Algorithm-Based Fault Location and Recovery for Matrix Computations *

Amber Roy-Chowdhury and Prithviraj Banerjee
Coordinated Science Lab
1308 W Main Street
University of Illinois
Urbana IL 61801
email: amber@crhc.uiuc.edu or banerjee@crhc.uiuc.edu
phone: 217-333-6564 fax: 217-244-5685

Abstract

Previous algorithm-based methods for developing reliable versions of numerical algorithms have mostly concerned themselves with error detection. A truly fault-tolerant algorithm, however, needs to locate errors and recover from them once they are located. In a parallel processing environment, this corresponds to locating the faulty processors and recovering the data corrupted by the faulty processors. In our paper, we discuss in detail a fault-tolerant version of a matrix multiplication algorithm. The ideas developed in the derivation of the fault-tolerant matrix multiplication algorithms may be used to derive fault-tolerant versions of other numerical algorithms. We outline how two other numerical algorithms, QR factorization and Gaussian Elimination may be made fault-tolerant using our approach. Our fault model assumes that a faulty processor can corrupt all the data it possesses. We present error coverage and overhead results for the single faulty processor case for fault-locating and fault-tolerant versions of three numerical algorithms on an Intel iPSC/2 hypercube multicomputer.

Keywords: Parallel Algorithms, Algorithm-Based Fault Tolerance, Error Recovery, Multiple Faults, Weighted Checksum Encoding

1 Introduction

Algorithm-based fault tolerance techniques (ABFT) are a cost effective method for increasing the reliability of applications running on hardware which may incorporate little or no fault tolerance. The technique was first proposed for numerical algorithms executing on systolic architectures which required both modification of the algorithms as well as the hardware for correct and reliable operation [1],[2]. Some algorithms devised for these architectures could locate and correct errors in case of a single processor fault. Later, reliable algorithms were devised for general purpose multicomputers requiring no modification of the underlying hardware for a wide variety of numerical applications [3],[4]. However, most of the reliable algorithms for general purpose multicomputers have concentrated on error detection rather than the full problem of locating the errors and recovering from the errors once these are located. In a reliable algorithm devised for a multicomputer environment, this corresponds to detecting errors caused by a single or multiple faulty processors, locating the set of faulty processors, and then either recovering the data by incurring some additional computational overhead or restarting the application on the remaining nonfaulty processors. Banerjee and Abraham [5] derived a graph theoretic model for studying the error detecting and locating capabilities for various data and check distributions for ABFT schemes. Based on their model, they derived bounds on the number of checks required for achieving a certain level of fault-tolerance. Subsequently, these bounds were improved by Gu, Rosenkrantz and Ravi in [6]. In [7], Vinnakota and Jha proposed a synthesis of ABFT schemes from dependence graphs. Subsequently, in [8], Vinnakota and Jha extended the ABFT synthesis scheme to systems where data elements are shared by multiple processors. However, none of the approaches so far have attempted to apply their design schemes to real applications to demonstrate their practicality. In this paper, we demonstrate the practicality of our ABFT schemes for fault location and correction by presenting results on three popular numerical algorithms. Also, the literature for ABFT design reported so far has not addressed the issue of extending the design scheme to perform forward error recovery (recovery without having to restart the algorithm from the very beginning). We present a scheme which is able to achieve forward error recovery for the case when t processors might be faulty, where t is a design parameter.

In this paper we present approaches for locating and correcting errors due to multiple faulty processors by a

*Acknowledgement: This research was supported in part by the Office of Naval Research under contract N00014-91-J-1096 and in part by the Joint Services Electronics Program under contract N00014-84-C-1049.

suitable encoding of parallel algorithms for general purpose distributed memory multicomputers. We present a version of a matrix multiplication algorithm which works on data encoded initially in such a manner that even in the event of multiple processor failures, the result matrix may be recovered from the encoded data on the fault-free processors. Our approach is based on a generalization of the weighted checksum approach, which has been discussed in [9]. Our approach may be applied to a wide variety of numerical algorithms to make them fault-tolerant. To illustrate this, we present error coverage and timing overhead results on three algorithms - matrix multiplication, QR factorization, and Gaussian elimination,

The organization of this paper is as follows. In Section 2, we discuss an algorithm for matrix multiplication which can locate a single faulty processor. We then discuss a modification which can recover corrupted data in the event of a single processor failure in Section 3. We next present a general method for fault location and then discuss a recovery scheme in which the corrupted data may be recovered in the event of multiple processor failures in Sections 4 and 5. In Section 7, we present experimental results to demonstrate the overhead incurred in our encoding of single-fault-tolerant versions of matrix multiplication, QR factorization and Gaussian elimination algorithms over the corresponding simple algorithms involving no fault tolerance. Overheads are presented for both the single fault location as well as the single fault correction algorithms in each case.

2 A Matrix Multiplication Algorithm for Single Fault Location

We assume that we have available a multiple processor system with p processors which is responsible for performing the computations of our parallel algorithm. We refer to each processor responsible for such computations as a node. We assume we have a module called the host responsible for initial loading of data and collection and interpretation of results. While we assume that one or more nodes may be faulty and may corrupt all the data they possess, the host is assumed to be fault free. We assume that the non-arrival of a message can be detected by a timeout or other means and constitutes an error.

2.1 Algorithm description

We consider the problem of multiplying two nxn matrices A and B to obtain an nxn matrix C on a parallel computer with p nodes. Our initial data distribution replicates B on all nodes. A is divided into strips of dimension mxn where $m = \frac{n}{p}$. (In order to simplify the discussion we assume that n is divisible by p, though problems for which this is not true may be converted to a problem with n divisible by p by the addition of at most $p-1$ rows of 0's to A). Let us denote the mxn strip of A possessed by node i by A_i, $0 \leq i \leq p-1$. For simplicity of explanation, we assume that p is a multiple of 3 and group the p nodes into $\frac{p}{3}$ disjoint groups of 3 nodes each. We call each of these groups a check group. The case when p is not a multiple of 3 may be handled in several ways. A simple method is to let each check group consist of 3 or 4 nodes, so that the extra nodes are included in check groups with 4 nodes.(There will be 0,1 or 2 check groups with 4 nodes depending on whether p mod 3 is 0,1 or 2). Let us denote the nodes in the ith check group by $g_0^{(i)}, g_1^{(i)}$ and $g_2^{(i)}$. The nodes in each check group are logically configured in a directed cycle, which we subsequently refer to as a check group ring. Before the start of the actual matrix multiplication of its own strip of A with B, each node receives and computes the checksum of the rows of the strip of A belonging to the node immediately following it in its check group ring. Thus, in matrix notation, node $g_j^{(i)}$ computes

