted by H_n, consists of 2^n nodes, each of which is labeled by an n-bit binary string, l_{n-1} l_{n-2} ... l_1l_0, with bit l_i corresponding to dimension i. Two nodes are connected by a link only if their labels differ in exactly one bit position. A four-dimensional hypercube is depicted in Fig. 1. Each subcube in H_n can be uniquely represented by a string of n symbols over the set {0, 1, *}, called its *address*, where * is a *don't care* symbol. Specifically, a k-dimensional subcube has exactly k *s in its address, as it involves a collection of 2^k cube nodes. As an example, nodes 0001, 0011, 0101, and 0111 in H_4 constitute a two-dimensional subcube addressed by 0**1, or, equivalently, $0^{*^2}1$, where α^c denotes c consecutive αs.

Let a subcube be 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 $\bar{b}_i$), if position i is 1 (or 0), and then dropping all *s. The operations on subcubes in a hypercube then can be performed elegantly, following a way similar to Boolean algebra. For example, subcube 0**1 is represented by $\bar{b}_3b_0$. The union of the three subcubes, 0**1, 01*0, and 00*0, is given by

$$\bar{b}_3b_0 + \bar{b}_3b_2\bar{b}_0 + \bar{b}_3\bar{b}_2\bar{b}_0 = \bar{b}_3b_0 + \bar{b}_3\bar{b}_0 = \bar{b}_3,$$

which is 0^{*^3}. As will become clear later, the use of a Boolean expression to specify the union of subcubes greatly facilitates our determination 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 two subcubes in H_4:

$$\bar{b}_3b_0 + \bar{b}_3b_2\bar{b}_0 = W.$$

The collection of all the nodes outside W is expressed by

$$\overline{W} = \left(b_3 + \bar{b}_0\right)\left(b_3 + \bar{b}_2 + b_0\right),$$

which is simplified to $b_3 + \bar{b}_2\bar{b}_0$, designating subcubes 1^{*^3} and *0*0, as expected.

3 Methodology for Finding Subcubes Containing a Given Node

The methodology introduced here will serve as the basis of our subcube determination procedure. Faults are assumed to occur only at cube nodes for easy presentation, with faulty links treated later on by extending the result. For a given healthy node in an injured hypercube, it is possible to systematically identify every fault-free subcube which contains the given node, by the use of interesting properties to be described below.

3.1 Basic Properties

Consider H_4 with faults at nodes 0001 and 0110, as shown in Fig. 1. For a given cube node, say 1000, we arrive at immediately two "regions" which can never contribute to any fault-free subcube containing the given node, with one region due to one fault. Such a region, called a *reject region*, is defined subsequently.

DEFINITION 1. *A* reject region *in a faulty hypercube is the smallest subcube which contains both a faulty node and the antipodal node of the given node.*

Let ⊗ denote 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. A reject region is addressed simply by performing operation ⊗ on the labels of a faulty node and the antipodal node. In Fig. 1, for example, the antipodal node of a given node 1000 is 0111, and the two reject regions are 0001 ⊗ 0111 = 0**1 and 0110 ⊗ 0111 = 011*, as indicated by bold lines. The concept of reject regions is critical to our methodology, and the property that a fault-free subcube involving the given node can never contain any node inside the reject regions, will be proved after the next definition.

DEFINITION 2. *A* prime subcube *with respect to a given node, say D, is a fault-free subcube which involves D but is not contained entirely in any other fault-free subcube involving D.*

Note that the prime subcube is defined with respect to a given node, and prime subcubes with respect to different nodes could be of different sizes. No proper subcube of a prime subcube can be a candidate largest subcube, so our attention is limited to only prime subcubes.

THEOREM 1. *A prime subcube contains no node inside reject regions.*

PROOF. Consider an n-dimensional hypercube H_n, in which node X is faulty. Without loss of generality, suppose

that the labels of node X and the given node are $0^{n-i}1^i$ and 0^n, respectively. The reject region involving node X is then $*^{n-i}1^i$. Assume that a node in a prime subcube, say node Y, is inside reject region $*^{n-i}1^i$. It will be illustrated that this assumption leads to a contradiction.

