Graceful Degradation in Algorithm-Based Fault Tolerant Multiprocessor Systems

Shalini Yajnik, *Member*, *IEEE*, and Niraj K. Jha, *Senior Member*, *IEEE*

Abstract—Algorithm-based fault tolerance (ABFT) is a technique which improves the reliability of a multiprocessor system by providing concurrent error detection and fault location capability to it. It encodes data at the system level and modifies the algorithm to operate on the encoded data in order to expose both transient and permanent faults in any processor. Work done till now in this area takes care of only the fault detection and location part of the problem. However, if spare processors are not available, then after a faulty processor has been located, the work initially assigned to it has to be mapped to some nonfaulty processors in the system in such a way that the fault tolerance capability of the system is still maintained with as small a degradation in performance as possible. In this paper, we propose an integrated deterministic solution to the above problem which combines concurrent error detection and fault location with graceful degradation. There exists no previous deterministic ABFT method for the design of general t-fault locating systems, even for the case of $t = 1$. We propose a general method for designing one-fault locating/s-fault detecting systems. We use an extended model for representing ABFT systems. This model considers the processors computing the checks to be a part of the ABFT system, so that faults in the check_computing processors can also be detected and located using a simple diagnosis algorithm, and the checks can be mapped to other nonfaulty processors in the system.

Index Terms—Algorithm-based fault tolerance, concurrent error detection, concurrent fault location, fault diagnosis, graceful degradation, transient faults.

1 Introduction

Multiprocessor architectures are commonly used for signal processing and other compute-intensive applications. These applications require high-speed data processing. A fault in the system can have damaging consequences on the result of a computation. Therefore, it is desirable to use on-line fault detection and location techniques which improve the reliability of the multiprocessor system. Algorithm-based fault tolerance (ABFT) is one such technique which provides concurrent error detection and location capability to the system [1]. It encodes data at the system level and modifies the algorithm to operate on the encoded data. The output data is also in an encoded form. This provides an inherent redundancy to the system which is useful in detecting and locating faults.

A considerable amount of work has been done on the design and analysis of ABFT systems [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. Banerjee and Abraham [4] presented a graph-theoretic model to represent ABFT systems. The model developed by them was later used by several researchers for designing [8], [9], [11], [12], [13] and analyzing [5], [8], [11], [13], [14] ABFT systems. Nair and Abraham [5] proposed a matrix-based model which was derived from the graph-theoretic model. This model has also been used in the design of ABFT systems. Banerjee and Abraham [4] also gave construction methods for ABFT systems, using the graph-theoretic model. Gu, Rosenkrantz, and Ravi [8] improved their methods by reducing the number of checking operations required for fault tolerance. Nair and Abraham [13] proposed a two-stage design methodology for designing ABFT systems. They proposed constructing unit systems with the desired characteristics first. Once the unit system had been designed, they proposed the construction of the composite ABFT system from the unit system. Vinnakota and Jha [12] proposed a concurrent error detection method which allowed sharing of data among different processors and used uniform checks.

Most earlier work in the design of general ABFT systems [11], [12], [13], with the exception of [9], assumes that the checking operations are performed by processors outside the system which are either fault-free or have some means of exposing their own faults, such as the self-checking property. This assumption makes the design process easier. However, the checking operations are usually an integral part of an ABFT system and, in many applications, are likely to be performed on the system processors. When the system processors on which they are computed fail, such checks become unreliable. Therefore, the accuracy of the computations is dependent on the reliability of the processors performing the checking operations as well. Banerjee and Abraham [10] introduced check evaluating nodes in their graph model and showed how to analyze such a system for fault tolerance. However, they did not consider the mapping of checks onto the system processors which compute them, such that the fault tolerance properties of the system could be preserved. In this paper, we use an extended graph-theoretic ABFT model which considers the check_computing processors to be a part of the ABFT system

- *S. Yajnik is with Lucent Technologies Bell Laboratories, Murray Hill, NJ 07974.*
- *N.K. Jha is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544. E-mail: jha@ee.princeton.edu.*

Manuscript received July 8, 1994.
For information on obtaining reprints of this article, please send e-mail to: transpds@computer.org, and reference IEEECS Log Number D95229.

and allows faults in these processors [6]. Our model removes a major weakness of the previous models which made the unrealistic assumption that checks are implemented on fault-free processors outside the system. We propose methods for designing ABFT systems with the desired fault tolerance capability under our extended ABFT model. The approach taken in [9] to attack this problem is different in the sense that a randomized algorithm is used to obtain an ABFT system which can inherently tolerate faults in processors computing the checks, by adding more checks than would be required if the checks were always assumed to be fault-free. In this paper, we take a direct and deterministic approach. Our method can actually be used in conjunction with the method in [9] to reduce the number of processors computing the checks.

Most of the work in the ABFT area takes care of only the fault detection and location part of the problem. If the system does not have any spare processors and a fault is located, the work which was initially assigned to the faulty processor has to be mapped to some other nonfaulty processors. This work proposes an integrated solution to the problem of fault detection, location, and graceful degradation in the ABFT area. This is the first deterministic approach for solving the above problem, though randomized techniques for the same exist [7]. Randomized techniques use a randomized algorithm for designing gracefully degrading ABFT systems and the final system has the required fault tolerance capability with a probability very close to one. Also, there exists no systematic deterministic approach for designing t-fault locating systems with bounded checks, even for $t = 1$, although randomized construction methods [9], with unbounded checks, are known. For a bounded check, the upper limit on the number of data elements checked by it is fixed, whereas for an unbounded check there is no such limit.

As assumed in the design of many concurrent error detection systems, we assume that permanent faults occur one at a time. Transient faults are known to constitute a very large fraction of the total faults occurring in a system [19]. However, we do not need to locate such faults, just detect them. Once these faults have been detected, a number of retries are done on the computation, after which only permanent faults are left in the system. Permanent faults are rare occurrences as compared to transient faults [19]. Therefore, it is reasonable to suppose that when a permanent fault occurs, there is sufficient time to detect and locate the presence of the fault before another permanent failure occurs. Hence, we assume that permanent faults occur one at a time. Our strategy is to design a one-fault locating/s-fault detecting system. This system will detect the presence of s or fewer faults in the system. Under our assumption, only one of these faults can be a permanent fault. Therefore, a number of retries on the computation should either remove all the faults (if all faults are transient) or should leave the permanent fault in the system. This fault can then be located by running the diagnosis algorithm on the output of the computation. In this paper, we first present a method for constructing one-fault locating/s-fault detecting systems under the extended ABFT model. We then propose a new diagnosis algorithm to locate the fault in the system. After the fault has been located, we distribute the work of the faulty processor to other processors in the system in such a way that the fault tolerance capability of the system is either maintained or slightly degraded with as small a loss in performance as possible.

The rest of the paper is organized as follows. In Section 2, we present a brief overview of the graph-theoretic and the matrix-based models. We also introduce our extended graph-theoretic model and give the necessary definitions and concepts. In Section 3, we give our design method for ABFT systems. In Section 4, we give the diagnosis algorithm for diagnosing a single fault in the system designed using our techniques. Section 5 describes the algorithm used for mapping the work of the faulty processor to other processors in the system such that the desired fault tolerance is maintained. Finally, in Section 6, we give the conclusions.

2 Preliminaries

In this section, we will describe the models used in representing and studying ABFT systems. We use the traditional processor-level fault model [1]. A fault in a processor is assumed to be manifested as an error in one or more data elements produced by it. The set of faulty processors in the system is said to be a *fault pattern*. The set of erroneous data elements is referred to as an *error pattern*. The set of data elements computed by a processor is referred to as its *data set*. A faulty processor can make any combination of data elements in its data set erroneous. Hence, a fault pattern can give rise to several error patterns.

Consider a multiprocessor system consisting of m processors $p_1, p_2, ..., p_m$. For a given algorithm, suppose these processors produce k data elements $d_1, d_2, ..., d_k$. Two or more processors in the system may share in the computation of a single data element. In addition to processors and data elements, an ABFT system consists of *checks* $c_1, c_2, ..., c_q$. System-level coding techniques, e.g., checksums, are used to encode the input data. The algorithm is modified to operate on this encoded data and the output is also in encoded form. The redundant part of the encoded output data is defined as the *check_data*. The checks make use of the redundancy in the computation to detect and locate faults in the system. The set of data elements checked by a check is called its *data set*. The checking operation involves computing some function of the data elements in the data set and comparing the result with a particular check_data element in the data set, called the *check_compare* element [6]. The processors which perform the checking operation are called the *check_computing* processors.

We assume that the checks in the system are bounded. A (g, h) check is defined on g data elements, and the result of the checking operation is as follows [10]:

- The check outputs a 0 if the check_computing processor is nonfaulty and all the data elements in its data set are error-free.
- The check outputs a 1 if the check_computing processor is nonfaulty and at least one data element in its data set is in error, but the number of data elements which are erroneous in its data set does not exceed h.
- The check is unpredictable if

1) The check_computing processor is faulty or
2) The number of erroneous elements in its data set is greater than h.

The state of the checks in the system can be represented by a q-bit binary vector where q is the number of checks in the system. The ith bit in the vector is "1" if check c_i evaluates to "1," else it is "0." This q-bit vector is usually called the *syndrome* [14]. If there are no faults in the system, the syndrome consists of all "0"s.

DEFINITION 1. *A system is said to be s-fault detecting if for every fault pattern of size s or less there is at least one check which evaluates to a* "1."

DEFINITION 2. *A system is said to be t-fault locating, if given that t or fewer processors in the system are faulty, the fault pattern can be uniquely determined from the syndrome.*

DEFINITION 3. *A system is said to be t-fault locating/s-fault detecting* ($s \geq t$)*, if in the presence of s or fewer faults, the system is able to detect the faults and, in the presence of t or fewer faults, the system is able to detect as well as locate the faults.*

2.1 The Extended Graph-Theoretic ABFT Model

This section describes the extended graph-theoretic model we introduced in [6] for representing ABFT systems. In the graph-theoretic model proposed earlier in [4], an ABFT system is represented by a tripartite graph called the **PDC** graph. The graph consists of three types of nodes, namely the processor nodes (P), the data nodes (D), and the check nodes (C). Methods for designing and analyzing general ABFT systems [4], [5], [8], [11], [12], [13], [14], [17], [18] using this model or the matrix-based model described in Section 2.2 assume that the check_computing processors are either fault-free or have some internal self-checking circuitry to expose their own faults. Our extended ABFT model, however, takes care of the faults in the check_computing processors too without assuming that they are self-checking. The model considers the check_computing processors to be a part of the system and any fault in these processors is also detected and located. According to the extended model, an ABFT system is represented by two graphs, a tripartite graph called the extended **PDC** graph and a bipartite graph called the *check evaluation* (**CE**) graph. The two graphs have six types of nodes [6].

- **Info_processor nodes** which perform useful computations.
- **Real_data nodes** which are the result of the useful computations.
- **Check nodes** which represent the checks.
- **Check_data nodes** which represent the redundant part of the encoded output data.
- **Code_processor nodes** which perform computations on the encoded input to produce the check_data nodes.
- **Check_computing processor nodes** which perform the checking operations. In general, these can be of the info_processor node type or of the code_processor node type.

The extended PDC graph consists of the info_processor nodes, the code_processor nodes, the real_data nodes, the check_data nodes, and the check nodes. All the processor nodes in the system form the set P. The real_data nodes and the check_data nodes form the set D and the check nodes form the set C. In the extended PDC graph, there is an edge between a processor node $p_i \in P$ and a data node $d_j \in D$, if d_j is in the data set of processor p_i. Similarly, there is an edge between a data node $d_j \in D$ and a check node $c_k \in C$, if d_j is checked by check c_k. Each code_processor node in the extended PDC graph is connected to one or more check_data elements. The code_processors do not share data elements with other processors in the system. Each info_processor node is connected to one or more real_data nodes. Two or more info_processor nodes can share data elements. The CE graph consists of two types of nodes, the check_computing processor nodes and the check nodes. There is an edge between a check node $c_i \in C$ and a check_computing processor node $p_j \in P$, if check c_i is implemented on processor p_j. Each check in the system has at least one check_data element in its data set. Each check_data element is assumed to be computed on one and only one code_processor. For the purpose of the example given next, we present the following definitions [1].