$$(acs_{g^{(i)}_{(j+1)\bmod 3}})^T = e^T A_{g^{(i)}_{(j+1)\bmod 3}} \qquad (1)$$

where e^T denotes the $1xm$ row vector with all 1 components and $(acs_{g_k^{(i)}})^T$ denotes the $1xn$ checksum row of $A_{g_k^{(i)}}$. Each node then proceeds to perform the usual matrix multiplication of its own strip of A with the matrix B. In addition, each node also computes the vector-matrix product of the checksum it possesses with B. Thus, the following computations are performed by node $g_j^{(i)}$

$$C_{g_j^{(i)}} = A_{g_j^{(i)}} B \qquad (2)$$

$$(ccs_{g^{(i)}_{(j+1)\bmod 3}})^T = (acs_{g^{(i)}_{(j+1)\bmod 3}})^T B \qquad (3)$$

Following the computation of $C_{g_j^{(i)}}$, node $g_j^{(i)}$ also computes the checksum of $C_{g_j^{(i)}}$, which we denote by $(\widehat{ccs}_{g_j^{(i)}})^T$. In matrix notation, we have

$$(\widehat{ccs}_{g_j^{(i)}})^T = e^T C_{g_j^{(i)}} \qquad (4)$$

Next, node $g_j^{(i)}$ sends $\widehat{ccs}_{g_j^{(i)}}$ to node $g^{(i)}_{(j-1)\bmod 3}$. After node $g_j^{(i)}$ receives $\widehat{ccs}_{g^{(i)}_{(j+1)\bmod 3}}$ from node $g^{(i)}_{(j+1)\bmod 3}$, it compares $\widehat{ccs}_{g^{(i)}_{(j+1)\bmod 3}}$ and $ccs_{g^{(i)}_{(j+1)\bmod 3}}$. In the absence of node failures, $\widehat{ccs}_{g^{(i)}_{(j+1)\bmod 3}}$ must equal $ccs_{g^{(i)}_{(j+1)\bmod 3}}$ to within a tolerance. (The tolerance is necessary due to the accumulation of roundoff errors in the computation of $\widehat{ccs}_{g^{(i)}_{(j+1)\bmod 3}}$ and $ccs_{g^{(i)}_{(j+1)\bmod 3}}$. For a discussion of a tolerance determination methodology, see [10]). In the event of a single node failure, the checksum test for the strip of C computed by the faulty node fails on the node preceding it in its check group ring. The checksum test on the faulty node itself may fail or pass. In either case, the test on the node immediately preceding the faulty node fails and so the syndrome for any single fault within a check group is unique. The identity of the faulty node may thus be determined by the host after receiving the results of the checksum tests of each node. In fact, if the fault pattern is such that only one node in each check group is faulty, such fault patterns can also be detected and located.

The overheads posed by this scheme are $O(n^2)$ extra operations compared to $O(\frac{n^3}{p})$ for the original algorithm and thus become negligible for $n \gg p$.

3 A Matrix Multiplication Algorithm for Recovery From a Single Fault

3.1 Algorithm description

This algorithm is a modification of the algorithm described in the previous section. Initially, the matrix B is replicated on all p nodes, as before, but the matrix A is only distributed over $p-1$ nodes, so that the strips of A are of dimensions mxn where now $m = \frac{n}{p-1}$. (As before, for simplicity of explanation and to avoid cluttering the notation, we assume that n is divisible by $p-1$). We denote these strips as before by A_i, $0 \leq i \leq p-2$ with strip A_i going to node i. Now the host computes an extra strip A_{p-1} of dimension mxn which is the sum of all the other A_i's and communicates the strip to node $p-1$. Thus, we have

$$A_{p-1} = \sum_{i=0}^{p-2} A_i \tag{5}$$

As in the algorithm described in Section 2, nodes are grouped into disjoint check group rings of 3 (the case when p is not divisible by 3 may be treated as in the previous section), and each node maintains the checksum of the strip of A of the node immediately following it in its check group ring. The matrix multiplication proceeds exactly as before with node i generating $C_i = A_i B$. We also have the following relation between C_{p-1} and the rest of the C_i's

$$\begin{aligned} C_{p-1} &= A_{p-1}B \\ &= \sum_{i=0}^{p-2} A_i B \\ &= \sum_{i=0}^{p-2} C_i \end{aligned} \tag{6}$$

Each node also produces the product of its local checksum with B, exactly as before. Thus any single faulty node may be identified by the host, as before. If the host now determines that a single fault has occurred, it takes the following action depending on whether the faulty node is node $p-1$ or one of the other nodes. In the former case, the host need not initiate any recovery action, since it can directly assemble $C = AB$ from the C_i's computed by nodes 0 through $p-2$ thus

$$C = \begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{p-2} \end{pmatrix} \tag{7}$$

In the event that node f is faulty, where $0 \leq f \leq p-2$, the host first proceeds to recover C_f from the data it receives from the remaining nodes as shown in the following equation which follows directly from Eqn. (6)

$$C_f = C_{p-1} - \sum_{i=0, i\neq f}^{p-2} C_i \tag{8}$$

The matrix C may then be assembled as in Eqn. (7).

The asymptotic overhead of this scheme over the basic algorithm is $\frac{1}{p-1}$. If the number of nodes in the system is large, the asymptotic overhead is very small.

4 A Matrix Multiplication Algorithm for Multiple Fault Location

4.1 Algorithm description

Now we present a generalization of the algorithm in Section 2 which can locate multiple node faults. As before, matrix B is replicated on all nodes while A is divided into strips A_i of dimensions mxn where $m = \frac{n}{p-1}$, as before (As before, we assume that n is divisible by p; if this is not the case we may add extra rows of 0's to A to make it so). However, if t fault location is desired for $t > 1$, each check group now includes $2t+1$ nodes and every node is included in exactly one check group. (The case when p is not divisible by $2t+1$ may be treated by making some check groups consisting of $2t+2$ nodes; we may require at most $2t$ check groups with $2t+2$ nodes). Each node also needs to maintain the checksums of the strips of A of exactly t other nodes within its check group. Before we can proceed to describe which t checksums each node needs to maintain, we need the following definition of a $D_{\delta,t}$ system.