Without loss of generality, the label of node Y (which belongs to $*^{n-i}1^i$) is chosen to be $0^{n-j}1^j$, with $j \geq i$. The smallest subcube involving both the given node and node Y can be easily shown to be $0^{n-j}*^j$, which contains the faulty node X. Since the given node and node Y are also contained in a prime subcube (from Definition 2 and the above assumption), say PS, subcube $0^{n-j}*^j$, being the smallest subcube which involves the same two nodes, must be contained in prime subcube PS. As a result, PS involves faulty node X (for $X \in 0^{n-j}*^j$), contradicting definition of a prime subcube. □

THEOREM 2. *A node outside all reject regions is contained in at least one prime subcube.*

PROOF. (By contradiction.) Consider a given node in an n-dimensional faulty hypercube. Assume that a node outside all reject regions, say node A, is not contained in any prime subcube. Without loss of generality, suppose that the labels of node A and the given node are 0^i1^{n-i} and 0^n, respectively. Then, the smallest subcube involving both node A and the given node is addressed by 0^i*^{n-i}.

It can be argued that subcube 0^i*^{n-i} must involve at least one faulty node in the following. If 0^i*^{n-i} involves no faulty node, then consider the two cases:

1) There is no larger fault-free subcube containing the whole 0^i*^{n-i}, or
2) There is one such subcube.

In case 1), 0^i*^{n-i} itself is a prime subcube (as it is fault-free and involves the given node, from Definition 2). In case 2), a larger subcube containing 0^i*^{n-i} is a prime subcube. For either case, node A, which belongs to 0^i*^{n-i}, is contained in a prime subcube, violating our above assumption. Thus, 0^i*^{n-i} contains at least one faulty node, say node X.

Without loss of generality, suppose that the label of node X is 0^j1^{n-j}, with $j \geq i$. The reject region involving faulty node X is easily shown to be addressed by $*^j1^{n-j}$, which obviously contains node A (whose label is 0^i1^{n-i}). This is a contradiction, because node A should be outside all reject regions. □

It can be seen that any cube node outside the two reject regions shown in Fig. 1 is contained in at least one of the three prime subcubes: *0*0, **00, and $1*^3$. Theorems 1 and 2 reveal interesting properties of a faulty hypercube, and they make it possible to partition all the cube nodes into two *disjoint sets* with respect to a given node: the union of all reject regions and the union of all prime subcubes, denoted, respectively, by R and P. As a result, the fact that knowing R can directly get P holds valid for any faulty hypercube, and it is the fundamental idea behind our methodology. 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 defined earlier). A reject region which lies completely 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-products. From R, we immediately arrive at $P = \overline{R}$, which is in the form of product-of-sums (from DeMorgan's theorem). What remains to be solved now is to identify the largest subcube from P, since P contains all the prime subcubes (which are fault-free and contain the given node). To this end, we convert P from its product-of-sums form into a sum-of-products equivalence, which is then searched for the shortest product terms (i.e., minterms), because such a term corresponds to a largest subcube.

3.2 Generating Sum-of-Products Equivalence

The expression for P may be converted into its sum-of-products equivalence by using the distributive law: $x(y + z)(a + b) = xy(a + b) + xz(a + b)$. In the course of the conversion process, terms comprising the products of minterms (denoted by M_i) and maxterms (which refer to sum terms, denoted by S_i), are normally created and called the "genterms" (standing for generalized terms, denoted by G_i). The conversion process often involves lots of unnecessary effort due to generating undesirable minterms or the same minterms redundantly, unless care is taken to simplify the process. Consider a given node 110011 in H_6 with four faults: 010011, 101111, 100101, and 110110. Expression P for the given node is easily found to be

$$P = b_5\left(b_4 + \overline{b}_3 + \overline{b}_2\right)\left(b_4 + \overline{b}_2 + b_1\right)\left(\overline{b}_2 + b_0\right). \quad (1)$$

This expression is converted into its sum-of-products equivalence by applying the distributive law alone without simplification, as depicted in Fig. 2, where the result is the collection of the subcubes addressed by minterms given in the bottom level. In the first level of the conversion process, the distributive law is applied to sum term $(b_4 + \overline{b}_3 + \overline{b}_2)$, yielding three genterms, each due to one variable in the sum term, as indicated by an arrow with the variable shown next to it. It should be noted that all variables with subscript i in any expression P appear in the form of b_i or $\overline{b}_i$ exclusively; it is impossible to have b_i and $\overline{b}_i$ coexisting in P for any i, due to the way of calculating the addresses of reject regions.

