EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

A framework for ABFT techniques in the design of fault-tolerant computing systems

Hodjat Hamidi*, Abbas Vafaei and Seyed Amirhassan Monadjemi

Abstract

We present a framework for algorithm-based fault tolerance (ABFT) methods in the design of fault tolerant computing systems. The ABFT error detection technique relies on the comparison of parity values computed in two ways. The parallel processing of input parity values produce output parity values comparable with parity values regenerated from the original processed outputs. Number data processing errors are detected by comparing parity values associated with a convolution code. This article proposes a new computing paradigm to provide fault tolerance for numerical algorithms. The data processing system is protected through parity values defined by a high-rate real convolution code. Parity comparisons provide error detection, while output data correction is affected by a decoding method that includes both round-off error and computer-induced errors. To use ABFT methods efficiently, a systematic form is desirable. A class of burst-correcting convolution codes will be investigated. The purpose is to describe new protection techniques that are easily combined with data processing methods, leading to more effective fault tolerance.

Keywords: algorithm-based fault tolerance (ABFT), burst-correcting convolution codes, parity values, syndrome

1. Introduction

Algorithm-based fault tolerance (ABFT) was first introduced by Huang and Abraham [1] and was directed toward detection of high-level errors because of internal processing failures. ABFT techniques are most effective when employing a systematic form [2-6]. The motivational model basic ABFT as applied to data processing of blocks of real data is shown in Figures 1 and 2. The ABFT philosophy leads directly to a model from which error correction can be developed. The parity values are determined according to a systematic real convolution code. Detection relies on two sets of parity values which are computed in two different ways, one set from the input data but with a simplified combined processing subsystem, and the other set directly from the output processed data, employing the parity definitions directly. These comparable sets will be very close numerically, although not identical because of round-off error differences between the two parity generation processes. The effects of internal failures and round-off error are modeled by additive error sources located at the output of the processing block and input at threshold detector. This model combines the aggregate effects of errors and failures and applies them to the respective outputs. ABFT for arithmetic and numerical processing operations is based on linear codes. Bosilca et al. [7] proposed a new ABFT method based on parity check coding for high-performance computing. The application of low density parity check (LDPC) based ABFT is compared and analyzed in [8], as the use of LDPC to classical Reed-Solomon (RS) codes with respect to different fault models. However, Roche et al. [8] did not provide a method for constructing LDPC codes algebraically and systematically, such as RS and BCH codes are constructed, and LDPC encoding is very complex because of the lack of appropriate structure. ABFT methodologies used in [9] present parity values dictated by a real convolution code for protecting linear processing systems.

A class of high rate burst-correcting convolution codes is discussed in [10]. Convolution codes provide error detection in a continuous manner using the same computational resources as the algorithm progresses. Redinbo [11] presented a method to wavelet codes into systematic forms for ABFT applications. This method applies high-rate, low-redundancy wavelet codes which

* Correspondence: hamidi@eng.ui.ac.ir
Department of Computer Science, University of Isfahan, Post Code 81746-73441, Isfahan, Iran

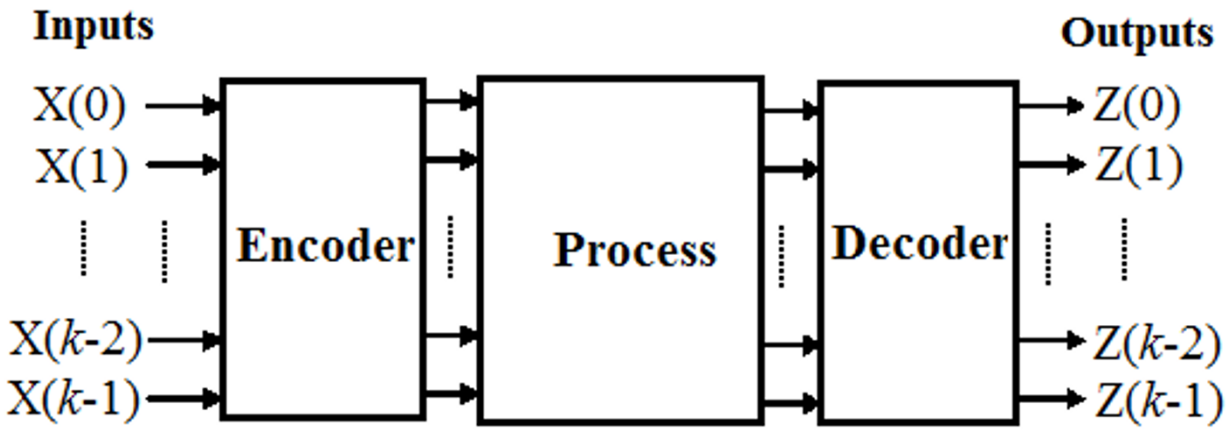

Figure 1 General architecture of ABFT [13].

use continuous checking attributes for detecting the onset of errors. However, this technique is suited to image processing and data compression applications. In addition, there is a difficult analytical approach to accurate the measures of the detection performances of the ABFT technique using wavelet codes [11,12].