Definition 1 *A $D_{\delta,t}$ system is a directed graph $G = (V, E)$ where an edge $e_{ij} \in E$ exists from a vertex $v_i \in V$ to a vertex $v_j \in V$ if and only if $j = (i + \delta m) \bmod n$ where n is the number of vertices in G, δ is an integer and $m = 1, 2, \ldots, t$.*

It is well known that a one-step t-fault diagnosable system [11][12] such as the one we are seeking to construct for the parallel matrix multiplication algorithm must satisfy the following two constraints:

a. There must be at least $2t+1$ nodes in the system.

b. Each node must be diagnosed by at least t other nodes.

It is also known [11][12] that for a class of $D_{\delta,t}$ systems with $n = 2t+1$ and in which δ and n are relatively prime, the conditions stated above are not only necessary but also sufficient if we assume that the vertices of the graph represent the processing nodes in the system and the edges represent the testing links. Such systems are thus optimal both with respect to the number of processing nodes as well as testing links.

From the discussion in the previous paragraph, it is obvious now which t checksums each node needs to maintain. Check groups consisting of $2t+1$ nodes are formed so that node $g_j^{(i)}$ corresponds to vertex v_j of an optimal $D_{\delta,t}$ system. Node $g_j^{(i)}$ maintains checksums of the strips of A of exactly those nodes within its check group which correspond to vertices of the $D_{\delta,t}$ system which are adjacent to v_j. Node i then performs the multiplication of A_i with B to generate C_i as well as multiplying the checksum rows it possesses with B to generate checksums of the strips of C produced by the nodes it is assigned to test. After each node performs checksum comparisons, it sends the results back to the host which will then be able to diagnose the faulty nodes in the event that t or fewer node failures have occurred on each check group. (Since each check group was configured so that it was isomorphic to an optimal $D_{\delta,t}$ system, every scenario involving failure of t or fewer nodes within a check group is guaranteed to result in a unique syndrome of diagnosis results from the nodes, resulting in a correct diagnosis of the faulty nodes by the host). If t is small, the host may do the diagnosis by keeping a table of syndromes, one for each pattern of t or fewer faults, and perform a table lookup with the syndrome received from the nodes. If t is large, then one of the several diagnosis algorithms mentioned in the literature for system level diagnosis based on the PMC model [12], may be used by the host to determine the faulty nodes.

The overheads posed by this scheme are $O(tn^2)$ extra operations, compared to $O(\frac{n^3}{p})$ for the original algorithm. Thus, the overheads become negligible when $n \gg pt$.

5 A Matrix Multiplication Algorithm for Recovery From Multiple Faults

In this section we present an algorithm for recovery from multiple faults which builds on the algorithm of the previous section. In order to prove the results in this section we shall need the following lemma

Lemma 1 *Vectors which are linearly independent over $GF(q)$ (q prime), where $GF(q)$ denotes the finite field with q elements, are also linearly independent over the field of real numbers.*

Proof: Refer to [13] □

We partition the set of p nodes into two sets, one consisting of nodes 0 through $p-t-1$, which we denote by $\mathcal{P}$, and the other consisting of nodes $p-t$ through $p-1$, which we denote by $\mathcal{C}$. In the rest of the section we will refer to the nodes in set $\mathcal{P}$ as computation nodes and number them from 0 through $p-t-1$, while we will refer to the nodes in set $\mathcal{C}$ as check nodes and number them from 0 through $t-1$. Although the numbering of the nodes in the two sets overlap, context will serve to distinguish the two.

The method of data distribution and the fault recovery procedure described in the remainder of the section guarantees recovery from t node failures if all failures are confined to set $\mathcal{P}$, and $2(\sqrt{t+1}-1)$ node failures if node failures can occur in both sets $\mathcal{P}$ and $\mathcal{C}$. We would like to mention at the outset that in fact, many correctable failure patterns exist where the number of failures exceeds the lower bound for the second case mentioned, and, in fact, our correction algorithm is able to correct any correctable failure pattern involving less than t faulty nodes even when faults can affect nodes from both sets $\mathcal{P}$ and $\mathcal{C}$.

5.1 Algorithm description

As in the algorithms in the earlier sections, matrix B is replicated on all nodes. A is distributed over computation nodes so that computation node i, $0 \leq i \leq p-t-1$, possesses a strip of A of dimensions mxn, where $m = \frac{n}{p-t}$, which we denote as before by A_i. Each check node j receives a strip S_j, $0 \leq j \leq t-1$, of dimensions mxn, which is a weighted sum of the strips A_i, $0 \leq i \leq p-t$. The weights are chosen so that for a particular S_j, all the elements in a particular strip A_i are multiplied by the same weight. The relation between the S_j's and the A_i's is given by the following equation

$$S_j = \sum_{i=0}^{p-t-1} w_{ji} A_i, \; 0 \leq j \leq t-1 \tag{9}$$

We also denote by S the following $mtxn$ matrix

$$S = \begin{pmatrix} S_0 \\ S_1 \\ \vdots \\ S_{t-1} \end{pmatrix} \tag{10}$$

and we define $T = SB$ and $T_j = S_j B$ so that

$$T = \begin{pmatrix} T_0 \\ T_1 \\ \vdots \\ T_{t-1} \end{pmatrix} \tag{11}$$

The procedure for choosing the weights is described next and we prove later that this choice of weights leads to the error correcting capabilities claimed at the start of the section.