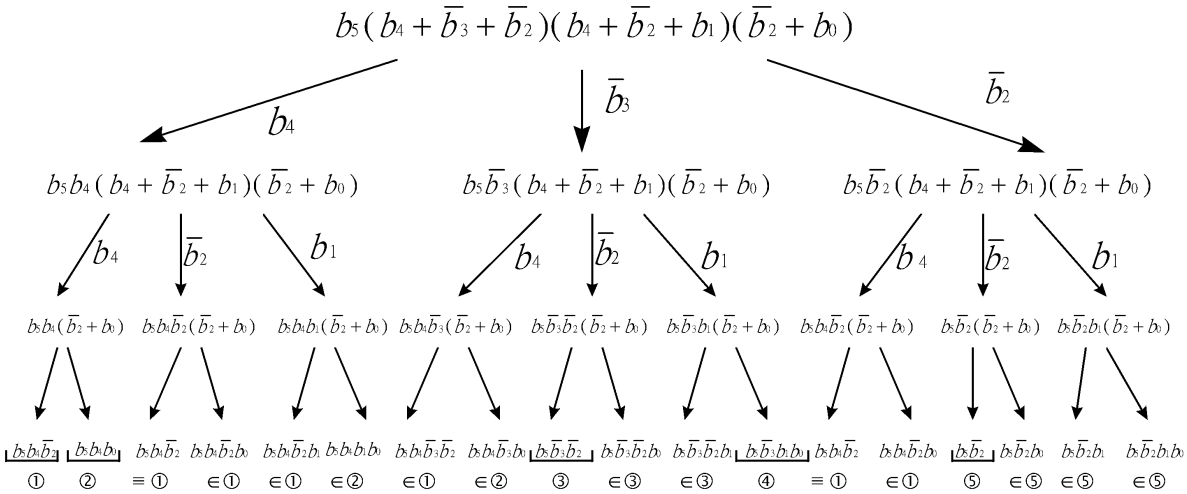

Fig. 2. Converting expression P into its sum-of-product equivalence. (Distinct minterms are numbered in the order of generation, with unnecessary minterms marked.)

Our goal for simplification is to prevent any unnecessary genterms from being created in the conversion process so that the minterms produced at the end all correspond to distinct subcubes, involving much less effort. The following simplification rules are useful:

1) $G_i + G_iG_j = G_i$, and
2) $b_jS_i = b_j$ (*or* $\bar{b}_jS_i = \bar{b}_j$), if sum term S_i contains variable b_j (or $\bar{b}_j$).

Rule 1) indicates that genterm G_iG_j should be removed. Rule 2) allows unnecessary sum term S_i to be eliminated, and, from this rule, one may easily arrive at the equation given below, after applying the distributive law to sum term $(b_j + b_k)$, provided that b_j is in sum term S_p but b_k is not, b_k is in sum term S_q but b_j is not, and neither b_j nor b_k is in sum term S_r:

$$\left(b_j + b_k\right)S_pS_qS_r = b_jS_qS_r + b_kS_pS_r.$$

This result implies that when the distributive law is applied to a sum term in a genterm, a variable in the sum term causes any other sum term containing the same variable to be discarded from the genterm produced due to the variable. In the above equation, for example, variable b_j produces genterm $b_jS_qS_r$, which involves no S_p as the variable is contained in S_p. This result is an essential criterion for eliminating unnecessary sum terms in produced genterms during a conversion process, referred to as *criterion I*.

Using Rule 1), the above equation is further simplified to $b_jS_qS_r + b_kS'_pS_r$, where S'_p is sum term S_p excluding variable b_j, because genterm $b_jb_kS_r$, which is contained in $b_kS_pS_r$, will be produced when the distributive law is applied to S_q of genterm $b_jS_qS_r$ (since $b_k \in S_q$). This reveals another important simplification criterion: After a genterm is created due to a variable in a sum term, this variable will be dropped from all genterms produced due to other variables in the same sum term, referred to as *criterion II*.