DEFINITION 4. *A column checksum matrix* $A'((n+1) \times m)$ *of a matrix* $A(n \times m)$ *consists of the matrix A in the first n rows and the* ith, $1 \leq i \leq m$, *element of the* $(n+1)$th *row consists of the summation of the elements in the* ith *column in matrix A.*

DEFINITION 5. *A row checksum matrix* $A'(n \times (m+1))$ *of a matrix* $A(n \times m)$ *consists of the matrix A in the first m columns and the* ith, $1 \leq i \leq n$, *element of the* $(m+1)$th *column consists of the summation of the elements in the* ith *row in matrix A.*

EXAMPLE 1. Consider the multiplication of two 3×3 matrices A and B. Suppose the matrix multiplication is to be made one-fault locating / three-fault detecting. A is encoded using column checksums and B is encoded using row checksums, as shown below. The last row of encoded matrix A' and the last column of the encoded matrix B' constitute the redundant input data. The matrix multiplication result is a 4×4 matrix C whose real_data elements are $d_1, d_2, ..., d_9$. The last column and the last row of C are the redundant output data elements. These seven elements, $d_{10}, d_{11}, ..., d_{16}$, are the check_data elements. Suppose this matrix multiplication is performed on a 4×4 mesh array. Processors $p_1, p_2, ..., p_9$, perform the useful computation to produce $d_1, d_2, ..., d_9$, whereas processors $p_{10}, p_{11}, ..., p_{16}$, perform the redundant computations and produce the check_data elements. Checks $c_1, c_2, ..., c_7$, give the whole system one-fault locatability / three-fault detectability [1].

$$A' = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ \sum_i a_{i1} & \sum_i a_{i2} & \sum_i a_{i3} \end{pmatrix} \quad B' = \begin{pmatrix} b_{11} & b_{12} & b_{13} & \sum_j b_{1j} \\ b_{21} & b_{22} & b_{23} & \sum_j b_{2j} \\ b_{31} & b_{32} & b_{33} & \sum_j b_{3j} \end{pmatrix}$$

$$C = A' \times B' = \begin{pmatrix} d_1 & d_2 & d_3 & d_{10} \\ d_4 & d_5 & d_6 & d_{11} \\ d_7 & d_8 & d_9 & d_{12} \\ d_{13} & d_{14} & d_{15} & d_{16} \end{pmatrix}$$

Info_processor nodes	$p_1, p_2, ..., p_9$
Code_processor nodes	$p_{10}, p_{11}, ..., p_{16}$
Real_data nodes	$d_1, d_2, ..., d_9$
Check_data nodes	$d_{10}, d_{11}, ..., d_{16}$
Check nodes	$c_1, c_2, ..., c_7$

Checks

$c_1 : d_{10} = d_1 + d_2 + d_3$?
$c_2 : d_{11} = d_4 + d_5 + d_6$?
$c_3 : d_{12} = d_7 + d_8 + d_9$?
$c_4 : d_{16} = d_{13} + d_{14} + d_{15}$?
$c_5 : d_{13} = d_1 + d_4 + d_7$?
$c_6 : d_{14} = d_2 + d_5 + d_8$?
$c_7 : d_{15} = d_3 + d_6 + d_9$?

In this example, d_{10} is the check_compare element of check c_1, d_{11} is the check_compare element of check c_2, and so on. The original non-fault-tolerant *PD* graph and the corresponding extended PDC graph are shown in Fig. 1. We, of course, still need to find a mapping of checks to processors such that the one-fault locatability / three-fault detectability of the system is maintained. A general method for finding such a mapping is given in Section 3.3.

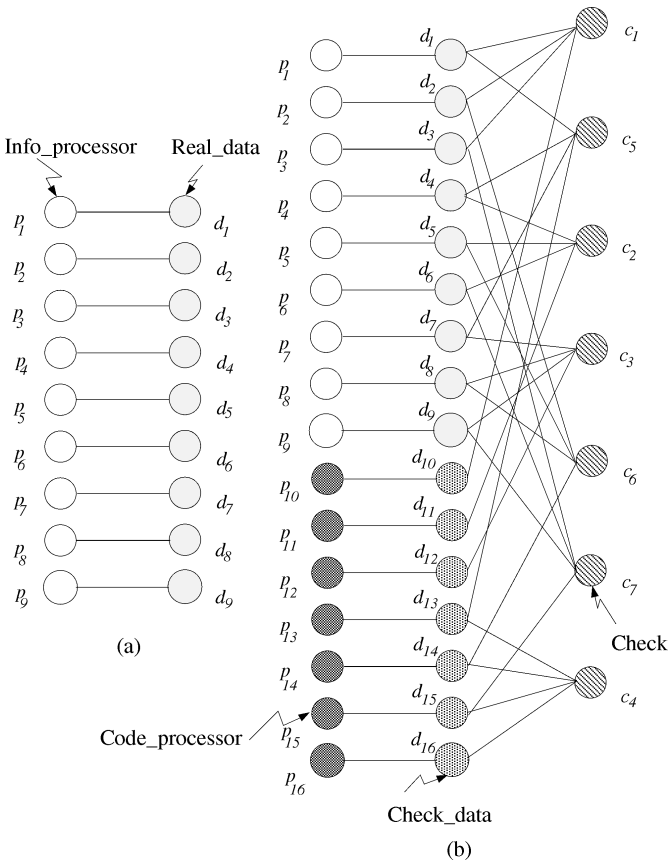

Fig. 1. (a) The original *PD* graph, (b) the one-fault locating/three-fault detecting PDC graph.

2.2 The Matrix-Based Model

The matrix-based model for representing ABFT systems was proposed in [5]. The model represents an ABFT system by a set of matrices. It is derived from the graph-theoretic model. The PDC graph is divided into two parts, the *PD* graph and the *DC* graph. These graphs are represented by *PD* and *DC* matrices, respectively. The *PD* matrix has processors as rows and data elements as columns. There is a "1" entry in the (i, j) position in the matrix if there is an edge between processor p_i and data element d_j in the *PD* graph. The *DC* matrix has rows which correspond to data elements in the system and columns which correspond to checks. The matrix has a "1" in the (i, j) position if data element d_i is checked by check c_j. The *PC* matrix is defined as the product of the *PD* and *DC* matrices. An entry in the (i, j) position in this matrix indicates the number of paths from processor p_i to check c_j in the PDC graph. It is clear that the matrix-based model can also be derived from our extended PDC graph in a straightforward manner. We give here a few definitions from [5], [14].

DEFINITION 6. *^rPD is defined as the matrix whose rows are formed by adding r different rows of matrix PD, for all possible combinations of r rows, and setting all nonzero entries in the resulting matrix to* 1.

DEFINITION 7. *^rPC is the matrix obtained by the product of the matrices ^rPD and DC.*

A row in the matrix ^rPD represents a unique set of r processors and the set of "1" entries in the row represents the union of the data sets of these processors. The ijth entry in the ^rPC matrix represents the number of data elements of the corresponding r processors that are checked by the jth column check.

3 Design for One-Fault Location/s-Fault Detection

A procedure for designing one-fault locating / s-fault detecting systems has to consider the fact that a single fault pattern can give rise to several error patterns. Nair and Abraham [5] proposed a two-stage design procedure. They introduced the concept of a *unit system*. A unit system is a system in which each processor has only one data element in its data set. The *PD* matrix is an identity matrix and the *DC* matrix is designed for the desired fault detectability / locatability. The unit systems are extended to form the desired final *composite* ABFT system with the same detectability / locatability. The procedure given in [5] cannot be used if the processors share data elements. Vinnakota and Jha [12] gave a procedure which allowed sharing of data elements among processors for fault-detecting systems. We propose a three-stage design procedure which allows sharing of data elements among processors as well as allows the check_computing processors in the system to be faulty. A one-fault locating / s-fault detecting unit system is first designed, assuming checks are fault-free. Its copies are created and the final composite system is obtained by superposition of the original and the copies. However, this is not enough if the check_computing processors are also allowed to be faulty. We also have to design the CE graph which shows the mapping of checks to check_computing processors. The methods given ahead can be used to form the unit system when the checks are assumed to be bounded. The method used in [8], [9] can also be used for

constructing the unit systems, when unbounded checks are assumed.

If only a single fault is present in the system, then for locating the single fault, it is necessary that each syndrome produced by the fault be distinct from the syndromes produced by other single faults in the system. If there is sharing of data elements among processors, the system cannot always locate faults in individual processors [5]. This can be seen as follows. Suppose two or more processors in the system share in the computation of data element d_k. Suppose one of these processors, p_i, develops a fault which manifests itself as an error in only d_k, and no other data element computed by it is erroneous. Faults in an ABFT system are located indirectly through errors in the data elements. In this case, when only d_k is in error, it cannot be said whether p_i or some other processor which shares in the computation of d_k is faulty. Hence, the system cannot always locate faults in individual processors.

In most systems, processors have nondisjoint data sets. In our extended ABFT model, only the info_processors in the system can share data elements. The set of processors which share one or more data elements is defined as a *processor class* [5]. If only a shared data element is found to be in error, we conservatively declare all the processors involved in the computation of that data element to be faulty. However, if at least one of the erroneous data elements in the data set of a processor class is not a shared one, then the faulty processor can be pinpointed exactly. Therefore, in general, a one-fault locating system can only guarantee fault resolution to one processor class in the presence of shared data elements, not necessarily to individual processors. Henceforth, in this paper when we refer to locatability, we mean processor class locatability.

In Section 3.1 and Section 3.2, we design the unit system and the composite system assuming that the checks in the system are fault-free. In Section 3.3, we take care of the possibility of failures in the check_computing processors.

3.1 Unit System Construction

The real_data elements in the original *PD* graph are distributed among the template unit system and its copies. The method of distributing the real_data among the unit systems is shown later in Section 3.2. In this section, we assume that the distribution has already been done. We are given a template *PD* graph consisting of m processors and m real_data elements. Each info_processor in the unit system is connected to exactly one real_data element. A fault in a processor of the unit system means an error in the corresponding data element. Therefore, the cardinality of the fault patterns will be the same as the cardinality of the resulting error patterns. In this case, designing one-fault locating/s-fault detecting systems will be the same as designing one-error locating/s-error detecting systems. We can take advantage of the methods proposed by Rosenkrantz and Ravi in [11] for this purpose. The resulting system has to satisfy the constraint that each check should have at least one check_data element in its data set. The methods given in [11] have to be modified to introduce check_data elements in the *DC* graph. For each check in the system, we add a check_data element and a code_processor to the unit system. Each code_processor is attached to only one check_data element in the unit system. Most practical ABFT schemes, such as those based on the commonly used checksum codes, utilize (g, 1) checks. So we will also be using only (g, 1) checks in our system design. Such a check may be invalidated if there is more than a single erroneous data element in its data set. Let n be the total number of data elements in the unit system. In such systems, the number of real_data elements is equal to the number of info_processor nodes m and the number of check_data elements is equal to the number of checks q. Only one check_data element can be mapped to one code_processor. So the number of code_processors introduced in a unit system is equal to the number of checks. Therefore, $n = m + q$.

Next, we present Lemma 1, which is a variation of a lemma given in [9]. The proof of the lemma can be derived in a straightforward way from the proof in [9].