Let α be a primitive element over $GF(q)$, where q is any prime greater than $p-t$. Let the weights be chosen so that $w_{ij} = \alpha^{ij} \bmod q$. Let us define the matrix W as follows

$$W = \begin{pmatrix} w_{00} & w_{01} & \cdots & w_{0\,p-t-1} \\ w_{10} & w_{11} & \cdots & w_{1\,p-t-1} \\ \vdots & \cdots & \cdots & \vdots \\ w_{t-1\,0} & w_{t-1\,1} & \cdots & w_{t-1\,p-t-1} \end{pmatrix} \tag{12}$$

Then, we have the following property possessed by the columns of W

Lemma 2 *Consider any submatrix W_{SM} of W consisting of any c consecutive rows of W. Every c columns of W_{SM} are linearly independent over the field of real numbers*

Proof: (For the properties used in the following proof the reader is referred to [14])
Replacing each w_{ij} by α^{ij} in W_{SM}, we find that W_{SM} consists of the first $p-t$ columns of the parity check matrix of a $(q, q-c)$ Reed-Solomon code. A $(q, q-c)$ Reed-Solomon code has minimum distance $c+1$ and so any c columns of its parity check matrix $H_{RS(q,q-c)}$ are linearly independent

over $GF(q)$. Since W_{SM} consists of the first $p-t$ columns of $H_{RS(q,q-c)}$, any c columns of W_{SM} are also linearly independent over $GF(q)$. By Lemma 1, any c columns of W_{SM} are linearly independent over the field of real numbers as well □

The following two corollaries follow immediately from Lemma 2

Corollary 1 Every t columns of W are linearly independent over the field of real numbers

Corollary 2 Every cxc submatrix of W_{SM} is of full rank

As in the algorithm for location of multiple faults described in Section 4, the p node system is partitioned into $\frac{p}{2t+1}$ disjoint check groups consisting of $2t+1$ nodes each, which maintain checksums of the A strips or S strips, as the case may be, of t other nodes belonging to their check group. The multiplication of the matrix strip and the checksums possessed locally by each node with B is carried out in exactly the same manner as the algorithm in Section 4. As before, each computation node i produces a strip $C_i = A_i B$. Each check node j also produces a strip $T_j = S_j B$. In the absence of faults, C may be assembled from the C_i's while if faults are present, we have the problem of recovering the corrupted strips of C from the noncorrupted C_i's and the noncorrupted T_j's. The faulty nodes may be diagnosed by the host based on the responses received from the nodes again in a manner similar to that described in Section 4. Once the host has located the faulty nodes, it may initiate the data recovery as follows.

Let the set of faulty nodes be denoted by $\mathcal{F}$. Let us define two subsets $\mathcal{F}_P \subseteq \mathcal{F}$ and $\mathcal{F}_C \subseteq \mathcal{F}$ such that every node in $\mathcal{F}_P$ belongs to $\mathcal{P}$ and every node in $\mathcal{F}_C$ belongs to $\mathcal{C}$. Let the cardinality of $\mathcal{F}_P$ be denoted by ν_P and the cardinality of $\mathcal{F}_C$ be denoted by ν_C. Let the indices of the nodes in $\mathcal{F}_P$ be $f_{P_0}, f_{P_1}, \ldots, f_{P_{\nu_P}}$, the indices of the nodes in $\mathcal{F}_C$ be $f_{C_0}, f_{C_1}, \ldots, f_{C_{\nu_C}}$, the indices of the nodes in $\mathcal{P}-\mathcal{F}_P$ be $g_{P_0}, g_{P_1}, \ldots, g_{P_{p-t-\nu_P-1}}$, and the indices of the nodes in $\mathcal{C}-\mathcal{F}_C$ be $g_{C_0}, g_{C_1}, \ldots, g_{C_{t-\nu_C-1}}$. Let us consider the system of matrix equations which may be derived from system of matrix equations in Eqn. (9) by deleting equations corresponding to indices in $\mathcal{F}_C$ and moving matrix terms corresponding to indices in $\mathcal{P}-\mathcal{F}_P$ and $\mathcal{C}-\mathcal{F}_C$ to the right hand side and then postmultiplying both sides by B

$$\sum_{i=0}^{\nu_P-1} w_{f_{P_i} g_{C_j}} C_{f_{P_i}} = T_{g_{C_j}} - \sum_{i=0}^{p-t-\nu_P-1} w_{g_{P_i} g_{C_j}} C_{g_{P_i}}, \quad 0 \leq j \leq t-\nu_C-1 \tag{13}$$

We notice that in Eqn. (13), the left hand side involves C_i's which are unknown since they were to have been computed by the nodes in the faulty set $\mathcal{F}_P$ while the right hand side involves known C_i's and T_j's since these were computed by nodes in nonfaulty sets $\mathcal{P}-\mathcal{F}_P$ and $\mathcal{C}-\mathcal{F}_C$. Thus we have a system of $t-\nu_C$ matrix equations in ν_P matrix unknowns which may be solved if there exist at least ν_P linearly independent equations involving the unknowns in the system.

We use C_R to denote the reduced matrix constructed from C with the C_i's corresponding to indices in $\mathcal{F}_P$ deleted. Let us further denote by C_{R_f} the matrix consisting of only those C_i's in C_R with indices in $\mathcal{F}_C$ and by C_{R_g} the remaining C_i's in C_R. Let us denote by W_R the reduced matrix constructed by deleting from W all rows corresponding to indices of nodes in $\mathcal{F}_C$ and a reduced matrix T_R by deleting from T all T_j's corresponding to indices of nodes in $\mathcal{F}_C$. Let us now define W_{R_f} to be the matrix consisting of only those columns of W_R with indices corresponding to nodes in $\mathcal{F}_P$ and W_{R_g} to be the matrix consisting of the remaining columns of W_R. Then, Eqn. (13) may be represented more succinctly in matrix notation by

$$W_{R_f} C_{R_f} = T_R - W_{R_g} C_{R_g} \tag{14}$$

In Eqn. (14), unlike normal matrix multiplication, each element of W_{R_f} and W_{R_g} multiplies an entire mxn block of C_{R_f} and C_{R_g} respectively.