Making use of the above two simplification criteria, we obtain the conversion result of expression P (for H_6, given earlier), as shown in Fig. 3, where the set of variables to be discarded from a produced genterm is also given. In the first level of the conversion process, genterm $b_5b_4(\bar{b}_2 + b_0)$ is produced due to variable b_4 because sum term $(b_4 + \bar{b}_2 + b_1)$ is discarded (for it contains variable b_4), and genterm $b_5\bar{b}_3(\bar{b}_2 + b_1)(\bar{b}_2 + b_0)$ is produced after b_4 is dropped from sum term $(b_4 + \bar{b}_2 + b_1)$. Note that, if a genterm to be produced contains a sum term which consists of only discarded variables, the genterm is removed altogether. The conversion process, with the aid of the two simplification criteria, produces only distinct subcubes (see Fig. 3), saving lots of effort, and the set of distinct subcubes is identical to that depicted in Fig. 2 after unnecessary subcubes (which correspond to marked minterms) are removed.

An algorithm for generating the sum-of-products (SOP) equivalence of a given product-of-sums expression P is provided below, where a stack is employed to keep genterms produced during conversion. The stack holds only expression P, initially, and SOP starts with an empty set. After the algorithm ends, all minterms produced will be stored in SOP, which is then scanned to find the shortest ones.

Algorithm A: (generate a sum-of-products equivalence):

```
While (the stack is not empty)
  {
      Pop a genterm G_i from the stack;
      If (G_i has no sum terms)
            Append G_i to SOP;
      else {
            Apply the distributive law to the first sum
            term of G_i;
            Simplify the genterms produced using
            simplification criteria;
            Push all genterms after simplification into
```

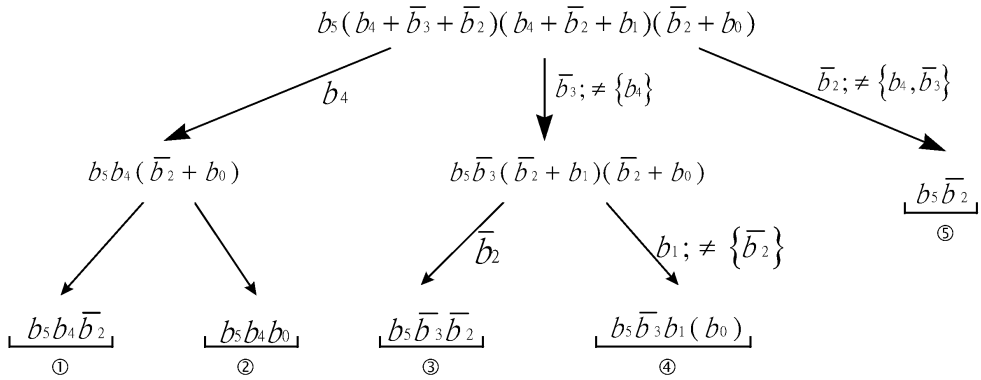

Fig. 3. Converting expression P with the aid of simplification criteria.

```
                the stack.
            }
        }
```

Since expression P specifies all subcubes containing a given node and the two simplification criteria remove only unnecessary subcubes, Algorithm A is ensured to generate *all* prime subcubes with respect to the given node. It is then needed to investigate time complexity involved in Algorithm A, i.e., the time complexity of finding the shortest minterm from P. Unfortunately, the problem of finding the shortest minterm from P is NP-complete in general, as stated in the next theorem, whose proof can be found in the Appendix.

THEOREM 3. *The shortest minterm determination (SMD) problem is NP-complete: Given a product-of-sums expression P for a faulty H_n, and a positive integer $k \le n$, determine if there is a minterm of length k or less, such that the minterm contains at least one variable in each sum term of P.*

While finding the largest prime subcube (i.e., shortest minterm) with respect to a given node in H_n is exponentially complex in terms of n in general, it is possible to devise an algorithm with polynomial time complexity for this problem, provided that n is restricted to a practical range, say $n \le 20$. This algorithm always finds the largest prime subcubes quickly by treating only promising genterms.

3.3 An Efficient Algorithm

The complexity of Algorithm A can often be reduced considerably by incorporating the greedy strategy as well as the "branch and bound" technique for identifying shortest minterms from a given P. A variable is said to "cover" a sum term if the variable belongs to the sum term. In order to arrive at the shortest minterm as soon as possible, the distributive law is not applied to an arbitrary constituent sum term (of a genterm G_i), but rather to a sum term which contains the variable covering the largest number of sum terms (in G_i). This causes a genterm consisting of fewest sum terms to be produced, and the genterm so produced is very likely to contain the shortest minterm. A solution is thus searched in a greedy manner this way.