LEMMA 1. *Any unit system which is s-fault locating in the presence of s or fewer faults is also $2s$-fault detecting in the presence of $2s$ or fewer faults.*

LEMMA 2. *Suppose a collection of checks T in a unit system, satisfies the following four conditions:*

1) *Each real_data element is checked by two checks in T.*
2) *Each check_data element is checked by exactly one check in T.*
3) *For each pair of checks in T, at most one data element is checked by both the checks.*
4) *Each check in T checks only one data element not checked by any other check.*

Then T is a valid set of checks for locating a single fault or detecting two faults in the unit system.

PROOF. In a unit system, a single fault will result in a single erroneous data element. Suppose a real_data element d_i is erroneous. The real_data element is checked by two checks which will go to "1"in the syndrome. All the other checks in the system will be at "0" since they do not contain any erroneous data elements in their data sets. By Condition 3 in the lemma, the two checks have only one data element in common which is d_i. Therefore, we will be able to locate the erroneous data element d_i by taking the intersection of the data sets of the two checks at "1." Now, suppose a check_data element is in error. By Condition 2, a check_data element is checked by only one check and that check goes to "1" in the syndrome, with all the other checks being "0." By Condition 4, a check has only one data element in its data set which is not checked by any other check. Therefore, we can locate the erroneous data element and hence the system is one-fault locating. If there are two faults in the unit system, though the checks in the system will not be able to locate any fault, they have the ability to detect the presence of the two faults. This can be proved as a direct result of Lemma 1. Therefore, the above system is one-fault locating/two-fault detecting. □

Let α be the maximum number of processors in any processor class. We do not want any two processors in a processor class to be checked by the same check. If we al-

low that, then some copy of the template unit system may have two data elements of the same processor being checked by the same check, which might result in check invalidation. Our diagnosis algorithm for one-fault location is based on the fact that there is no check invalidation due to finite error detectability of a check if only one fault occurs in the system.

Consider a square grid G of size $n \times n$ or a rectangular grid of size $(n + 1) \times n$. Number the rows in the grid as 0, 1, ..., $n - 1$, n, and similarly the columns from 0 to $n - 1$. The jth diagonal in the grid is defined as containing the positions $(i, (i + j) \bmod n)$, $0 \le i \le n - 1$, of the grid. The following theorems give the method for construction of one-fault locating / s-fault detecting unit systems. If $\alpha \le \lfloor \sqrt{m} \rfloor$, choose g such that $m \le (g-1)^2$. The conditions in Theorem 2 are satisfied and the design procedure follows the steps given in the proof of the theorem. If $\alpha > \lfloor \sqrt{m} \rfloor$, choose g such that $\alpha \le \left\lfloor \frac{m}{(g-1)^2} \right\rfloor (g-1)$. This value of g also satisfies the condition $m \ge (g-1)^2$. Therefore, Theorem 1 can be used for the design. Similar arguments hold for the use of subsequent theorems in this paper.

THEOREM 1. *For* $s = 2$, $\alpha \le \left\lfloor \frac{m}{(g-1)^2} \right\rfloor (g-1)$, $g > 2$, $m > (g-1)^2$, *the number of checks sufficient for constructing a one-fault locating/s-fault detecting unit system is* $\left\lceil \frac{m}{g-1} \right\rceil + \left\lceil \frac{m}{(g-1)^2} \right\rceil (g-1)$.

PROOF AND METHOD OF UNIT SYSTEM CONSTRUCTION. Let A = set of real_data elements in the unit system, where $|A| = m > (g-1)^2$. Let B = set of check_data elements in the unit system, where $|B| = q = \left\lceil \frac{m}{g-1} \right\rceil + \left\lceil \frac{m}{(g-1)^2} \right\rceil (g-1)$. Let $m = x(g-1)^2 + y$, where x and y are integers, such that $x = \left\lfloor \frac{m}{(g-1)^2} \right\rfloor$ and $y < (g-1)^2$. Arrange the set A of real_data elements in grids $A_1, A_2, ..., A_x$, of size $(g-1) \times (g-1)$ each. Arrange the rest of the y elements in a grid A_{x+1} of $(g-1)$ columns. Place the real_data elements corresponding to a processor class either along the diagonals in one grid or in different grids. Once a processor class occupies some positions in a diagonal, the unused positions can be used by another processor class. Construct **row checks** by taking all real_data elements of a row in A_i, $1 \le i \le x + 1$, and a single unused element from the check_data set B. Construct **column checks** by taking an unused element from B and all elements of a column from A_i, $1 \le i \le x + 1$. There will be g elements in each check except the checks corresponding to grid A_{x+1} which may have less than g elements. In the above construction procedure, the data elements corresponding to a processor class are placed along the diagonals in a grid or in separate grids because no two diagonal entries or entries from separate grids can be in the same check's data set. The size of the diagonals in grids $A_1, A_2, ..., A_x$, is $(g-1)$ and the number of grids of size $(g-1) \times (g-1)$ is x. Therefore, the number of elements in a processor class which can be handled by the above construction procedure is $x(g-1)$. The set of checks generated by the above method of construction has the properties given in Lemma 2. Therefore, the resulting system is one-fault locating / two-fault detecting. □

EXAMPLE 2. Let $m = 14$, $g = 4$. Then

$$|B| = \left\lceil \frac{m}{g-1} \right\rceil + \left\lceil \frac{m}{(g-1)^2} \right\rceil (g-1) = 11$$

and $|A| = 14$. Let $A = [a_1, a_2, ..., a_{14}]$ and $B = [b_1, b_2, ..., b_{11}]$. Suppose processors p_1, p_4, and p_8 computing a_1, a_4, and a_8, respectively, are in one processor class P_1, and processors p_2, p_6, and p_7 computing a_2, a_6, and a_7, respectively, are in another processor class P_2. The remaining real_data elements a_3, a_5, a_9, ..., a_{14} are computed by single processors. Hence, processors p_3, p_5, p_9, ..., p_{14}, do not have to form processor classes. The data elements corresponding to P_1 are placed along diagonal 0, and the data elements corresponding to P_2 are placed along diagonal 1 in grid A_1.

Arrange set A in two grids, A_1 of size 3×3 and A_2 of size 2×3, as follows:

$$A_1 : \begin{matrix} a_1 & a_2 & a_3 \\ a_5 & a_4 & a_6 \\ a_7 & a_9 & a_8 \end{matrix} \qquad A_2 : \begin{matrix} a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & \end{matrix}$$

Row checks	**Column checks**
$r_1 = [a_1, a_2, a_3, b_1]$	$c_1 = [a_1, a_5, a_7, b_6]$
$r_2 = [a_5, a_4, a_6, b_2]$	$c_2 = [a_2, a_4, a_9, b_7]$
$r_3 = [a_7, a_9, a_8, b_3]$	$c_3 = [a_3, a_6, a_8, b_8]$
$r_4 = [a_{10}, a_{11}, a_{12}, b_4]$	$c_4 = [a_{10}, a_{13}, b_9]$
$r_5 = [a_{13}, a_{14}, b_5]$	$c_5 = [a_{11}, a_{14}, b_{10}]$
	$c_6 = [a_{12}, b_{11}]$

THEOREM 2 *For* $s = 2$, $\alpha \le \lfloor \sqrt{m} \rfloor$, $g > 2$, $m \le (g-1)^2$, *the number of checks sufficient for constructing a one-fault locating/s-fault detecting unit system is* $\lceil 2\sqrt{m} \rceil$.

PROOF AND METHOD OF UNIT SYSTEM CONSTRUCTION. Let A = set of real_data elements in the unit system, where $|A| = m \le (g-1)^2$. Let B = set of check_data elements, where $|B| = \lceil 2\sqrt{m} \rceil$. If $m \le \lceil \sqrt{m} \rceil \times \lfloor \sqrt{m} \rfloor$, then construct a grid of elements of A in a $\lceil \sqrt{m} \rceil \times \lfloor \sqrt{m} \rfloor$ fashion, else construct a grid in a $\lceil \sqrt{m} \rceil \times \lceil \sqrt{m} \rceil$ fashion. The real_data elements in A corresponding to processors in one processor class should lie along the diagonals in the grid so that no two data elements in a processor class are checked by the same check. Since the size of the diagonals is $\lfloor \sqrt{m} \rfloor$ or $\lceil \sqrt{m} \rceil$ depending on the grid, the maximum number of processors in a

processor class which this construction procedure can handle is $\lfloor\sqrt{m}\rfloor$ or $\lceil\sqrt{m}\rceil$, respectively. Construct **row checks** by taking all the elements of a row in *A* and an unused element from *B*. There are $\lceil\sqrt{m}\rceil$ such checks. Each of these checks will have less than or equal to g data elements. Construct **column checks** with all the elements of a column and an unused element from *B*. Each of these checks will also have less than or equal to g data elements in them. The set of checks generated by this method also has the properties given in Lemma 2. Therefore, the unit system is one-fault locating/two-fault detecting. □

It was shown in [11] that at least $\lceil\frac{2n}{g+1}\rceil$ checks are required to detect two errors in n data elements. For our calculations, $n = m + q$, where q is the number of checks in the system. Therefore, the lower bound on the number of checks for detecting two errors evaluates to $\lceil\frac{2m}{g-1}\rceil$. The construction procedure given in Theorem 1 adds $\lceil\frac{m}{g-1}\rceil + \lceil\frac{m}{(g-1)^2}\rceil(g-1)$ checks to the system. If m is a multiple of $(g-1)^2$, the number of checks adheres to the known lower bound. The value of the lower bound in [11] has been calculated without any consideration for fault location in the presence of processor classes. Our construction procedures may have to add a few extra checks for one-fault location in the presence of processor classes.

THEOREM 3. *For $s = 3$, $g > 2$, $m \geq (g-1)^2$, $\alpha \leq \lfloor\frac{m}{(g-1)^2}\rfloor(g-1)$ the number of checks sufficient for the construction of one-fault locating/three-fault detecting unit system is* $\lceil\frac{m}{g-1}\rceil + g\lceil\frac{m}{(g-1)^2}\rceil$.

PROOF AND METHOD OF UNIT SYSTEM CONSTRUCTION. Let *B* be the set of $\lceil\frac{m}{g-1}\rceil + g\lceil\frac{m}{(g-1)^2}\rceil$ check_data elements. Let *A* be the set of the real_data elements in the unit system. $|A| = m$. Let $m = x(g-1)^2 + y$, where x, y are integers and $x = \lfloor\frac{m}{(g-1)^2}\rfloor$ and $y < (g-1)^2$. Divide *A* into x sets A_1, A_2, ..., A_x, with $(g-1)^2$ real_data elements each and set A_{x+1} with y data elements. Construct a grid of data elements for each set A_i, $1 \leq i \leq x+1$, with $(g-1)$ columns. The grids will have $g-1$ rows, except the grid corresponding to set A_{x+1}. Place the real_data elements corresponding to a processor class either along a diagonal in the grid or in different grids so that no two elements in a processor class are checked by the same check. Introduce $g-1$ unused check_data elements from the set *B* into each grid and place them as the last row in the grids. Call these appended sets A_i', $1 \leq i \leq x+1$. Construct **row checks** with a row of A_i', $1 \leq i \leq x+1$, and an unused element of *B*. The number of row checks is $\lceil\frac{m}{g-1}\rceil + \lceil\frac{m}{(g-1)^2}\rceil$. Construct **column checks** with all elements from each column in the grid corresponding to A_i', $1 \leq i \leq x+1$. There are $(g-1)\lceil\frac{m}{(g-1)^2}\rceil$ column checks. The total number of checks is thus $\lceil\frac{m}{g-1}\rceil + g\lceil\frac{m}{(g-1)^2}\rceil$. The resulting set of checks satisfies the conditions of Lemma 2 and, hence, will detect two faults or locate one fault. Now assume that three faults are present. If all the three faults are in *B*, then there will be at least one row/column check which will contain exactly one erroneous data element from *B*. Therefore, the faults will be caught. If *A* has a single fault and *B* has two faults, then a fault in *B* will be caught by a row/column check which contains no erroneous data element. If *A* has two faults in different rows, then at least one of the faults will be caught by a row check which contains an error-free data element from *B*. If *A* has both the faults in the same row, there will be at least one column check which will detect the fault. If *A* has all the three faults, then there will always be at least one row or column check that contains exactly one erroneous data element. □

The known lower bound for detecting three errors in a set of n data elements was computed in [11]. They established a lower bound of $\lceil\frac{1}{g}(2n - \lfloor\frac{n}{g}\rfloor)\rceil$, where $n = m + q$. This evaluates to a lower bound of $\frac{mg}{(g-1)^2} + \frac{m}{(g-1)}$. The number of checks required by the construction procedure in the above theorem is very close to this known lower bound and when m is a multiple of $(g-1)$, the number of checks is equal to the lower bound.