W_{R_f} is a matrix of dimensions $(t-\nu_C)x\nu_P$. The system represented by Eqn. (14) possesses a unique solution if and only if the rank of W_{R_f} equals ν_P. The host can construct Eqn. (14) using only data from nonfaulty nodes and leaving the data from faulty processors as unknowns to be solved for. The host premultiplies both sides of Eqn. (14) by $W_{R_f}^T$ to get the new system

$$(W_{R_f}^T W_{R_f}) C_{R_f} = W_{R_f}^T (T_R - W_{R_g} C_{R_g}) \tag{15}$$

The product of $W_{R_f}^T$ with W_{R_f} follows the usual definition of matrix products but the product of $(W_{R_f}^T W_{R_f})$ with C_{R_f} and the product of $W_{R_f}^T$ with $(T_R - W_{R_g} C_{R_g})$ is such that a single element of $(W_{R_f}^T W_{R_f})$ or $W_{R_f}^T$ multiplies a mxn block of C_{R_f} or $(T_R - W_{R_g} C_{R_g})$ respectively, in keeping with the fact that the unknowns in Eqn. (15) are mxn matrices rather than simple elements. The host then performs an LU decomposition of the $\nu_P x \nu_P$ matrix $(W_{R_f}^T W_{R_f})$. (Note that the system defined by Eq. 15 is symmetric, so that its LU decomposition may be obtained by using only half the operations and memory for a general unsymmetric system). If the rank of $(W_{R_f}^T W_{R_f})$ is ν_P, i.e., the matrix is of full rank, its LU decomposition does not lead to any 0 elements occurring on the diagonals of either triangular matrix in the decomposition of $(W_{R_f}^T W_{R_f})$. (If roundoff errors are of concern, we may first attempt to determine if the matrix is of full rank over $GF(q)$ and only proceed with the LU decomposition over the field of real numbers if we find that the matrix has full rank over $GF(q)$. As a consequence of Lemma 1, a matrix with full rank over $GF(q)$ must have full rank over the field of real numbers. This extra step adds approximately $\frac{\nu_P^3}{3}$ operations, which is not expensive since ν_P can be expected to be small). All unknown elements can then be recovered by backsolves. The matrix $(W_{R_f}^T W_{R_f})$ is of full rank if and only if $W_{R_f}^T$ has rank ν_P [15]. So in these cases, the host is able to recover from the pattern of faulty nodes.

We now examine in what cases the matrix $(W_{R_f}^T W_{R_f})$ is of full rank since in these cases, corrupted strips of C may be recovered by the host.

Theorem 1 *If only nodes in $\mathcal{P}$ fail, then t faults can be tolerated by the algorithm for multiple fault recovery*

Proof: If only nodes in $\mathcal{P}$ fail, then the set $\mathcal{F}_C$ is empty. Thus $W_R = W$ and therefore W_{R_f} for this fault pattern consists of ν_P columns of W, where $\nu_P \leq t$. By corollary 1 to Lemma 2, the columns of W_{R_f} are linearly independent. Thus, $(W_{R_f}^T W_{R_f})$ is of full rank and Eqn. (15) may be solved to recover all corrupted strips of C □

Theorem 2 *The algorithm for recovery from multiple faults can tolerate any pattern involving $2(\sqrt{t+1}-1)$ or fewer nodes*

Proof: Let a total of $\nu = \nu_P + \nu_C$ faults have occurred. Consider the matrix W_R which is constructed by deleting all rows in W corresponding to indices in the set $\mathcal{F}_C$. Since W has t rows, no matter which ν_C rows of W are deleted, W_R will contain at least $\frac{t-\nu_C}{(\nu_C+1)}$ consecutive rows which were also consecutive in W. (This occurs when the deleted rows are evenly spaced). Let the matrix formed by these consecutive rows be denoted by W'. By Lemma 2, every $\frac{t-\nu_C}{(\nu_C+1)}$ columns of W' are also linearly independent. Thus, if $\nu_P \leq \frac{t-\nu_C}{(\nu_C+1)}$, the ν_P columns of W_{R_f} are guaranteed to be linearly independent and thus W_{R_f} has rank ν_P. Hence the matrix $(W_{R_f}^T W_{R_f})$ is of full rank and Eqn. (15) may be solved by the host to recover the corrupted C_i's. Thus, in the event that ν_C check nodes have failed, the host can perform complete recovery provided fewer than $\frac{t-\nu_C}{(\nu_C+1)}$ computation nodes have failed, i.e., it can tolerate a total number of failed nodes $\nu \leq \nu_C + \frac{t-\nu_C}{(\nu_C+1)}$. Treating ν as a function of ν_C, we find that it possesses a minimum at $\nu_C = \sqrt{t+1}-1$ and the value of ν at this minimum is $2(\sqrt{t+1}-1)$. Thus any fault pattern involving $2(\sqrt{t+1}-1)$ or fewer nodes can be tolerated □

5.2 An example

We now show an example to make our methodology clearer. Let us assume that we have a 14 node system and we desire to locate 3 faults and correct 2. We partition our nodes into sets $\mathcal{P}$ and $\mathcal{C}$ such that $\mathcal{P} = \{0,1,2,3,4,5,6,7,8,9,10\}$ and $\mathcal{C} = \{11,12,13\}$. The next prime greater than 11 is 13 and 2 is a primitive element over $GF(13)$. So the matrix W constructed according to the rules described earlier in the section is as follows

$$W = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 3 & 6 & 12 & 11 & 9 & 5 & 10 \\ 1 & 4 & 3 & 12 & 9 & 10 & 1 & 4 & 3 & 12 & 9 \end{pmatrix} \tag{16}$$

It may be verified that any three columns W as shown above are linearly independent over the field of real numbers, as was proved earlier. The host then constructs S_0, S_1 and S_2 from the A_i's as follows and communicates these to check nodes 11,12 and 13 respectively.

$$\begin{aligned} S_0 &= A_0 + A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + \\ &\quad A_8 + A_9 + A_{10} \\ S_1 &= A_0 + 2A_1 + 4A_2 + 8A_3 + 3A_4 + 6A_5 + 12A_6 + \\ &\quad 11A_7 + 9A_8 + 5A_9 + 10A_{10} \\ S_2 &= A_0 + 4A_1 + 3A_2 + 12A_3 + 9A_4 + 10A_5 + A_6 + \\ &\quad 4A_7 + 3A_8 + 12A_9 + 9A_{10} \end{aligned} \tag{17}$$

Since we want triple fault location, we group the 14 nodes into two check groups of 7 nodes each. Each node keeps checksums on 3 other nodes within its check group where the nodes to be tested by each node are assigned according to the links of an optimal $D_{1,3}$ system. Fault location may then be performed using the algorithm in Section 4. We assume that following the production of C, nodes 0,1 and 12 are determined to be faulty. The host then constructs the following system of equations

$$\begin{aligned} C_0 + C_1 &= T_0 - (C_2 + C_3 + C_4 + C_5 + C_6 + C_7 + \\ &\quad C_8 + C_9 + C_{10}) \\ C_0 + 4C_1 &= T_2 - (3C_2 + 12C_3 + 9C_4 + 10C_5 + C_6 + \\ &\quad 4C_7 + 3C_8 + 12C_9 + 9C_{10}) \end{aligned} \tag{18}$$

where, as before, we have $T_j = S_j B$. We observe that the equations involving the unknowns C_0 and C_1 are linearly independent and the host is therefore able to solve the system (18) and recover C_0 and C_1 (We note here that the system of (18) involves two equations in two unknowns, so we may avoid the computations to generate the system of Eqn. (15) in this case). Our methodology guarantees that any pattern of 2 faults can be tolerated but it may be verified that, in fact, any pattern of 3 faults can also be tolerated for this example.