To this end, G_i is scanned to identify a sum term containing the variable with the largest coverage (if there is a tie, a random one is chosen), and all the variables in the identified sum term are then rearranged in decreasing degrees of coverage. Consider expression P given in (1) as an example. Variable $\bar{b}_2$ has the largest coverage, and sum term $(b_4+\bar{b}_3+\bar{b}_2)$ is rearranged into $(\bar{b}_2+b_4+\bar{b}_3)$ before the distributive law is applied to it. The rearranged sum term is then placed ahead of the remaining sum terms, as illustrated in Fig. 4. In order to probe the genterm involving a solution as soon as possible, the genterms produced from a rearranged sum term are pushed to a stack in such an order that the genterm due to the variable with the smallest coverage is first, followed by that due to the variable with the next smallest coverage, and so on. This ensures that the first minterm probed is a shortest or near shortest one.

In addition, the shortest minterm obtained so far can serve as a bound to prevent longer minterms from being generated, avoiding unnecessary probes. The bound is compared with the number of variables existing in front of the first sum term (to which the distributive law is to be applied), called the *leading length*. For example, the leading length of the genterm at the top of Fig. 4 is 1, as there is only one variable (i.e., b_5) in front of sum terms. It is guaranteed to obtain all shortest minterms after every promising genterm is treated. The described greedy strategy and the branch-and-bound technique are incorporated in the subsequent algorithm. All shortest minterms produced so far are kept in set s_minterm_set, which is reset to $\varnothing$ initially and whenever a shorter minterm is found.

Algorithm B: (find all shortest minterms):

```
Reset s_minterm_set; s_minterm_length := dimension + 1;
/* length of the shortest minterm obtained so far */
While (the stack is not empty)
  {
    Pop a genterm G_i from the stack;
    Simplify G_i using the two simplification criteria;
    If (the leading length of G_i ≤ s_minterm_length)
        {
          if (G_i has no sum term)
            {
                if (the leading length of G_i <
                    s_minterm_length)
                    {
                        Reset s_minterm_set;
                        s_minterm_length := the (leading)
```

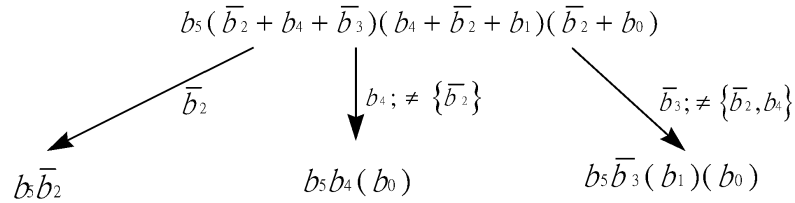

Fig. 4. Converting expression P by Algorithm B.

```
                                        length of G_i;
                    Add G_i to s_minterm_set;
                }
            else
                    Add G_i to s_minterm_set;
        }
    else
        {
        if (the leading length of G_i < s_minterm_length)
            {
            Select a variable with the largest cover-
            age in G_i and rearrange a sum term
            which contains the selected variable;
            Place the rearranged sum term ahead of
            the remaining sum terms;
            Apply the distributive law to the first
            (i.e., rearranged) sum term;
            Push genterms produced into the stack,
            according to the reverse order of the
            sequence at which they are produced.
            }
        }
    }
}
```

It should be noted that, since G_i is simplified immediately after it is popped from the stack, the genterms produced after applying the distributive law are pushed into the stack without simplification. This is correct, because minterms are created at the bottom level of probes, and no minterm requires simplification. The time complexity of Algorithm B is estimated in the following. For an n-dimensional hypercube with m faulty nodes, a sum term in any G_i contains no more than n variables, and the length of G_i is less than or equal to $n * m$. It is easy to derive that simplifying G_i requires time $O(nm)$, as does selecting a variable with the largest coverage. Rearranging a sum term takes $O(n \log_2 n)$. Unless m is very small, $n * m$ is greater than $n * \log_2 n$. As a result, time taken to treat a genterm popped from the stack equals $O(nm)$ in general

The overall complexity of this algorithm depends on the number of genterms created. Extensive simulation has been carried out to examine the number of genterms involved for every possible m in a given H_n and find out the worst case m. For each m, a sufficient number of fault patterns are produced using a random number generator and the average number of genterms is collected in an attempt to reflect the situation that faults happen randomly. As listed in Table 1, the row of genterms indicates the largest mean numbers of genterms produced among all possible ms for H_n, with n = 5, 10, 15, and 20 (it is beyond our computation power to investigate larger systems). The m value which leads to the largest number of genterms for each H_n is given in the next row. It can readily be observed that the number of genterms produced in the worst case scenario is bounded by $n * m$, for n up to 20. Algorithm B thus exhibits total time complexity $O(n^2m^2)$. It is faster than the methods described in [10], [11] when m is of O(n). This algorithm is exact, able to find all shortest minterms.