EXAMPLE 3. Let $m = 27$, $s = 3$, $g = 5$. Hence, the number of checks $= \lceil\frac{m}{g-1}\rceil + g\lceil\frac{m}{(g-1)^2}\rceil = 17$. Let real_data set $= [a_1, a_2, ..., a_{27}]$ and check_data set $= [b_1, b_2, ..., b_{17}]$. Let

$$A_1' = [a_1, a_2, \ldots, a_{16}, b_1, b_2, b_3, b_4],$$
$$A_2' = [a_{17}, a_{18}, \ldots, a_{27}, b_5, b_6, b_7, b_8].$$

A_1' and A_2' arranged in grids with four columns are shown below.

$$A_1' = \begin{matrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} \\ b_1 & b_2 & b_3 & b_4 \end{matrix} \qquad A_2' = \begin{matrix} a_{17} & a_{18} & a_{19} & a_{20} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{25} & a_{26} & a_{27} & \\ b_5 & b_6 & b_7 & b_8 \end{matrix}$$

Checks:

Row Checks

$r_1 = [a_1, a_2, a_3, a_4, b_9]$
$r_2 = [a_5, a_6, a_7, a_8, b_{10}]$
$r_3 = [a_9, a_{10}, a_{11}, a_{12}, b_{11}]$
$r_4 = [a_{13}, a_{14}, a_{15}, a_{16}, b_{12}]$
$r_5 = [b_1, b_2, b_3, b_4, b_{13}]$
$r_6 = [a_{17}, a_{18}, a_{19}, a_{20}, b_{14}]$
$r_7 = [a_{21}, a_{22}, a_{23}, a_{24}, b_{15}]$
$r_8 = [a_{25}, a_{26}, a_{27}, b_{16}]$
$r_9 = [b_5, b_6, b_7, b_8, b_{17}]$

Column Checks

$c_1 = [a_1, a_5, a_9, a_{13}, b_1]$
$c_2 = [a_2, a_6, a_{10}, a_{14}, b_2]$
$c_3 = [a_3, a_7, a_{11}, a_{15}, b_3]$
$c_4 = [a_4, a_8, a_{12}, a_{16}, b_4]$
$c_5 = [a_{17}, a_{21}, a_{25}, b_5]$
$c_6 = [a_{18}, a_{22}, a_{26}, b_6]$
$c_7 = [a_{19}, a_{23}, a_{27}, b_7]$
$c_8 = [a_{20}, a_{24}, b_8]$

For the case of $s = 3$ and $m < (g-1)^2$, the method will be same as the method based on Theorem 2. Row and column checks are constructed from a grid of

$$\left(\lceil\sqrt{m}\rceil + 1\right) \times \lceil\sqrt{m}\rceil \text{ or } \left(\lceil\sqrt{m}\rceil + 1\right) \times \lfloor\sqrt{m}\rfloor$$

elements as before, except that in this case a row of check_data elements is added to the grid, which accounts for the extra row in the grid. So, the number of checks is $(\lceil 2\sqrt{m}\rceil + 1)$. The one extra check comes from the row of check_data elements added to the grid.

For $s \geq 4$, we will use a hierarchical construction method. An analysis of the hierarchical approach has been given before in [17]. In all previous methods, we have assumed that each check has a distinct check_compare element, so that the number of check_data elements was equal to the number of checks. But in the next procedure, since we are using hierarchical construction methods, the checks can share the same check_compare element. So the number of check_data elements will no longer be equal to the number of checks. Before going into the construction procedure, a theorem from [11] is presented here. This will be used in the design method.

THEOREM 4. *Let $s \geq 4$ and $r = \lceil \log_2(\frac{s}{3}) \rceil$. Given a set I containing g^{r+2} data elements, $(r+2)g^{r+1} - g^r$ checks are sufficient to detect s faults in I.*

THEOREM 5. *For $s \geq 4$, $r = \lceil \log_2(\frac{s}{3}) \rceil$, $m = (g-1)^{r+2}$, the number of checks sufficient for the unit system construction is $(r+2)g^{r+1} - g^r$.*

PROOF. Let A be the set of all the $(g-1)^{r+2}$ real_data elements in the unit system and B be the set of all the check_data elements. The number of check_data elements required, as will be shown later, is $g^{r+2} - (g-1)^{r+2}$. Therefore, the total number of data elements is g^{r+2}. Then by Theorem 4, the number of checks sufficient for the system is $(r+2)g^{r+1} - g^r$. □

METHOD OF UNIT SYSTEM CONSTRUCTION. We will base our construction procedure on the method given in [11].

1) Partition the set A into $(g-1)^r$ sets $A_1, A_2, \ldots, A_{(g-1)^r}$, each containing $(g-1)^2$ data elements. For each of the sets, we need to detect the presence of three faults or locate a single fault.
2) By Theorem 3, a set A_i needs $(2g-1)$ checks to detect three faults or locate one fault. Add $(2g-1)$ distinct check_data elements from B to each set A_i, $1 \leq i \leq (g-1)^r$. Let us term these augmented sets as A_{1i}, $1 \leq i \leq (g-1)^r$. Each set A_{1i} has g^2 data elements.
3) Construct row and column checks for each A_{1i} using the method described in Theorem 3. There will be $(2g-1)(g-1)^r$ such checks.
4) Construct B', a subset of B, by removing all the $(2g-1)(g-1)^r$ check_data elements used in Step 2. So $|B'| = g^{r+2} - g^2(g-1)^r$.
5) Partition the set B' into $[g^r - (g-1)^r]$ sets

$$B_{11}, B_{12}, \ldots, B_{1\left[g^r-(g-1)^r\right]},$$

of g^2 elements each. For each of these sets too, we need to detect three faults or locate one fault. But these sets already have the check_data elements needed for construction of the checks.
6) Construct the row and column checks for each set B_{1j}, $1 \leq j \leq [g^r - (g-1)^r]$. There are $(2g-1)(g^r - (g-1)^r)$ such checks. We define the checks constructed in Step 3 and Step 6 as *basic checks*.
7) For $k = 1$ to r, do the following:
   a) Group together $(g-1)$ of the A_{ki} sets and one B_{kj} set. Let us define these groups as the *A-groups*. There will be $(g-1)^{r-k}$ such groups.
   b) The B_{kj} sets, not used in the earlier step, are grouped together, with each group having g sets. We term these groups as the *B-groups*.
   c) Number the data elements in each of the A_{ki} and B_{kj} sets from 1 to g^{k+1}.
   d) Construct *position checks* T_{ij}, for group j, by taking the ith element from each set in the group. Since each set has g^{k+1} data elements, there will be g^{k+1} such checks for each group.
   e) Combine all the g sets in an A-group j, $1 \leq j \leq (g-1)^{r-k}$, to form one set $A_{(k+1)j}$. Each of these sets will have g^{k+2} data elements.
   f) Combine all the g sets in a B-group j, to form one set $B_{(k+1)j}$. Each of these sets, too, will have g^{k+2} data elements. □

The basic checks constructed for each group of g^2 data elements, in Steps 3 and 6, provide the group one-fault location / three-fault detection capability. Position checks are then added to increase the fault detection capability to s. The addition of the position checks does not violate the results of the basic checks. Therefore, even after the addition of the position checks, the unit system is still one-fault locating.

CLAIM. *The number of check_data elements sufficient for the unit system is $g^{r+2} - (g-1)^{r+2}$.*

PROOF OF CLAIM. If we look at the construction procedure in the top-down manner as shown in Fig. 2, we see that at every stage, we partition a given set into $(g-1)$ sets of the same size and add a set of check_data elements to this group of $(g-1)$ sets. In the last stage, each real_data set has $(g-1)^2$ data elements. There are $(g-1)^r$ such sets. To each of these sets, we add $(2g-1)$ check_data elements to construct the basic checks.

The number of check_data elements required by the real_data sets for the basic checks = $(2g-1)(g-1)^r$. Now each real_data set, after being augmented by the $(2g-1)$ check_data elements, has g^2 data elements. Therefore, the number of check_data elements required for position checks for each group of $(g-1)$ sets = g^2. If we do this for every stage, we get the following:

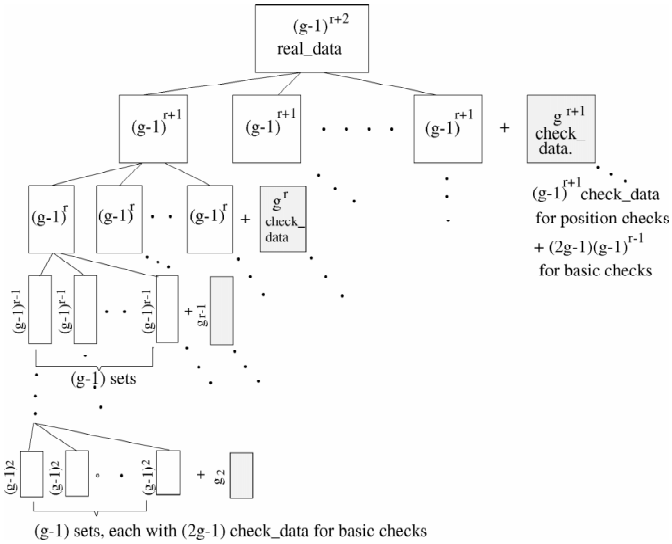

Fig. 2. Hierarchical construction.

$$
\begin{aligned}
&\text{Number of check\_ data elements}\\
&= (2g-1)(g-1)^r + (g-1)^{r-1}g^2\\
&\quad + (g-1)^{r-2}g^3 + \cdots + (g-1)g^r + g^{r+1}\\
&= (2g-1)(g-1)^r + g^{r+1}\left[\frac{1-\left(\frac{g-1}{g}\right)^r}{1-\frac{g-1}{g}}\right]\\
&= (2g-1)(g-1)^r + g^2\left[g^r - (g-1)^r\right]\\
&= g^{r+2} - (g-1)^{r+2}
\end{aligned}
$$

□

EXAMPLE 4. Let $m = 10^4$, $g = 11$, $s = 12$, $r = 2$. Then $|A| = (11-1)^4 = 10{,}000$ and $|B| = 11^4 - 10^4 = 4{,}641$. Partition A into 100 sets $A_1, A_2, \ldots, A_{100}$ of size 100 each. We require that each of these sets should have three-fault detectability. Add 21 check_data elements from B to each A_i, $1 \le i \le 100$, to form A_{1i} as shown in Stage 1 in Fig. 3. Then $|B'| = 4{,}641 - 2{,}100 = 2{,}541$. Partition B' into 21 sets, $B_{11}, B_{12}, \ldots, B_{1(21)}$ of size 121 each. For each of these sets, construct the basic checks. There will be $121 \times 21 = 2{,}541$ such checks. Fig. 3 shows the total number of checks in Stage 1.