The asymptotic overheads of this scheme over the basic algorithm are $\frac{1}{p-t}$. Thus, if $t \ll p$, the overheads are small.

6 Fault Location and Recovery Schemes for Other Algorithms

In this section, we show how to modify two other parallel algorithms, QR factorization and Gaussian elimination, for single fault location and recovery. From the discussion, it will be clear that the approach in each case is similar to that for matrix multiplication already detailed in earlier sections. However, due to space constraints, we only present an outline the modifications necessary.

6.1 QR factorization

The problem of QR Factorization is to factorize a given nxn matrix A into the product of two nxn matrices Q and R such that Q is orthogonal and R is upper triangular. This is typically achieved by successive premultiplication of A by a series of orthogonal matrices which zero out selected elements of A, a process known as orthogonal transformation. Two common orthogonal transformation techniques are Givens transforms and Householder transforms [16]. A Givens transform can be used to zero out a single element of the matrix A at every step, while by using a Householder transform one may zero out a series of consecutive elements in a single column at every step.

A parallel algorithm based on Householder transformations was developed which initially distributed the nxn matrix A in a column cyclic manner over all the participating processors. Initially, Q is the identity matrix and this is also distributed in a column cyclic manner. At the ith step the matrix A is transformed to introduce zeroes below the diagonal of the ith column. The same transformation is applied to Q. At the end of n steps A is transformed into the required upper triangular matrix R and Q becomes the final orthogonal matrix.

A property of orthogonal transforms such as Householder or Givens transforms is that they preserve the 2-norms of the matrix undergoing the transformation. This property is used as the basis for developing a fault-tolerant version of the QR algorithm.

In order to devise a version of the QR algorithm for single fault location, we may group the set of available nodes into disjoint check group rings of 3 nodes each as discussed in Section 2. Each node is then assigned to compute the sum-of-squares (square of the 2-norm) of the columns of Q and A owned by the node following it in its check group ring. At the end of the QR factorization algorithm, each node checks whether of the sum-of-squares of the columns of R and Q owned by the node following it in its check group ring equals its own sum-of-squares computed prior to the start of the QR algorithm. In the absence of faults in either node, the sum-of-squares values should be equal to within a tolerance. Also, the sum-of-squares of the ith column is checked prior to the ith Householder update. If the check fails, algorithm is terminated without any further Householder transforms and the final check phase, where each node checks the sum-of-squares of columns owned by the node following it in its check group ring, is performed to obtain a syndrome to identify the faulty node.

The modification of the above algorithm to locate multiple faulty nodes is straightforward and uses the ideas developed in Section 4.

The QR algorithm was modified to recover from errors introduced by a single faulty node in a manner paralleling that of the matrix multiplication algorithm for single fault recovery, by creating an extra check strip for both A and Q, each containing $\frac{n}{p-1}$ columns computed by adding columns $j, j+\frac{n}{p-1}, j+2\frac{n}{p-1}, \ldots, j+(p-2)\frac{n}{p-1}$ of A and Q to obtain column j of the check strip of A and Q $(1 \leq j \leq \frac{n}{p-1})$. One node is then designated as a check node in charge of updating the check strips as in the matrix multiplication algorithm. In case of a fault, the host is able to recover the strips of A and Q owned by the faulty node by simply subtracting off the remaining strips of A and Q from the checksum strips owned by the check node. The recovered data is sent to the check node which takes over the computations of the faulty node for the rest of the algorithm.

Again, using the ideas discussed Section 5, the derivation of a QR algorithm incorporating recovery from multiple faults is straightforward.

6.2 Gaussian elimination

The solution of a set of linear equations of the form $Ax = b$ for a single right hand side b is usually obtained by performing Gaussian elimination to create an upper triangular matrix. The system is then easily solved by back substitution from the last unknown upwards.

A parallel algorithm for Gaussian elimination was developed which initially distributed A row-wise in a cyclic fashion on the available nodes.

The basic algorithm (incorporating no error detection features) proceeds as follows. For a system of size n there are $n-1$ iterations in the algorithm from 1 to $n-1$, during each of which one more column of 0's is introduced into matrix A. Iteration k of the algorithm results in the update of an $(n-k)$ by $(n-k)$ submatrix in parallel. If row pivoting is used, at the start of step k, the row with the largest pivot element is swapped with the kth row. All nodes may cooperate to determine the pivot by searching for the largest candidate row pivot in a distributed manner. The swapping of the pivot row with the kth row can then be achieved by an exchange of messages between the nodes holding the pivot row and the kth row. However, for our parallel implementation of the algorithm, we implemented a restricted form of pivoting in which the pivot row for the kth iteration is chosen from amongst the rows owned by the owner of the kth row. This eliminates the large number of messages needed to implement row pivoting over all the rows and also simplifies the development of a fault-tolerant version of the algorithm. The pivot row (now the kth row) is then communicated to all nodes. Upon receiving the pivot row, each node can update the rows of the $(n-k)$ by $(n-k)$ submatrix which are owned by it.

Row checksums may be introduced into the basic algorithm to incorporate fault-tolerance. (Details of the update equations for the row checksums may be found in [17]). However, the equations for updating the row checksum variables in the ith iteration require the elements in the ith column. In order to check for the integrity of these elements before they are used in the update of the row checksum elements in the ith iteration, a check is performed on the ith column sum. Thus for Gaussian elimination, not only do we need to introduce row checksums for every row, but we also need to introduce column checksums for every column.