Figure 1, [13], shows the basic architecture of an ABFT system. Existing techniques use various coding schemes to provide information redundancy needed for error detection and correction. The coding algorithm is closely related to the running process and is often defined by real number codes generally the block types [14]. Systematic codes are of most interest because the fault detection scheme can be superimposed on the original process box with the least changes in the algorithm and architecture. The goal is to describe new protection techniques that are easily combined with normal data processing methods, leading to more effective fault tolerance. The data processing system is protected through parity sequences specified by a high rate real convolution code. Parity comparisons provide error detection, while output data correction are affected by a decoding method that includes both round-off error and computer-induced errors. The error detection structures are developed and they not only detected subsystem errors, but also corrected errors introduced in the data processing system. Concurrent parity values' techniques are very useful in detecting numerical error in the data

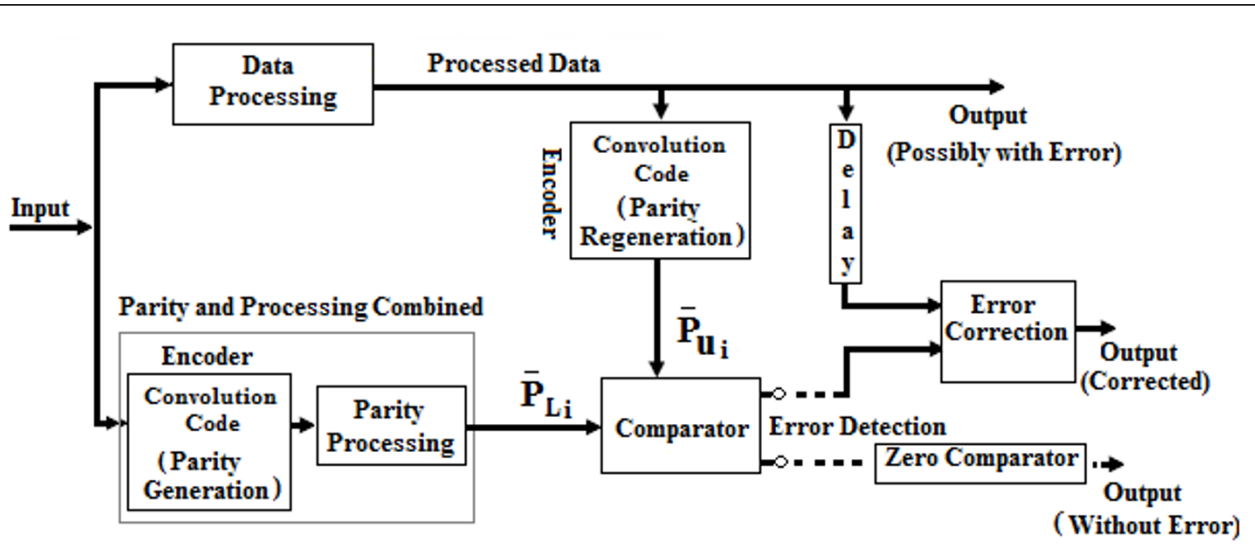

Figure 2 Block diagram of the ABFT technique.

processing operations, where a single error can propagate to many output errors.

The following contributions are made in this article: In Section 2, the convolution codes are discussed briefly; in Section 3, the architecture of ABFT (ABFT scheme) and modeling errors are proposed and the method for detecting errors using parity values is discussed; in Section 4, the class of convolution codes: burst-error-correcting convolution codes is discussed; in Section 5, the decoding and corrector system is discussed; in Section 6, the results and evaluations and simulations are presented and finally in Section 7, conclusions are presented.