Group the A_{1i} and B_{1j} sets into 11 groups. The groupings are given in Stage 2 of Fig. 3. Construct position checks for each of the groups. Each group will have 121 position checks. So the number of position checks = $121 \times 11 = 1{,}331$. Now consider each group as a set of 1,331 elements. Group all the 11 groups together and construct position checks, as given in Stage 3 of Fig. 3. There are 1,331 such checks. Therefore, the total number of checks is 2,541 + 1,331 + 1,331 = 5,203 which is also equal to $(r+2)g^{r+1} - g^r = 4 \times 11^3 - 11^2$.

There are no known tight lower bounds for the number of bounded checks required to detect four or more errors in the data elements. The trivial lower bound for detecting s faults is $\left\lceil \frac{2m}{g-1} \right\rceil$. The number of checks used in the above construction procedure is within a factor $O(\log_2 s)$ of this trivial lower bound. In [11], a construction procedure is proposed for $s \ge 4$. Our construction procedure is based on their hierarchical method and hence we achieve the same number of checks as required by their method.

For the case of $s \ge 4$, $g > 2$, $r = \left\lceil \log_2\left(\frac{s}{3}\right) \right\rceil$, $m > (g-1)^{r+2}$, partition the set of real_data elements into sets containing $(g-1)^{r+2}$ data elements (the last one may contain less than $(g-1)^{r+2}$ data elements). For each set, we can use the above construction procedure.

THEOREM 6. *For $s \ge 4$, $r = \left\lceil \log_2\left(\frac{s}{3}\right) \right\rceil$, $m < (g-1)^{r+2}$, the number of checks sufficient for the unit system construction is $r(x + \lceil 2\sqrt{x} \rceil + 1)g^{r-1} + (\lceil 2\sqrt{x} \rceil + 1)g^r$, where $x = \frac{m}{(g-1)^r}$.*

PROOF. Consider a unit system with m real_data elements, where $m < (g-1)^{r+2}$. We use the same construction procedure as used for $m = (g-1)^{r+2}$. At each stage, the set A of real_data elements is divided into $(g-1)$ subsets. After r stages, there will be $x \le (g-1)^2$ real_data elements in each of the $(g-1)^r$ subsets of A. To each of these, we need to add $\lceil 2\sqrt{x} \rceil + 1$ check_data elements to construct basic checks for detecting three faults or locating one fault. Therefore, each real_data subset will have $a = x + \lceil 2\sqrt{x} \rceil + 1$ data elements. The number of basic checks for the real_data sets is $(\lceil 2\sqrt{x} \rceil + 1)(g-1)^r$. We group each of these subsets into groups of $(g-1)$ subsets each. So, there will be $a(g-1)^{r-1}$ position checks at this stage.

$$
\begin{aligned}
&\text{Number of position checks for the real\_ data sets}\\
&= a(g-1)^{r-1} + ag(g-1)^{r-2} + ag^2(g-1)^{r-3} + \cdots + ag^{r-1}\\
&= a\left[g^r - (g-1)^r\right]
\end{aligned}
$$

$$
\begin{aligned}
&\text{Number of check\_ data required}\\
&= (\text{no. of basic checks} + \text{no. of position checks})\\
&\quad \text{for the real data}\\
&= \left(\lceil 2\sqrt{x} \rceil + 1\right)(g-1)^r + \left(x + \lceil 2\sqrt{x} \rceil + 1\right)\left[g^r - (g-1)^r\right]\\
&= \left(x + \lceil 2\sqrt{x} \rceil + 1\right)g^r - x(g-1)^r
\end{aligned}
$$

$$
\begin{aligned}
&\text{Number of data elements in the system}\\
&= \text{Number of real\_ data elements}\\
&\quad + \text{Number of check\_ data elements}\\
&= m + \left(x + \lceil 2\sqrt{x} \rceil + 1\right)g^r - x(g-1)^r\\
&= x(g-1)^r + \left(x + \lceil 2\sqrt{x} \rceil + 1\right)g^r - x(g-1)^r\\
&= \left(x + \lceil 2\sqrt{x} \rceil + 1\right)g^r
\end{aligned}
$$

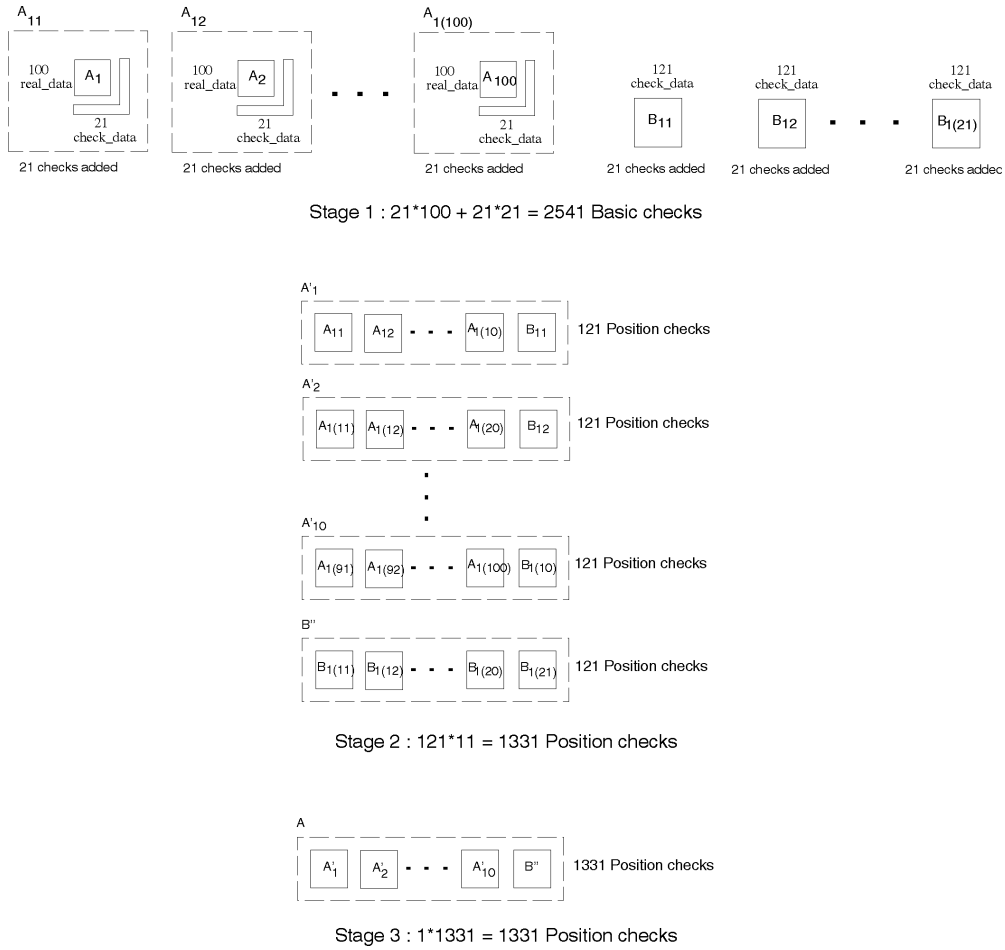

Fig. 3. Hierarchical construction in Example 4.

We use the same construction procedure given in the earlier theorem. We can see that each data element, both real_data and check_data, is in one position check at each stage. There are r stages in the construction and each position check has a maximum of g data elements in its data set. Therefore, there are $r(x+\lceil 2\sqrt{x}\rceil+1)g^{r-1}$ position checks in the system. After the rth stage, we have g^r subsets of $(x+\lceil 2\sqrt{x}\rceil+1)$ elements each. So there are $(\lceil 2\sqrt{x}\rceil+1)g^r$ basic checks in the system. Therefore, the total number of checks in the system is

$$r\left(x+\lceil 2\sqrt{x}\rceil+1\right)g^{r-1}+\left(\lceil 2\sqrt{x}\rceil+1\right)g^r. \qquad \square$$

3.2 Composite System Construction

It is assumed here, as in [12], that the processors in the original non-fault-tolerant system can share data elements, but each processor produces at least one data element which no other processor in the system affects. We will refer to such data elements as the *distinguishing data* elements of the processor. For designing the complete ABFT system, the unit template system has to be constructed first. From the unit template system, copies are created and then these systems are combined to form the composite ABFT system using some rules. The composition method is based on the design process given in [12]. Before going into the design process, we state a definition from [12].

DEFINITION 8. *A data element is defined as filled if it has been added to or used in the tripartite graph of the present or a previous unit system, else it is called unfilled.*

Before describing the formal procedure for unit system and composite system construction, let us take a look at an example system.

EXAMPLE 5. Consider the implementation of some algorithm on a parallel system with the *PD* graph as shown in Fig. 4a. We wish to design a fault-tolerant version of the system which is one-fault locating/two-fault detecting. A one-fault locating/two-fault detecting template unit system for the *PD* graph is shown in Fig. 4b. The value of g has been chosen to be 3. In order to create the template unit system, we choose one distinguishing data element for each processor. Data elements d_1, d_3, d_4, and d_5 are chosen to be in the template unit system. Therefore, the real_data set of the template system is $\{d_1, d_3, d_4, d_5\}$. By Theorem 1, we require four checks to locate a single fault in the template system. Therefore, we need to add four check_data nodes $\{d_6, d_7, d_8, d_9\}$, and, consequently, four code_processor nodes p_5, p_6, p_7, and p_8 to the

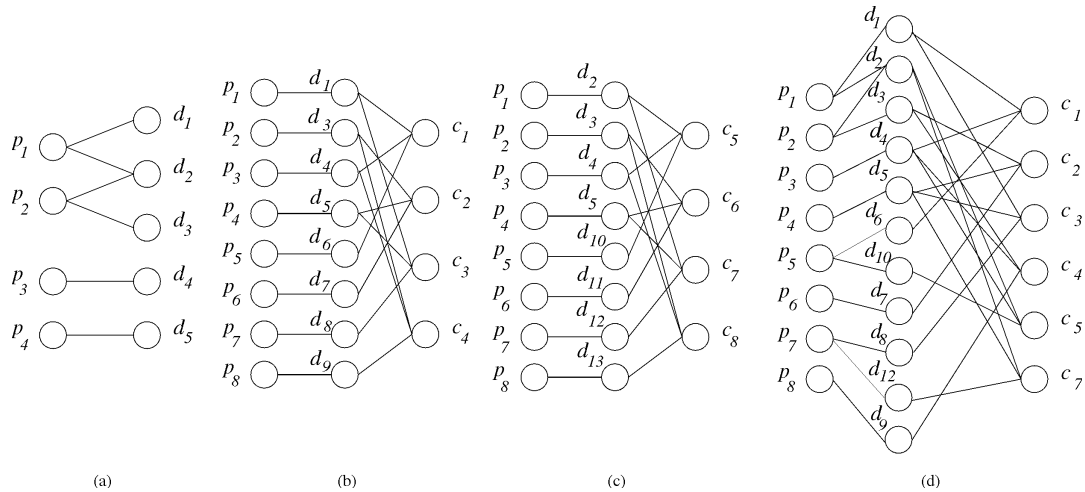

Fig. 4. (a) The original *PD* graph, (b) a one-fault locating/two-fault detecting template unit system, (c) a copy of the template system, (d) the composite system.

graph. Processors p_1 and p_2 are in a processor class. Therefore, during the unit system construction, the data elements d_1 and d_3 are placed along the diagonal in the grid. Now, once the template unit system has been created, data element d_2 has not been used anywhere in the template system. Therefore, we have to create a copy that will use this unfilled data element. Processor p_1 will use unfilled data element d_2 in its data set. Other processors p_2, p_3, and p_4 do not have any unfilled data elements left to use. Therefore, they reuse their already filled distinguishing data elements. The real_data set of the copy is $\{d_2, d_3, d_4, d_5\}$, as shown in Fig. 4c. The template system and the copy are now composed to form the final system which is shown in Fig. 4d. The data set of each info_processor is its data set in the original system. The data set of each code_processor is the union of its data sets in each of the unit systems. Two checks which have identical real_data elements in their data sets are merged to form a single check. In the final system, check c_6 is equivalent to check c_2 and check c_8 is equivalent to check c_4. So checks c_6 and c_8 have been removed.