The version of Gaussian elimination developed for single fault location again uses the ideas of grouping nodes into disjoint check group rings of 3 nodes each. Each node maintains the row checksum of the node following it in its check group. Also, column checksums are maintained for each column on node 0. Prior to performing the ith elimination step, the owner of row i swaps the ith row with the pivot row (As mentioned earlier, the pivot row is chosen by the owner of row i from amongst its own rows). The corresponding checksums are swapped on the node owning the checksums for the ith row and the pivot row. The pivot row is communicated to all nodes to enable them to perform the ith elimination step. The pivot row sum is checked on the node owning the checksum of the pivot row. only if this check passes do all nodes go ahead with the ith elimination step. (Note the similarity of this step to the sum-of-squares check of the ith column prior to the ith Householder update for the QR algorithm). If the check fails, the algorithm terminates without performing further elimination steps and a row checksum check is performed on all the rows to determine the syndrome for locating the faulty node as was the case for the QR algorithm. Node 0 updates the row of column checksums at the end of each iteration. Note that even if node 0 incorrectly declares the column checksum test to have passed due to a fault in the node, this does not affect the correct execution of the algorithm, since our assumption of a single faulty node implies that only the elements of node 0 may be erroneous. The ith column elements of node 0 are only used to update the row checksum elements owned by the node preceding it in its check group. This may cause the row checksum elements to take on incorrect values, causing the row sum checks involving node 0's rows to fail at some later point. However, the syndrome still identifies node 0 as the faulty node since only the node checking node 0's rows declares an error.

From the above discussion, it is clear that the steps to developing a single fault locating parallel Gaussian elimination algorithm use the same ideas which were developed in Section 2. The ideas in Section 4 may be used to derive a version of the algorithm which is able to locate multiple faulty nodes.

A version of parallel Gaussian elimination for single-fault recovery may be derived using ideas very similar to those in Section 3 and in the single-fault-tolerant parallel QR algorithm. As before, one node is designated as the check node. The matrix A is distributed in a row cyclic manner over the remaining $p-1$ nodes. A check strip is computed by adding rows $j, j+\frac{n}{p-1}, j+2\frac{n}{p-1}, \ldots, j+(p-2)\frac{n}{p-1}$ to obtain row j of the check strip $(1 \leq j \leq \frac{n}{p-1})$. A data strip corrupted by a faulty node may be recovered by subtracting off the good data strips from the checksum strip. The recovered data strip is then sent to the check node, which takes over the computations of the faulty node for the remaining iterations of the algorithm.

A multiple-fault-tolerant version of Gaussian elimination may be developed by using the ideas developed in Section 5.

7 Experimental Results

As noted earlier, ABFT schemes are attractive since they can be implemented on general purpose multiprocessors without requiring extra hardware or harware modifications. In this section we present performance overheads for the algorithm with single fault location and the algorithm with single fault recovery over the basic algorithm for the applications discussed in the earlier sections, viz., matrix multiplication, as well as for two other algorithms, QR factorization, and Gaussian elimination. The testbed used was a 16 node Intel iPSC/2 hypercube. Our encodings guarantee that single faults can be located or recovered from (depending on which algorithm is being run) in the event that the faults are detected in the first place. Non-detection of faults can occur if the magnitude of the data corruption due to the fault is so small that the deviation from the nonfaulty results is not greater than the tolerance. So, in a separate subsection we give error detection results and a short outline of the tolerance determination methodology.

7.1 Timing overheads

Fig. 1 shows the timing overheads for the algorithms for single fault location and correction over the basic algorithms on various matrix dimensions. The overhead for the algorithms for single fault correction is shown for two cases - the case when no fault occurred, when only a diagnosis of the system state had to be performed and the case when a single fault was actually injected, when, following the diagnosis of the faulty node the data was recovered in the manner described in Section 2. As expected, the overhead for the single fault location algorithm becomes very low for large matrix sizes. Overheads for the single fault location algorithm asymptotically tend to 0 with increased matrix size. Overheads for the single fault correction algorithm are somewhat higher but are also very modest for moderately large matrix sizes. Asymptotically the overhead tends to $\frac{1}{p-1}$ for a p node system. Since we used a 16 node system, the overhead for the single fault correction algorithm can be expected to approach 7% for large matrix sizes. We observe from Fig. 1 that even for the modest matrix sizes we obtained timing results on, the overhead for the single fault correction algorithm is only around 10-15%. Another point to note is that there is very little extra overhead for the recovery phase, especially for large matrix sizes. This is not surprising since the fractional overhead contributed by the recovery phase decreases linearly with n, the matrix dimension.

7.2 Fault coverage

In order to determine fault coverage results, we injected transient and permanent bit- and word-level faults in floating point computations and computed the percentage of times these were detected. We determined the threshold to be used by the simplified error analysis method suggested in [17]. Since most of the undetected faults were not detected due to their very marginal effects on the data (such as affecting the least significant 5 bits in a 23-bit mantissa) we determined new coverage results for faults causing errors we classified as ***significant errors*** which may be loosely defined as errors which caused a deviation from the correct results by 10 times the deviation caused by roundoff on a fault-free system. For details of the definitions and exact methodology of determining error coverage we refer the reader to [17].

The fault coverage results are summarized in Table 1. The coverages for location and recovery are reported for only those cases when the injected faults were detected.