2. Convolution codes

A convolution code is an error correcting code that processes information *serially* or, continuously, in short block lengths [15-21]. A convolution encoder has *memory*, in the sense that the output symbols depend not only on the input symbols, but also on previous inputs and/or outputs. In other words, the encoder is a *sequential circuit* [15,17,20]. A rate $R = k/n$ convolution encoder with memory order m can be realized as a k-input, n-output linear sequential circuit with input memory order m; that is, inputs remain in the encoder for an additional m time units after entering. Typically, n and k are small integers, $k < n$, the information sequence is divided into blocks of length k, and the codeword is divided into blocks of length n. In the important special case, when $k = 1$, the information sequence is not divided into blocks and is processed continuously. Unlike with block codes, large minimum distances and low error probabilities are achieved not by increasing k and n but by increasing the memory order m [16, Chapter 11]. We consider only systematic forms of convolution codes because the normal operation of Process block is not altered and there is no need to decoding for obtaining true outputs. A systematic real convolution code guarantees that faults representing errors in the processed data will result in notable non-zero values in syndrome sequence. Systematic encoding means that the information bits always appear in the first k positions of a code word, leftmost. The remaining $n - k$ bits in a code word are a function of the information bits, and provide redundancy that can be used for error correction and/or detection purposes, rightmost. Real number convolution codes may find applications in channel coding for communication systems and in fault-tolerant data processing systems containing error correction. Real-number codes can be constructed easily from finite-field codes, viewing the field elements as corresponding integers in the real number field, and as such theoretically have as good if not better properties as the original finite field structures [6].

3. Code usage for ABFT and ABFT scheme

3.1. Code usage for ABFT

A real convolution code in systematic form [16] is used to compute parity values associated with the processing outputs as shown in Figure 2. Certain classes of errors occurring anywhere in the overall system including the parity generation and regeneration subsystems are easily detected. A convolution code with its encoding memory can sense the onset of errors before they increase beyond detection limits. For a rate k/n real convolution code with constraint parameter, it is always possible by simple linear operations to extract the parity generating part. The $(n - k)$ parity samples for each processed block of samples are produced in block processing fashion. Since processing resources are in close proximity, it is easily demonstrated [9] that an efficient block processing structure can produce the $(n - k)$ parity values directly from the inputs. When these two comparable parity values are subtracted, one from the outputs and the others directly from the inputs, only the stochastic effects remain, and the syndromes are produced as shown in Figure 2.

3.2. Modeling errors

It is generally assumed that transient errors can occur in the intermediate values at any time during the course of data processing as shown in Figure 3. Furthermore, only one error is permitted during a sequence of operations to avoid complete overload. The proposed error model implies that errors are described by adding a modeling numerical value e to the calculated output: $z = y + e$.

3.3. ABFT scheme

To achieve fault detection and correction properties of convolution code in a linear process with the minimum overhead computations, the architecture is proposed in Figure 2. For error correction purposes, redundancy must be inserted in some form and convolution parity codes will be employed, using the ABFT. A systematic form of convolution codes is especially profitable in the ABFT detection plan because no redundant transformations are needed to achieve the processed data after the detection operations. Figure 2 summarizes an ABFT technique employing a systematic convolution code to

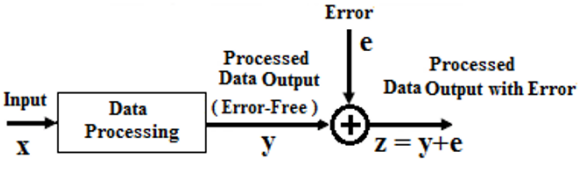

Figure 3 Modeling errors in data processing.

define the parity values. The data processing operations are combined with the parity generating function to provide one set of parity values. The k is the basic block size of the input data, and n is the block size of the output data, new data samples are accepted and (n - k) new parity values are produced.

The upper way, Figure 2, is the processed data flow which passes through the process block (data processing block) and then feeds the convolution encoder (parity regeneration) to produce parity values. On the other hand, the comparable parity values are generated efficiently and directly from the inputs (parity and processing combined, see Figure 2), without producing the original outputs. The difference in the comparable two parity values, which are computed in different ways, is called the syndrome; the syndrome sequence is a stream of zero or near zero values. The convolution code's structure is designed to produce distinct syndromes for a large class of errors appearing in the processing outputs. Figure 2 employs convolution code parity in detecting and correcting processing errors.