The design process is discussed next. Given the original nonfault tolerant *PD* graph:

1) Construct the *PD* part of the template unit system, where each processor is connected to exactly one of its distinguishing data elements. If more than one distinguishing data element exist, then one can be chosen at random.
2) Construct copies of the template unit system such that all the unfilled data elements in the system are filled. If there are no unfilled data elements left for a processor, then reuse one of its distinguishing data elements. For details of this part the readers are referred to [12].
3) Using the methods given in Section 3.1, design the *DC* part of the template unit system such that it has one-fault locatability/*s*-fault detectability. Add check_data nodes, code_processor nodes, and check nodes to the template unit *PD* graph.
4) In the copies of the template, add the code_processor nodes, check_data nodes, and check nodes with these edges connected in the same way as in the unit template system. The check nodes and the check_data nodes in the copies are numbered differently though.

Given the template system and its copies, the composite system is formed by superimposing the unit systems, as follows:

1) The data sets of the info_processor nodes are the same as in the original nonfault tolerant *PD* graph. The data set of a code_processor is the union of the data sets of that processor in all the unit systems.
2) While merging the *DC* part of the graphs, the set of checks to which a data element is connected is formed by taking the union of the set of checks to which it is connected in each of the unit systems. After this, the checks whose data sets (excluding the check_compare element) are found to be identical are merged into a single check, and if the check_compare element of the removed check is not connected to any other check, it is removed from the composite system.

The proof of validity of this procedure is given in [12]. The following theorem from [12] is stated here.

THEOREM 7. *If the template system is t-fault locating/s-fault detecting, the composition of the template system and the copies is also t-fault locating/s-fault detecting.*

3.3 Check Mapping

We are given a composite system which is one-fault locating/*s*-fault detecting, assuming that the checks are fault-free. We now need to map checks to the processors in such a way as to maintain one-fault locatability/*s*-fault detectability of the system even when the checks may themselves fail. If the check_computing processors in the system are allowed to be faulty, then a fault pattern is *detectable* if and only if for every error pattern that the fault generates, there is at least one nonfaulty processor evaluating a check which has a single erroneous data element in its data set. We will

henceforth use both the matrix model and the graph model for system description and design. We assume that all the processors in the system have the capability to perform the checking operations. We give some definitions and theorems from [5].

DEFINITION 9. *A row of ${}^{r}PC$ is said to be completely detectable if and only if the fault represented by the row is detectable for all possible error patterns produced by that fault.*

THEOREM 8. *A system is s-fault detecting if and only if the matrices ${}^{i}PC$, for $i = 1, 2, ..., s$, are completely detectable, i.e., all the rows of each of the matrices are completely detectable.*

For a row of ^{r}PC to be detectable, there should be at least one entry in the row which is less than or equal to the error-detectability (h) of the check used, and the corresponding check should be evaluated on a processor which is not in the set of faulty processors defined by the row. This should be true for all the rows of each of the matrices ${}^{i}PC$, for $i = 1, 2, ..., s$, for a system to be s-fault detecting. We define an entry in the ${}^{r}PC$ matrix as *true* if the check corresponding to the column of the entry is not evaluated on any processor representing the row. Since we want to take care of all the possible error patterns that a fault pattern can generate, we consider the *PC* matrices of each of the unit systems, instead of operating on the *PC* matrix of the final composite system. If the *PC* matrices of the unit systems are considered, the information regarding the sharing of data elements is lost. Therefore, instead of working with the *PC* matrices of the unit systems directly, we modify these matrices to preserve this information. For the info_processor part of the system, we take the *PD* matrix of the original nonfault tolerant system, but consider only those columns whose corresponding data elements are present in the unit system. The *DC* matrix has rows corresponding to the data elements in the unit system and columns corresponding to all the checks in the composite system. The columns whose checks are not in the unit system have all "0" entries. In the *DC* matrix, a check in the unit system which is equivalent to a check in another unit system is represented by their corresponding check in the composite system. The augmented *PC* matrix of the unit system is then obtained by multiplying the *PD* and the *DC* matrices. Hereafter, when we refer to the *PC* matrix of a unit system, we mean the augmented *PC* matrix. The *PC* matrices of the unit systems in Fig. 4 are given in Fig. 5.

Before giving the algorithm for mapping of checks, we present some more definitions and theorems from [5], adapted to our needs. This adaptation takes into account the fact that the error detectability of checks cannot be exceeded in a one-fault locating system designed by our method if only one processor fault occurs in the system.

DEFINITION 10. *Rows R_1 and R_2 of matrix PC are said to have a* $0 - 1$ *disagreement if there is at least one true* "0" *in either R_1 or R_2, such that the other row has a true* "1" *in the corresponding position.*

DEFINITION 11. *If all pairs of rows of matrix PC have a* $0 - 1$ *disagreement, then PC is said to have a* $0 - 1$ *disagreement with itself.*

	c_1	c_2	c_3	c_4	c_5	c_7
p_1	1	0	1	0	0	0
p_2	0	1	0	1	0	0
p_3	1	0	0	1	0	0
p_4	0	1	1	0	0	0
p_5	1	0	0	0	0	0
p_6	0	1	0	0	0	0
p_7	0	0	1	0	0	0
p_8	0	0	0	1	0	0

(a)

	c_1	c_2	c_3	c_4	c_5	c_7
p_1	0	0	0	0	1	1
p_2	0	1	0	1	1	1
p_3	0	0	0	1	1	0
p_4	0	1	0	0	0	1
p_5	0	0	0	0	1	0
p_6	0	1	0	0	0	0
p_7	0	0	0	0	0	1
p_8	0	0	0	1	0	0

(b)

Fig. 5. (a) *PC* matrix of the template unit system, (b) *PC* matrix of the copy.

When two rows of the *PC* matrix are in $0 - 1$ disagreement, it means that the fault corresponding to one row is distinguishable from the fault corresponding to the other row provided that all the data elements of the faulty processors are in error. When all pairs of rows of the *PC* matrix have a $0 - 1$ disagreement, it implies that all single faults in the system are distinguishable, again provided that all data elements corresponding to the faulty processors are in error. However, in an actual system, all the data elements of a faulty processor need not be in error. Nair and Abraham introduced a stronger relation, defined as *complete disagreement* between rows, to take care of all error patterns arising from a fault pattern [5].

DEFINITION 12. *A disagreement between two rows is said to be a complete disagreement, if the disagreement exists for all possible error combinations caused by the faults associated with those rows.*

We next state a corollary of a theorem from [5].

THEOREM 9. *A system is one-fault locating if and only if PC has a complete* $0 - 1$ *disagreement with itself.*

LEMMA 3. *If PC matrices of each of the unit systems have* $0 - 1$ *disagreement with themselves and no check in the composite system has more than a single data element from a processor's data set, then the PC matrix of the composite system has a complete* $0 - 1$ *disagreement with itself.*

PROOF. Consider the *PC* matrix of the unit template system. The $0 - 1$ disagreement of this matrix with itself implies a complete $0 - 1$ disagreement, since the unit system has a single error pattern for each fault pattern. Similarly, the *PC* matrices of the other unit systems also have a complete $0 - 1$ disagreement with themselves. The different combinations of the unit systems represent different error patterns corresponding to a fault pattern. When we *combine* the *PC* matrices PC_1 and PC_2, of two unit systems S_1 and S_2, respectively, a logical OR of the corresponding entries of PC_1 and PC_2 is taken. The columns corresponding to the checks which are common in the two unit systems will have the same entries in the two *PC* matrices, so a logical OR will not change the entries. The columns which correspond to checks unique to unit

system S_1 will have all "0" entries in matrix PC_2. So when a logical OR is taken, no "0" entry of PC_1 goes to a "1" and vice versa. Therefore, a 0 – 1 disagreement which existed in matrix PC_1 is preserved in the combination matrix. Similarly, a 0 – 1 disagreement which existed in matrix PC_2 is also preserved. This shows that a complete 0 – 1 disagreement in the composite system is ensured by individual 0 – 1 disagreements in the unit system *PC* matrices. □

The *PC* matrix of the composite system should have a complete 0 – 1 disagreement with itself to ensure one-fault locatability in the system. By Lemma 3, we can consider the *PC* matrices of the unit systems instead of the *PC* matrix of the composite system. A *PC* matrix having a 0 – 1 disagreement with itself means that the ${}^2PC$ matrix should have at least one "1" in each row. For the 0 – 1 disagreement between two rows R_1 and R_2 to hold even in the presence of faults in the check_computing processors in the system, at least one "1" in the row in ${}^2PC$, corresponding to the (R_1, R_2) pair, should be true.

We define the *processor set* of a check to be the set of processors, any one of which can be used to evaluate the check. We need to find the processor sets of each check such that the mapping of checks to processors maintains the fault tolerance capability of the system, i.e., the unit system matrices ${}^rPC$, $1 \le r \le s$, all have at least one true "1" entry in each row.

Algorithm for mapping of checks

1) For each of the unit systems, construct a *PC* matrix. Tag each row of the matrix with the corresponding processor. Let d be the number of unit systems. Term the *PC* matrix of the ith unit system as PC_i, $1 \le i \le d$. A check in the final system has a column in each of the *PC* matrices. Throw out the columns corresponding to the checks which are redundant and have been removed from the composite system.
2) For each of the PC_i matrices, construct ${}^rPC_i$, $1 \le r \le s$. Each row in ${}^rPC_i$ is a combination of r rows of PC_i. Tag each row in ${}^rPC_i$ with a set of r processors corresponding to the r rows whose combination it is.
3) Initialize the processor sets of each check to all the processors in the system.
4) Columnwise concatenate all the matrices ${}^rPC_i$, $1 \le r \le s$, $1 \le i \le d$, to form one matrix PF.
5) Arrange the rows in the final matrix PF in an increasing order of the number of "1"s and number the rows from 0 onwards. Number the columns from 0 to $q - 1$, where q is the total number of checks in the system.
6) In each row of PF, mark the "1" which appears first in or after the column *i modulo q*, where i is the row number.
7) For all rows of PF starting from the first do
   $i = 0$; throw = 0;
   while (throw = 0 & $i < q$)
   if (ith column entry is "1")
   {if (set of processors in the row tag) $\bigcap$
   (ith column check's processor set) = ϕ then
   {throw the row out of the matrix;
   throw = 1;
   }
   }
   i = i + 1;}
   if (throw = 0) then do the following
   a) Let present_check = Marked "1" column.
   b) If (processor set of present_check) – (set of processors in the row tag) = ϕ, label the marked "1" as *bad* and mark a "1" which is not labeled bad. Go back to Step 7a. If the above condition is not true, remove those processors from the present_check's processor set which are also in the row tag of the present row.
   c) Throw the present row out of the matrix.
8) Map each check to any of the processors in its processor set. Here, we can consider the communication overhead or the processor load while assigning the checks to the processors.

Correctness of Algorithm: In the above method, we go through each row in the matrix and make sure that the check corresponding to the marked "1" is being evaluated on a fault-free processor. We continue modifying the processor sets of each of the checks as we go down the rows. At any given stage, the rows which have at least one true "1" entry, i.e., one "1" entry whose corresponding check does not have any of the processors in the row tag in its processor set, are thrown out of the matrix. Thus the solution which is generated at the end of the construction makes sure that each row in ${}^rPC$, $1 \le r \le s$, has at least one "1" whose corresponding check is being evaluated on a fault-free processor, i.e., at least one true "1." Therefore, each row of ${}^rPC$, $1 \le r \le s$, is completely detectable. Therefore, by Theorem 8, the system is s-fault detecting. At the same time, since all unit system ${}^2PC$ matrices also have true "1" entries in each row, the *PC* matrix of each unit system has a complete 0 – 1 disagreement with itself. Thus, by Theorem 9 and Lemma 3, the system is also ensured one-fault locatability.