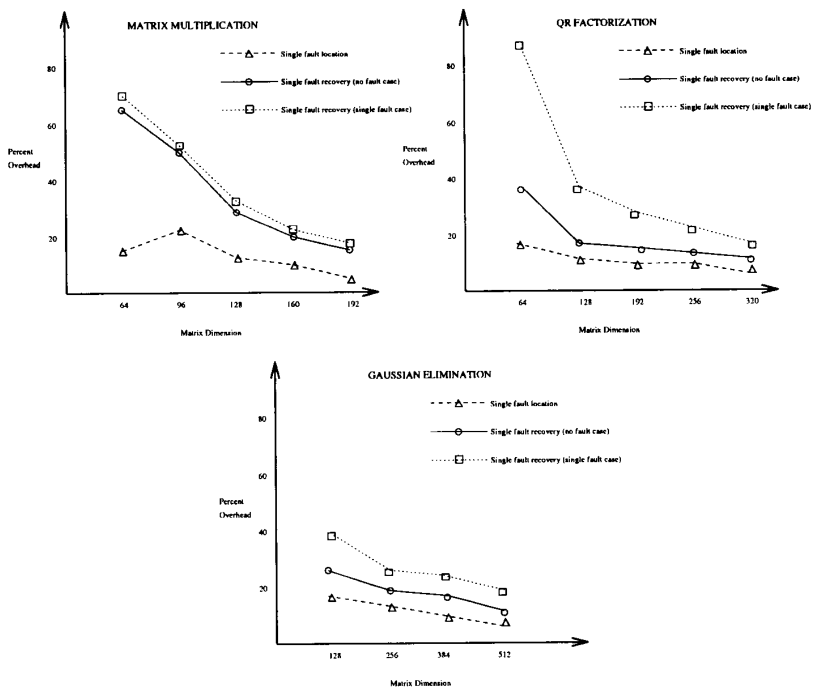

Figure 1: Timing overheads for single fault location and correction algorithms

Table 1: Fault detection, location and recovery coverage

Fault Types	Error Detection Coverage			Significant Error Coverage			Fault Location Coverage			Fault Recovery Coverage		
	MM	QR	GE	MM	QR	GE	MM	QR	GE	MM	QR	GE
Transient Bit-level	73	70	86	94	90	100	100	100	98	100	100	100
Transient Word-level	100	100	100	100	100	100	100	100	98	100	100	100
Permanent Bit-level	78	74	93	95	86	99	100	100	100	100	100	100
Permanent Word-level	100	100	100	100	100	100	100	100	100	100	100	100

Note that in some cases, the error location coverage is less than 100%. This can occur due to a transient fault corrupting a checksum, rather than any of the original data elements, so that the faulty processor flags an error but its check processor does not, since all the data elements communicated to it are error free. In these cases, we may incorrectly flag as faulty the processor being checked by the faulty processor, but since the data computed by the faulty processor is error-free the recovery phase goes through correctly and we may restart the computation from the point of failure. Error coverage results for the fault-recovery algorithms are always 100% since our algorithms for reconstructing the data yield correct results if the data involved in the reconstruction is fault-free.

8 Conclusions

In this paper we have developed algorithm-based schemes for fault location and recovery and illustrated their application to three parallel matrix algorithms. We have illustrated the methodology for fault location and recovery by discussing how a parallel matrix multiplication algorithm may be modified for fault location and recovery, but the methodology is sufficiently general to be applied to other algorithms involving matrix manipulations. To illustrate this fact, we have presented results on two other parallel matrix algorithms, QR factorization and Gaussian elimination, which have been modified using the methods discussed here to make them tolerant to single processor faults. Upto this point, algorithm-based schemes for general purpose multicomputers had only been reported for error detection. The fault-tolerant encodings can be expected to offer fault tolerance at a small overhead over the simple algorithm involving no fault tolerance. An ABFT design scheme for forward error recovery from multiple faults has been presented for the first time in our paper. Unlike previously reported design schemes for ABFT, we have experimentally evaluated the efficiency of our schemes on real applications. We have presented actual experimentally determined timing overheads for our encoding of the single fault location and correction algorithms for three applications and have shown that the overheads are small, especially for moderately large problem sizes. We believe that the ideas in this paper can be used to develop fault-locating and correcting algorithms for a large class of numerical applications.

References

[1] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, pp. 518–528, June 1984.

[2] J.-Y. Jou and J. A. Abraham, "Fault-tolerant matrix operations on multiple processor systems using weighted checksums," *SPIE Proceedings*, vol. 495, August 1984.

[3] P. Banerjee, J. T. Rahmeh, C. Stunkel, V. S. Nair, K. Roy, V. Balasubramanian, and J. A. Abraham, "Algorithm-based fault tolerance on a hypercube multiprocessor," *IEEE Trans. Comput.*, vol. 39, pp. 1132–1145, September 1990.

[4] Y.-H. Choi and M. Malek, "A fault-tolerant fft processor," *IEEE Trans. on Computers*, vol. 37, pp. 617–621, May 1988.

[5] P. Banerjee and J. A. Abraham, "Bounds on algorithm-based fault tolerance in multiple processor systems," *IEEE Trans. Comput.*, vol. C-35, pp. 296–306, April 1986.

[6] D. Gu, D. J. Rosenkrantz, and S. S. Ravi, "Design and analysis of test schemes for algorithm-based fault tolerance," *Proc. 20th Int. Symp. Fault Tolerant Comput.*, pp. 106–113, June 1990.

[7] B. Vinnakota and N. K. Jha, "A dependence graph based approach to the design of algorithm-based fault tolerant systems," *Proc. 20st Int. Symp. Fault Tolerant Comput.*, June 1990.

[8] B. Vinnakota and N. K. Jha, "Design of multiprocessor systems for concurrent error detection and fault diagnosis," *Proc. 21st Int. Symp. Fault Tolerant Comput.*, June 1991.

[9] J.-Y. Jou and J. A. Abraham, "Fault-tolerant matrix operations on multiple processor systems using weighted checksums," *SPIE Proceedings*, vol. 495, August 1984.

[10] A. Roy-Chowdhury and P. Banerjee, "Tolerance determination for algorithm-based schemes based on simplified error analysis techniques," *Proc. 23rd Int. Symp. on Fault Tolerant Comput.*, June 1993.

[11] F. P. Preparata, G. Metze, and R. T. Chien, "On the connection assignment problem of diagnosable systems," *IEEE Trans. on Electronic Computers*, vol. EC-16, pp. 848–854, December 1967.

[12] D. K. Pradhan, *Fault-Tolerant Computing: Theory and Techniques, Vol. II.* Englewood Cliffs, NJ: Prentice Hall, 1986.

[13] V. S. S. Nair and J. A. Abraham, "Real number codes for fault-tolerant matrix operations on processor arrays," *IEEE Trans. on Computers*, vol. 39, pp. 426–435, April 1990.

[14] R. E. Blahut, *Theory and Practice of Error Control Codes.* Reading, MA: Addison-Wesley, 1984.

[15] G. Strang, *Linear Algebra and its Applications.* San Diego: Harcourt Brace Jovanovich, Publishers, 1988.

[16] G. H. Golub and C. F. V. Loan, *Matrix Computations.* Baltimore: Johns Hopkins University Press, 1987.

[17] A. Roy-Chowdhury, "*Evaluation of Algorithm Based Fault-Tolerance Techniques on Multiple Fault Classes in the Presence of Finite Precision Arithmetic.*" M.S. Thesis, Univ. of Illinois, Urbana-Champaign, August 1992. Tech. Report no. CRHC-92-15, UILU-ENG-92-2228.