3.4. Error detection

The method for detecting errors using parity values is shown in Figure 2. Except for small round-off errors, the two parity values $\bar{p}_{u_i}$ and $\bar{p}_{l_i}$ should be equal in the error-free case. The two parities are equal if an error does not occur, ignoring any round-off errors in the arithmetic computations. The comparator computes the difference, S, between the two parity values and determines if its magnitude is smaller than a chosen threshold determined by round-off error, $S = \bar{p}_{l_i} - \bar{p}_{u_i}$ if $|S| < \tau$ then there is no error (τ is threshold). The difference between the parity values, considering a round-off threshold, τ, can be used to detect a error. This threshold τ places a bound on the effects of errors appearing at the output, modeled here as a vector e which is added to the true output y to characterize the observed output $z = y + e$, see Figure 3. A total self-checking checker (comparator) for real number parities using a detection threshold is described in [9,11]. Its role is to indicate if an error has occurred in the process using the parities $\bar{p}_{l_i}$ and $\bar{p}_{u_i}$. The comparator is constructed by producing a 1-out-of-2 codeword at terminals (sign threshold, banded thresholds) = (T_{SGN}, T_τ) as shown in Figure 4. Given that s truly represents $\bar{p}_{l_i} - \bar{p}_{u_i}$, if either $|S| \geq \tau$, the sign, or the value-characterize unit has failed when valid parity inputs are applied, the output will not be a valid 1-out-of-2 code. Otherwise, the comparator and its checking parts give a 1-out-of-2 code indicating that no error has occurred in the data processing unit and its checking facilities. The precision required for the two parity values, the value characterizations in Figure 4, only need to meet the separation by the threshold value to be effective for detection.

4. Burst-error-correcting convolution codes

A burst of length d is defined as a vector and the non-zero components are confined to d consecutive digit positions, the first and last are non-zero [16,17]. A burst refers to a group of possibly contiguous errors which is characteristic of unforeseeable effects of errors in data computation. Only systematic forms of convolution codes are considered here because the normal operation of Process block has not changed and there is no need for decoding to obtain true outputs. Moreover, convolution codes have good correcting characteristics because of memory in their encoding structure [17].

4.1. Bounds on burst-error-correcting convolution codes

Costello and Lin [16] have shown that a sequence of error bits e_{d+1}, e_{d+2} , ..., e_{d+a} is called a burst of length a relative to a guard space of length b if

1. $e_{d+1} = e_{d+a} = 1$;
2. the b bits preceding e_{d+1} and the b bits following e_{d+a} are zero;
3. the a bits from e_{d+1} through e_{d+a} contain no subsequence of b zero.

For any convolution code of rate R that corrects all bursts of length a or less relative to a guard space of length b,

$$\frac{b}{a} \geq \frac{1+R}{1-R}. \tag{1}$$

The bound of (1) is known as the bound on complete burst-error correction [16]. Massey [20] has also shown

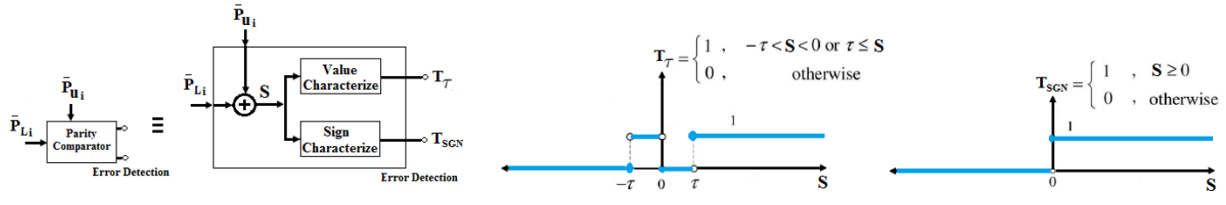

Figure 4 Comparator using threshold τ [11].

that if we allow a small fraction of the bursts of length a to be decoded incorrectly, the guard space requirements can be reduced significantly. In particular, for a convolution code of rate R that corrects all but a fraction ψ of bursts of length a or less relative to a guard space of length b

$$\frac{b}{a} \geq \frac{R + [\log_2(1-\psi)]/a}{1-R} \approx \frac{R}{1-R} \tag{2}$$