TABLE 1
THE MEAN NUMBER OF GENTERMS INVOLVED IN THE WORST CASE SCENARIO

Dimension (n)	5	10	15	20
Genterms	4	7	232	3,209
Faults (m)	3	3	59	568

4 SUBCUBE DETERMINATION

Algorithm B identifies every largest fault-free subcube which contains a given node efficiently. All maximum fault-free subcubes in a faulty hypercube can be determined fast using this algorithm, if an appropriate set of healthy nodes, referred to as *candidate nodes*, is selected and considered as given nodes. It should be noted that, if all healthy nodes are chosen as candidate nodes, the maximum healthy subcubes still can be obtained, but the overall time complexity involved will become unnecessarily high.

4.1 Candidate Nodes

The set of candidate nodes is chosen to be Φ = {node x | node x is healthy and immediately adjacent to a faulty node}. The largest subcubes with respect to each candidate node are identified by Algorithm B, and the (global) maximum fault-free subcubes in the faulty hypercube can be determined from all the largest subcubes so identified, as stated in the next theorem.

THEOREM 4. *Every maximum fault-free subcube belongs to the set of the largest subcubes derived using Algorithm B for each node in* Φ.

PROOF. Suppose that the set of the largest subcubes so derived does not contain a maximum fault-free subcube, say B_M. This means that B_M contains no candidate node in Φ. Consequently, all the neighbors of every node in B_M are fault-free, suggesting that a healthy subcube of the same size as B_M exists next to B_M. The two adjacent subcubes form a larger fault-free subcube, and, thus, B_M is not maximum, a contradiction. This completes the proof. □

For an n-dimensional hypercube with m faults, there are no more than $n * m$ candidate nodes in Φ. Since Algorithm B

takes $O(n^2m^2)$ for each candidate node in H_n, $n \le 20$, overall time complexity amounts to $O(n^3m^3)$. This subcube determination involves polynomial time complexity for practical hypercube systems.

4.2 Determination Procedure

The proposed approach lends itself perfectly to distributed determination by making every candidate node execute Algorithm B individually with respect to the node itself. It is assumed that the set of faulty nodes is identified in a distributed manner by the fault-free nodes, following the diagnostic algorithm introduced by Armstrong and Gray [14]. After fault diagnosis is done, every healthy node knows whether or not it is adjacent to a faulty node and should participate in subcube determination. The address of a faulty node is broadcast by a neighboring healthy node to all nodes, and only the candidate nodes keep such information. (Broadcasting in a faulty hypercube can be done efficiently according to the technique introduced by Lee and Hayes [15].)

On receiving the addresses of all faulty nodes, each candidate node finds the largest subcubes containing the node itself using Algorithm B. The addresses of the found subcubes are then sent to a designated node, at which all maximum fault-free subcubes are identified. It is clear that this determination procedure has the time complexity of $O(n^2m^2)$ on each candidate node.

Two other distributed subcube identification algorithms have been proposed [13], and they both require the same time complexity as ours at a participating node. Unlike our procedure, however, those two algorithms may fail to identify the maximum fault-free subcube and are carried out by all healthy nodes, not just a small fraction of them. On the other hand, if Algorithm B probes only the first minterm (instead of exhausting all promising genterms), our proposed procedure can locate the maximum fault-free subcube with probability 99.9 percent, which slightly exceeds what is attainable by the better one of the two prior distributed algorithms, for $m < n$, because it is found from our simulation that 99.9 percent of the first minterms identified by Algorithm B are indeed shortest ones, provided that the number of faults m is less than the system dimension n. In this situation, our procedure requires time complexity $O(nm^2)$, since the total number of genterms produced is always no more than m, as P involves at most m sum terms and the distributive law is applied to only the leftmost variable of a sum term. Consequently, our procedure not only needs lower time complexity than the two prior distributed algorithms (which require $O(n^2m^2)$) at each participating node, but also involves much fewer participants (i.e., no more than $n * m$).