The algorithm may take exponential time in the number of processors, for arbitrary values of s and n, where n is the total number of processors (both code_processors and info_processors). But in practice, the value of s is much lower than the value of n. Then the complexity of the algorithm is $O(dqn^s)$.

EXAMPLE 6. Consider the system given in Fig. 4d. The system is one-fault locating/two-fault detecting, assuming that the checks c_1, c_2, c_3, c_4, c_5, c_7, in the system are computed on fault-free processors. The checks have to be mapped to system processors such that the total system still remains one-fault locating/two-fault detecting. Fig. 5 gives the *PC* matrices of the template unit system and its copy. The ${}^2PC$ matrix is formed for both the unit systems. With the check mapping algorithm, the processor sets of the checks are obtained as follows:

Check	Processor set
c_1	$[p_2]$
c_2	$[p_5]$
c_3	$[p_2, p_8]$
c_4	$[p_1]$
c_5	$[p_4]$
c_7	$[p_8]$

We can map check c_3 onto processor p_8 to even out the processor load, and rest of the checks have only single choices in the mapping.

4 DIAGNOSIS

Once the fault-locating system has been designed and mapping of checks onto the processors is performed, the fault tolerant algorithm is run on the parallel system. It is known that the dominant cause of system failures are transient faults [19]. The effect of transient faults can be removed by performing a number of retries on the computation. If after several retries, the effect of a fault is still observed, it is declared to be a permanent fault. Once it has been determined that a permanent fault is present, a diagnosis algorithm needs to be run to locate the faulty processor from the syndrome.

For $s \geq 4$, our design procedure involves two types of checks, the basic checks and the position checks. Only the basic check part of the syndrome is required for performing diagnosis of a single fault. So, in this section, whenever we refer to checks we mean basic checks. We have ensured through our design method that no check in the system is invalidated due to its finite error detectability if only one fault occurs in the system. This is because we have guaranteed that no check has more than one data element from any processor's data set. Therefore, the *PC* matrix of the final composite system (with only the columns corresponding to the basic checks) does not have any entries greater than "1". Thus, we can use a diagnosis algorithm, which is based on the fact that the result of a checking operation is predictable in all cases except when the check_computing processor becomes faulty.

DEFINITION 13. *A data-binary $\hat{d}_i$ of a data element d_i is defined as follows:*

$\hat{d}_i = 0$ *if* d_i *is error-free,*

$\hat{d}_i = 1$ *if* d_i *is erroneous.*

DEFINITION 14. *A processor-binary $\hat{p}_{ij}$ of a processor p_i computing a check c_j is defined as follows:*

$\hat{p}_{ij} = 0$ *if processor* p_i *is fault-free,*

$\hat{p}_{ij} = 0$ *or* 1 *if processor* p_i *is faulty.*

Each check in the composite system can be represented by an equation. The check is expressed as a binary sum of data-binaries of the elements in its data set and the processor-binary of the processor computing the check. The set of equations for some or all the checks in the composite system can be represented by a matrix, defined as the *equation matrix*. The rows in the matrix correspond to data-binaries and processor-binaries and the columns correspond to the checks. There is a "1" entry in position (i, j) in the matrix if the data-binary or processor-binary corresponding to the ith row is present in the equation corresponding to the check in the jth column.

Diagnosis Algorithm:

1) Write the check equations for the composite system.
2) Remove all equations corresponding to the checks which are at "0" in the syndrome. Set the data-binaries of these equations to "0" in the remaining equations. The processor-binaries of those processors all whose data-binaries are "0" can also be removed from the set of equations. We thus get a reduced set of equations.
3) Represent the reduced set of equations by the equation matrix.
4) Take a logical OR, on an element-by-element basis, of the rows in the equation matrix corresponding to the data-binaries of the data elements in a processor's data set and the row corresponding to the processor's processor-binary.
5) The row consisting of all "1"s represents the set of erroneous data elements in the system.
6) If the row tag contains a processor-binary then the corresponding processor is faulty, else the intersection of the sets of processors producing each of the erroneous data elements gives the fault pattern.

The worst case complexity of Steps 1-3 is $O(qg)$, where q is the total number of checks in the system and g is the maximum number of data elements in a check's data set. The worst case complexity of Step 4 is $O(qv)$, where v is the total number of edges in the *PD* graph.

EXAMPLE 7. Suppose the system given in Fig. 4d gives a syndrome [1 0 0 1 1 0] for the checks c_1, c_2, c_3, c_4, c_5, c_7. The diagnosis algorithm can be run on the syndrome as follows:

Step 1: Check equations:

$$c_1 = \hat{d}_1 + \hat{d}_4 + \hat{d}_6 + \hat{p}_{21}$$
$$c_2 = \hat{d}_3 + \hat{d}_5 + \hat{d}_7 + \hat{p}_{52}$$
$$c_3 = \hat{d}_1 + \hat{d}_5 + \hat{d}_8 + \hat{p}_{83}$$
$$c_4 = \hat{d}_3 + \hat{d}_4 + \hat{d}_9 + \hat{p}_{14}$$
$$c_5 = \hat{d}_2 + \hat{d}_4 + \hat{d}_{10} + \hat{p}_{45}$$
$$c_7 = \hat{d}_2 + \hat{d}_5 + \hat{d}_{12} + \hat{p}_{87}$$

Step 2: Reduced set of check equations:

$$c_1 = \hat{d}_4 + \hat{d}_6$$
$$c_4 = \hat{d}_4 + \hat{d}_9$$
$$c_5 = \hat{d}_4 + \hat{d}_{10}$$

Step 3: Equation matrix:

$$\begin{array}{c} \\ \hat{d}_4 \\ \hat{d}_6 \\ \hat{d}_9 \\ \hat{d}_{10} \end{array} \begin{array}{c} \begin{array}{ccc} c_1 & c_4 & c_5 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

Step 4: Logical OR of rows corresponding to $\hat{d}_6$ and $\hat{d}_{10}$:

$$\begin{array}{c} \\ \hat{d}_4 \\ (\hat{d}_6, \hat{d}_{10}) \\ \hat{d}_9 \end{array} \begin{array}{c} \begin{array}{ccc} c_1 & c_4 & c_5 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

Step 5: The row corresponding to $\hat{d}_4$ covers all the checks in the equation matrix. Therefore, d_4 is the erroneous data element.

Step 6: The processor computing data element d_4 is p_3. Therefore, processor p_3 is faulty.

THEOREM 10. *In a one-fault locating system, the equation matrix obtained by throwing out columns corresponding to checks at* "0" *in the syndrome and taking a logical OR of the rows corresponding to the data-binaries of the data elements of a processor and its processor-binary, consists of one and only one row of all* "1"*s. This row represents the set of data elements in error.*

PROOF. To show that there exists a row of all 1s: Assume that there exists a single faulty processor in the system. The faulty processor will make one or more of its data elements erroneous. Any check which has one erroneous data element in its data set will always go to a "1" in the syndrome. These checks are not computed on the faulty processor and are also not invalidated due to the finite error detectability of the checks. The checks which do not have any erroneous data elements in their data sets and are also not computed on the faulty processor will always go to a "0" in the syndrome. The checks computed on the faulty processor do not have any erroneous data elements in their data sets. But they are unpredictable, so they may give either a "0" or a "1" in the syndrome. Therefore, all the checks which give a "1" in the syndrome have either an erroneous data element generated by the faulty processor in their data set or are themselves computed on the faulty processor. The row corresponding to the faulty processor's data-binaries and processor-binary has "1" entries in all the columns of the equation matrix. Therefore, there exists a row of all "1"s in the equation matrix.

To show that there exists only one row of all "1"s: We proved earlier that only those checks go to "1" in the syndrome which check a data element from the faulty processor's data set or are computed on the faulty processor itself. Suppose two rows in the reduced equation matrix have all "1" entries. One of the rows corresponds to the faulty processor's data-binaries. The data-binaries or the processor-binary corresponding to the other row should be in all the checks which go to "1." But, since the system is one-fault locating, at least one of the data-binaries corresponding to the nonfaulty processor row should be checked by a check which goes to "0." In that case, that data-binary will be set to "0" and will not appear in the reduced equation matrix. This contradicts the earlier assumption. It can similarly be shown that the assumption that more than two rows in the reduced equation matrix have all "1" entries also leads to a contradiction. □

5 FAULT MAPPING

After the fault has been located, the work of the faulty processor has to be distributed to one or more other processors in the system. This has to be done in such a way that the system is still one-fault locating/s-fault detecting. If a large number of faults occur in the system one at a time and it becomes difficult to maintain the fault tolerance capability, we degrade the fault tolerance in the system with no extra cost.

The diagnosis algorithm indicates a single processor or a subset of a processor class as faulty. We call the set of faulty processors generated by the diagnosis algorithm as the *fault set*. Let the processors in the fault set be $p'_1, p'_2, \ldots, p'_f$. Although only one of these processors is faulty, we cannot locate it due to sharing of data elements among processors. The work of all the processors in the fault set has to be distributed to other processors so as to maintain the one-fault locatability/s-fault detectability of the system. The load on a processor in the original nonfaulty system is defined as a *job*. We will refer to jobs by the names of the processors on which they are computed, e.g., the job on processor p_i will be referred to as job p_i. We assume that the jobs of the processors in the fault set are distributed to the other processors in the system in such a way that no processor is assigned more than two jobs. This is done to reduce the delay overhead of the fault tolerant computation. After the jobs are distributed, the final *PC* matrix should have a complete $0-1$ disagreement with itself and the matrices ${}^{i}PC$, $1 \le i \le s$, should have a true "1" entry in each row for every error pattern. If such a mapping cannot be found, we degrade the fault detection capability (s) of the system till a mapping is found.

We now give the algorithm for mapping the jobs initially assigned to the processors in the fault set to other processors in the system. When no fault is present in the original system, we initialize the matrix PC' to be the same as matrix *PC* of the composite system and matrix PD' to be the same as matrix *PD*.

Algorithm for fault mapping:

1) Remove the processors in the fault set from each check's processor set.
2) If any check's processor set becomes empty, no mapping exists. Go to Step 7.
3) Form matrix PD'' by removing the rows corresponding to the fault set from matrix PD'. Multiply the PD'' matrix with the *DC* matrix to get matrix PC''.
4) For each processor p'_i, $1 \le i \le f$, in the fault set do the following:
   a) For each job p_j on processor p'_i do,
      i) Let *PS* be the set of nonfaulty processors in the system which perform single jobs.
      ii) Map job p_j to each of the processors $p_k \in PS$ one at a time. For each mapping, add the row corresponding to processor p_j from the original *PD* matrix to the row corresponding to processor p_k in PD'' to form a new matrix *PT*. Multiply this with the original *DC* matrix to get matrix *T*. Check if matrix *T* has the following properties:
         - It has a complete $0-1$ disagreement with itself.
         - There are no entries greater than "1" in the

columns corresponding to the basic checks in the matrix.

- Matrix ${}^sT$ (product of ${}^sPT$ and DC matrices) has a "1" entry in every row corresponding to processor p_k for every error combination.

Let $F_i \in PS$ be the set of processors for which the mapping gives rise to matrix T which satisfies the above constraints. If F_i is empty, go to Step 6.

iii) Map the work of the faulty processor onto one of the processors in F_i. Let the resultant PD matrix of the system be PD_i' and the corresponding PC matrix be PC_i'.

iv) Let us call the rows in PC_i' changed in the earlier steps, the *modified* rows. Make all the new "1" entries, corresponding to the basic checks, in the modified rows true by removing the added processor corresponding to the modified row from the processor sets of the checks which have "1" entries. Use the check mapping method to make at least one "1" entry true, for every error combination, in those rows of ${}^sPC_i'$, which have the modified processor row in it. If at any stage the processor set of any check becomes empty, go back to Step iii in order to choose another mapping from set F_i, if one still exists, and go through Step iii and Step iv again. If there is no mapping left in set F_i, go to Step 6.

b) If processor p_i' has a check mapped to it, remap the check to another processor in its processor set.