5 Extension to Including Failed Links

Our fast subcube determination algorithm is extended to handle a hypercube in which both nodes and links could fail. Like a faulty node, a failed link makes several nodes become candidate nodes, based on which the largest fault-free subcubes are identified. Since a link connects two nodes, if both nodes connected by a failed link are healthy, an arbitrary one of the two nodes is selected as the node to dictate candidate nodes for the algorithm; otherwise, no node is selected (because the effect of the failed link is then considered by a faulty node to which the link connects). A selected node results in all its adjacent healthy nodes as candidate nodes.

All candidate nodes (whether due to faulty nodes or failed links) are treated uniformly by the algorithm, i.e., they are employed to determine reject regions. For each candidate node, the reject region due to a failed node is calculated in the same way as before (see Definition 1), but the reject region due to a failed link is to be defined next. Let a "preferred node" be the one of the two nodes connected by a failed link, such that it has a larger Hamming distance with respect to the candidate node (where the Hamming distance between a pair of nodes is the number of bits differing in the corresponding bits of the two node labels). The reject region due to the failed link is the smallest subcube which contains both a preferred node and the antipodal node of the candidate node, as will be proved in the subsequent theorem. Such a node is called a preferred node because it is closer to the antipodal node of the candidate node than the other node connected by the same link, and thus leads to a smaller reject region being created (this, in effect, signifies that the extent of size degradation is less due to a failed link than due to a faulty node).

THEOREM 5. *A prime subcube contains no node inside any reject region involving both a preferred node and the antipodal node of a candidate node.*

PROOF. Consider an n-dimensional hypercube, in which a link fails and node X_1 is the preferred node due to the failed link. Without loss of generality, suppose that the labels of node X_1 and the candidate node are $0^{n-i}1^i$ and 0^n, respectively. The reject region containing node X_1 is then $*^{n-i}1^i$. Let X_0 be the other node connected by the failed link. The labels of nodes X_0 and X_1 differ in exactly one bit, and the differing bit must be in the rightmost i positions (since the candidate node is 0_n, to which node X_0 is closer). The label of node X_0 is chosen $0^{n-i+1}1^{i-1}$ for convenience. Assume that a node in a prime subcube, say node Y, is inside reject region $*^{n-i}1^i$. This assumption will be shown to result in a contradiction.

Without loss of generality, let the label of node Y ($\in *^{n-i}1^i$) be $0^{n-j}1^j$, with $j \ge i$. The smallest subcube involving both the candidate node and node Y is $0^{n-j}*^j$, which obviously contains node X_0 and the preferred node X_1, implying that $0^{n-j}*^j$ contains the failed link. Since the candidate node and node Y are also involved in a prime subcube (from Definition 2 and the above assumption), say PS, subcube $0^{n-j}*^j$, being the smallest subcube which contain the same two nodes,

must be contained in prime subcube PS. As a result, PS involves the failed link, in contradiction to the definition of a prime subcube (see Definition 2). □

The preferred node is chosen based on a candidate node and is different from one candidate node to another. While a preferred node is fault-free, it serves to determine a region which is excluded from being a part of any prime subcube, just like a faulty node.

With reject regions due to failed nodes and links decided for a candidate node, one can immediately arrive at expression P, from which all largest fault-free subcubes are quickly identified using Algorithm B. Any maximum fault-free subcube in the faulty hypercube must belong to the set of the largest subcubes derived from all candidate nodes, as can be shown in a way similar to the proof of Theorem 4. For H_n, in which the amount of failed nodes and links combined is m, the number of candidate nodes involved in determining all maximum fault-free subcubes remains bounded by $O(nm)$, and the time complexity at each candidate node is $O(n^2m^2)$ as before, provided $n \leq 20$.

6 Conclusions

A fast procedure has been introduced for determining all maximum fault-free subcubes in a faulty hypercube. From interesting properties of faulty hypercubes, an expression P, which specifies all healthy subcubes containing a given node, is attainable directly, excluding reject regions from consideration to reduce search space. While identifying all largest subcubes specified by P is NP-complete in general, an efficient algorithm with the greedy strategy and the branch-and-bound technique incorporated has been determined experimentally to exhibit polynomial time complexity with respect to the number of faults (m) and the system dimension (n) for a practical range of n, i.e., $n \leq 20$. Our procedure selects no more than $n * m$ healthy nodes as candidate nodes, to which the efficient algorithm is applied individually to determine all largest prime subcubes. The time complexity at each candidate node is $O(n^2m^2)$, which is lower than those of the algorithms described in [10], [11] if m is of $O(n)$. The procedure is suitable for a hypercube with arbitrary node/link failures. With its low time complexity, this subcube determination procedure could be useful for systems designed to operate in a gracefully degraded manner after faults occur.