5) Let $PD' = PD_f'$. STOP.

6) Let PS be the set of processors with only single jobs on them. Map each faulty processor's job to some processor in PS such that no processor has more than two jobs on it. There will be f processors with two jobs on them. If any of the faulty processors had a check mapped to it, map the check to another processor in its processor set. This system will be one-fault locating/$(s-f)$-fault detecting for $s > 2f$ and will be one-fault locating/$\frac{s}{2}$-fault detecting for $2 \le s \le 2f$. STOP.

7) If at any stage the level of fault tolerance falls below the desired level, the system has to be redesigned for one-fault locatability/s-fault detectability. Map the jobs of the processors in the fault set onto other nonfaulty processors in the system with a single job on them. If a processor in the fault set has two jobs on it, map the two jobs to two different processors in the system. The PD graph of the original system is now changed. The DC part is redesigned for one-fault locatability using unit system and composite system construction procedures. Then check mapping is done as before.

In Step 4a)ii, we go through each row containing processor p_k in matrix ${}^sT$, so the complexity of the step is $O((n-f)^{s-1}q)$. In the worst case, we check the mapping of the faulty processor to all $(n-f)$ nonfaulty processors in the system. Therefore, the complexity for mapping each processor in the fault set is $O((n-f)^s q)$. There are f processors in the fault set and, for each processor, the fault mapping algorithm is run. In the worst case, $f = \alpha$ (see Theorems 1 and 2). Thus the overall complexity of the algorithm is $O(\alpha n^s q)$.

Correctness of Algorithm: In Step 1, we remove the faulty processors from the system. A check's processor set represents the set of processors on which the checking operation can be performed. In Step 2, if the processor set of any check becomes empty, it implies that the checking operation cannot be performed on any processor without invalidating the conditions for fault detectability/locatability. Hence, in Step 6, the fault detectability of the system is degraded. If all the checks have nonempty processor sets in Step 2, we proceed to Steps 3 and 4. For each faulty processor, we take each job on the processor and find the candidate processor to which this job can be moved without violating the one-fault locatability/s-fault detectability constraints. The conditions that matrices T and ${}^sT$ have to satisfy in Step 4a)ii are derived from Theorems 8 and 9. These conditions along with the check mapping in Step 4a)iv ensure that the one-fault locatability/s-fault detectability of the system is maintained. If at any step no solution is found, the fault mapping procedure degrades the fault detection capability of the system as shown in Step 6.

EXAMPLE 8. Suppose in the ABFT system given in Fig. 4d, a fault is located in processor p_3.

$$PC' = PC = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{array} \begin{array}{c} \begin{array}{cccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_7 \end{array} \\ \left[\begin{array}{cccccc} 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right] \end{array}$$

Step 1: Fault set = $[p_3]$. So $p_i' = p_3$. No processor set derived in Example 6 has p_3 in it.

Step 3:

$$PC'' = \begin{array}{c} \\ p_1 \\ p_2 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{array} \begin{array}{c} \begin{array}{cccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_7 \end{array} \\ \left[\begin{array}{cccccc} 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right] \end{array}$$

Step 4:

a) $PS = [p_1, p_2, p_4, p_5, p_6, p_7, p_8]$

b) $F_1 = [p_4, p_6, p_7]$

c) We map job p_3 on processor p_6.

$$PC_1' = \begin{array}{c} \\ p_1 \\ p_2 \\ p_4 \\ p_5 \\ p_3 \Rightarrow p_6 \\ p_7 \\ p_8 \end{array} \begin{array}{c} \begin{array}{cccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_7 \end{array} \\ \left[\begin{array}{cccccc} 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1^* & 1 & 0 & 1^* & 1^* & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right] \end{array}$$

d) The row corresponding to tag $p_3 \Rightarrow p_6$ is the modified row. The "1"s marked with "*" are the new "1" entries in the row. The checks c_1, c_4, and c_5 cannot be performed on processor p_6. Consider the rows of ${}^2PC_1'$ which have processor p_6 in their tags. Each of these rows should have at least one true "1" entry for every error pattern. The "*" entries in the sub-matrix of ${}^2PC_1'$ given below are true from the present processor sets.

$$\begin{array}{c} \\ (p_1,p_6) \\ (p_2,p_6) \\ (p_4,p_6) \\ (p_5,p_6) \\ (p_7,p_6) \\ (p_8,p_6) \end{array} \begin{array}{c} \begin{array}{cccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_7 \end{array} \\ \left[\begin{array}{cccccc} 2^* & 1^* & 1^* & 1 & 2^* & 1^* \\ 1 & 2^* & 0 & 2^* & 2^* & 1^* \\ 1^* & 2^* & 1^* & 1^* & 1 & 1^* \\ 2^* & 1 & 0 & 1^* & 2^* & 0 \\ 1^* & 1^* & 1^* & 1^* & 1^* & 1^* \\ 1^* & 1^* & 0 & 2^* & 1^* & 0 \end{array}\right] \end{array}$$

For every error pattern there is a "1" entry which is true in each row. Job p_3 is, therefore, given to processor p_6, with no loss in the fault tolerance capability of the system.

6 Conclusions

In this paper, we have given deterministic methods for designing one-fault locating/s-fault detecting systems under the extended ABFT model. We proposed a new diagnosis algorithm to locate the fault in the system. After fault location, we map the work of the faulty processors to other nonfaulty processors in the system such that the system maintains the desired level of fault tolerance, with a graceful degradation in performance.

Acknowledgment

This work was supported by the U.S. Office of Naval Research under Contract no. N00014-91-J-1199.

References

[1] K.H. Huang and J.A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," *IEEE Trans. Computers*, vol. 33, no. 6, pp. 518-528, June 1984.

[2] J.Y. Jou and J.A. Abraham, "Fault Tolerant Matrix Arithmetic and Signal-Processing on Highly Concurrent Computing Structures," *Proc. IEEE*, vol. 74, no. 5, pp. 732-741, May 1986.

[3] J.A. Abraham et al., "Fault Tolerance Techniques for Systolic Arrays," *Computer*, pp. 65-74, July 1987.

[4] P. Banerjee and J.A. Abraham, "Bounds on Algorithm-Based Fault Tolerance in Multiple Processor Systems," *IEEE Trans. Computers*, vol. 35, no. 4, pp. 296-306, Apr. 1986.

[5] V.S.S. Nair and J.A. Abraham, "A Model for the Analysis of Fault Tolerant Signal Processing Architectures," *Proc. Int'l Technical Symp. SPIE*, San Diego, pp. 246-257, Aug. 1988.

[6] S. Yajnik and N.K. Jha, "Design of Algorithm-Based Fault Tolerant Systems with In-System Checks," *Proc. Int'l Conf. Parallel Processing*, vol. 1, St. Charles, Ill., Aug. 1993.

[7] S. Yajnik and N.K. Jha, "Analysis and Randomized Design of Algorithm-Based Fault Tolerant Multiprocessor Systems Under the Extended Graph-Theoretic Model," *Proc. ISCA Parallel and Distributed Computing & Systems*, pp. 52-57, Louisville, Ky., Oct. 1993.

[8] D. Gu, D.J. Rosenkrantz and S.S. Ravi, "Design and Analysis of Test Schemes for Algorithm-Based Fault Tolerance," *Proc. Int'l Symp. Fault-Tolerant Computing*, pp. 106-113, Newcastle-upon-Tyne, U.K., June 1990.

[9] R. Sitaraman and N.K. Jha, "Optimal Design of Checks for Error Detection and Location in Fault Tolerant Multiprocessor Systems," *IEEE Trans. Computers*, vol. 42, no. 7, pp. 780-793, July 1993.

[10] P. Banerjee and J.A. Abraham, "Concurrent Fault Diagnosis in Multiple Processor Systems," *Proc. Int'l Symp. Fault-Tolerant Computing*, pp. 298-303, Vienna, June 1986.

[11] D.J. Rosenkrantz and S.S. Ravi, "Improved Bounds on Algorithm-Based Fault Tolerance," *Proc. Ann. Allerton Conf. Comm., Cont. and Computing*, pp. 388-397, Allerton, Ill., Sept. 1988.

[12] B. Vinnakota and N.K. Jha, "Design of Multiprocessor Systems for Concurrent Error Detection and Fault Diagnosis," *Proc. Int'l Symp. Fault-Tolerant Computing*, pp. 504-511, Montreal, June 1991.

[13] V.S.S. Nair and J.A. Abraham, "A Model for the Analysis, Design and Comparison of Fault-Tolerant WSI Architectures," *Proc. Workshop Wafer Scale Integration*, Como, Italy, June 1989.

[14] B. Vinnakota and N.K. Jha, "Diagnosability and Diagnosis of Algorithm-Based Fault Tolerant Systems," *IEEE Trans. Computers*, vol. 42, no. 8, pp. 924-937, Aug. 1993.

[15] B. Vinnakota and N.K. Jha, "A Dependence Graph-Based Approach to the Design of Algorithm-Based Fault Tolerant Systems," *Proc. Int'l Symp. Fault-Tolerant Computing*, pp. 122-129, Newcastle-upon-Tyne, U.K., June 1990.

[16] F.T. Luk and H. Park, "An Analysis of Algorithm-Based Fault Tolerance Techniques," *Proc. SPIE Advanced Algebraic Architecture and Signal Processing*, vol. 696, pp. 222-228, Aug. 1986.

[17] V.S.S. Nair and J.A. Abraham, "Hierarchical Design and Analysis of Fault-Tolerant Multiprocessor Systems Using Concurrent Error Detection," *Proc. Int'l Symp. Fault-Tolerant Computing*, pp. 130-137, Newcastle-upon-Tyne, U.K., June 1990.

[18] D.M. Blough and A. Pelc, "Almost Certain Fault Diagnosis Through Algorithm-Based Fault Tolerance," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 5, pp. 532-539, May 1994.

[19] R.K. Iyer and D.J. Rossetti, "Permanent CPU Errors and Systems Activity: Measurement and Modeling," *Proc. Real-Time Systems Symp.*, pp. 61-72, Arlington, Va., Dec. 1983.

Shalini Yajnik (S'92-M'95) received the BTech degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1990, and the PhD degree in electrical engineering from Princeton University, Princeton, New Jersey, in 1994. Since 1994, she has been working as a member of the technical staff at Lucent Technologies Bell Laboratories (formerly, AT&T Bell Laboratories) in Murray Hill, New Jersey.

Her research interests include fault tolerant computing, checkpointing, algorithm-based fault tolerance, switching systems, and design of efficient power and error control algorithms in wireless networks.

Niraj K. Jha (S'85-M'86-SM'93) received his BTech degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1981, his MS degree in electrical engineering from the State University of New York at Stony Brook in 1982, and his PhD degree in electrical engineering from the University of Illinois, Urbana-Champaign, in 1985.

Dr. Jha is an associate professor of electrical engineering at Princeton University. He served as an associate editor of *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*. He also served as the program chairman of the 1992 Workshop on Fault-Tolerant Parallel and Distributed Systems. He is the recipient of the AT&T Foundation award and the NEC Preceptorship award. He coauthored the book *Testing and Reliable Design of CMOS Circuits* (Kluwer Academic Publishers). He is the author or coauthor of more than 110 technical papers. He won the Best Paper award at the International Conference on Computer Design in 1993. His research interests include digital system testing, fault-tolerant computing, computer-aided design of integrated circuits, distributed computing, and real-time computing.