Appendix

Proof of Theorem 3

PROOF. A minterm is formed by taking exactly one variable from each sum term of P. The SMD problem is to find a smallest subset of variables such that every sum term of P contains at least one variable in the subset.

The "minimum cover" problem is known to be NP-complete [9]:.

Given a collection C of subsets of a finite set S, and a positive integer $k \leq |C|$. Does C contain a cover of size k or less for S, i.e., a subset $C' \subset C$ with $|C'| \leq k$, such that every element of S belongs to at least one member of C'?

To prove this theorem, it is sufficient to transform the minimum cover problem into the SMD problem. The transformation is defined as follows: An element in the finite set S is mapped to one sum term of P, and element $c_i \in C$ equals $\{x \mid x \in S$ *and* x is mapped to a sum term which contains the variable with index $i\}$. Considering expression P given in (1) as an example, if S_p and S_q are mapped, respectively, to $(b_4 + \bar{b}_3 + \bar{b}_2)$ and $(b_4 + \bar{b}_2 + b_1)$, then element c_4 in C consists of $\{S_p, S_q\}$, since they both contain variable b_4.

Under this transformation, the minimum cover problem is to determine if there is a subset of C such that the subset is of size no more than k, and every element of S belongs to (i.e., every sum term of P contains) at least one member of (i.e., one variable in) the subset, which is equivalent to the SMD problem. Therefore, any solution to the SMD problem can be trivially transformed into the solution for the minimum cover problem. This establishes the NP-completeness of the SMD problem. □

Acknowledgments

A preliminary version of this paper was presented at the 21st International Conference on Parallel Processing, August 1992. Hsing-Lung Chen was supported by the National Science Council of the Republic of China under Contract NSC81-0408-E011-511. Nian-Feng Tzeng 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(1992-94)-RD-A-32.

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] R. Arlauskas, "iPSC/2 System: A Second Generation Hypercube," *Proc. Third Conf. Hypercube Concurrent Computers and Applications*, vol. I, pp. 38-42, Jan. 1988.

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

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

[6] F.P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Comm. ACM*, vol. 24, pp. 300-309, May 1981.

[7] D.A. Rennels, "On Implementing Fault-Tolerance in Binary Hypercubes," *Proc. 16th Int'l Symp. Fault-Tolerant Computing*, pp. 344-349, July 1986.

[8] S.-C. Chau and A.L. Liestman, "A Proposal for a Fault-Tolerant Binary Hypercube Architecture," *Proc. 19th Int'l Symp. Fault-Tolerant Computing*, pp. 323-330, June 1989.

[9] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, p. 222. San Francisco: W.H. Freeman, 1979.

[10] 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.

[11] 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.

[12] B. Becker and H.-U. Simon, "How Robust Is the *n*-Cube?" *Proc. IEEE 27th Symp. Foundatons of Computer Science*, pp. 283-291, Oct. 1986.
[13] S. Latifi, "Distributed Subcube Identification Algorithms for Reliable Hypercubes," *Information Processing Letters*, vol. 38, pp. 315-321, June 1991.
[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] T.C. Lee and J.P. Hayes, "A Fault-Tolerant Communication Scheme for Hypercube Computers," *IEEE Trans. Computers*, vol. 41, no. 10, pp. 1,242-1,256, Oct. 1992.

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 to 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 of 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.

Dr. Tzeng has been with the Center for Advanced Computer Studies at the University of Southwestern Louisiana (USL), Lafayette, since June 1987. From 1986 to 1987, he was a member of the technical staff at AT&T Bell Laboratories, Columbus, Ohio. He is a member of 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 a co-guest editor of a special issue of the *Journal of Parallel and Distributed Computing* on distributed shared memory systems in 1995, and is 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 for the 10th International Conference on Distributed Computing Systems, May 1990. He received the USL Foundation Distinguished Professor Award in 1997.