for small ψ. The bound of (2) is known as the bound on *almost all* burst-error correction. Burst-correcting convolution codes at structure of the convolution codes are appropriate and efficient in detecting and correcting errors from internal computing failures. Burst-correcting convolution codes need guard bands (error-free regions) before and after bursts of errors, particularly if error correction is needed [16]. One class of burst-correcting codes is the Berlekamp-Preparata (BP) codes [16-20] that have many appropriate characteristic with regard to failure error-detecting. Their design properties guarantee for detecting the onset of errors because of failures, regardless of any error-free region following the beginning of a burst of errors. Consider designing an (n, k = n - 1, m) systematic convolution encoder to correct a phased burst error confined to a single block of n bits relative to a guard space of m error-free blocks. To design such a code, we must assure that each correctable error value $[E]_m = [e_0, e_1, ..., e_m]$ results in a distinct syndrome $[S]_m = [s_0, s_1, ..., s_m]$. This implies that each error value with $e_0 \neq 0$ and $e_d = 0$, d = 1, 2, ..., m must yield a distinct syndrome and that each of these syndromes must be distinct from the syndrome caused by any error value with $e_0 = 0$ and a single block $e_d \neq 0$, d = 1, 2, ..., m. Therefore, the first error block e_0 can correctly be decoded if first (m + 1) blocks of e contain at most one non-zero block, and assuming feedback decoding, each successive error block can be decoded in the same way. An (n, k = n - 1, m) systematic code is depicted by the set of generator polynomials $g_1^{(n-1)}(D)$, $g_2^{(n-1)}(D)$, ..., $g_{n-1}^{(n-1)}(D)$. The generator matrix of a systematic convolution code, G, is a semi-finite matrix evolving m finite sub-matrixes as

$$G = \begin{bmatrix} IP_0 & 0P_1 & 0P_2 & \dots & 0P_m & & \\ & IP_0 & 0P_1 & \dots & 0P_{m-1} & 0P_m & \\ & & IP_0 & \dots & 0P_{m-2} & 0P_{m-1} & 0P_m \\ & & & \ddots & & \ddots & \\ & & & & \ddots & & \ddots \end{bmatrix} \tag{3}$$

where I and 0 are identity and all zero $k \times k$ matrixes, respectively, and P_i with i = 0 to m is a $k \times (n - k)$ matrix [18]. The parity-check matrix is constructed from a basic binary matrix, labeled H_0, a $2n \times n$ binary matrix containing the skew-identity matrix in its top n rows (4).

$$H_m = [H_0, H_1, ..., H_m] \tag{4}$$

where H_0 is an $n \times (m + 1)$ matrix (5):

$$H_0 = \begin{bmatrix} g_{1,0}^{(n-1)} & g_{1,1}^{(n-1)} & \cdots & g_{1,m}^{(n-1)} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ g_{n-1,0}^{(n-1)} & g_{n-1,1}^{(n-1)} & & g_{n-1,m}^{(n-1)} \\ 1 & 0 & \dots & 0 \end{bmatrix} \tag{5}$$

For $0 < d \leq m$, we obtain H_d from H_{d-1} by shifting H_{d-1} one column to the right and deleting the last column. Mathematically, this operation can be expressed as

$$H_0 = \begin{bmatrix} 0\ 1\ 0\ \dots\ 0 \\ 0\ 0\ 1\ \dots\ 0 \\ \vdots \\ \vdots \\ \vdots \\ 0\ 0\ 0\ \dots\ 1 \\ 0\ 0\ 0\ \dots\ 0 \end{bmatrix} = H_{d-1}T \tag{6}$$

where T is an $(m + 1) \times (m + 1)$ shifting matrix. Another important parity check type of matrix is put together using H_0 and its d successive downward shifted versions [19]. However, all necessary information for forming the systematic parity check matrix H^{T} is contained in the basis matrix H_0. The lower triangular part of this matrix, (n - 1) rows, (n - 1) columns, hold binary values selected by a construction method to produce desirable detection and correction properties [19]. For systematic codes, the parity check matrix submatrices H_m in (4) have special forms that control how these equations are formed.

$$H_0^T = \left[P_0 | I_{n-k}\right], \quad H_i^T = \left[P_i | 0_{n-k}\right] \quad i = 1, 2, \dots L. \tag{7}$$

where I_{n-k} and 0_{n-k} are identity and all zero $k \times k$ matrixes, respectively, and P_i is an $(n - 1) \times k$ matrix. However, in an alternate view, the respective rows of H_0 contain the parity submatrices P_i needed in
H^{T}, (4) and (7):

$$H_0 = \begin{bmatrix} P_0 & \mid \mathrm{I}_1 \\ P_1 & \mid 0 \\ P_2 & \mid 0 \\ \vdots & \vdots \\ \vdots & \vdots \\ \vdots & \vdots \\ P_{L-1} & \mid 0 \\ \mathrm{P_L} & \mid 0 \end{bmatrix} \tag{8}$$

The n columns of H_0 are designed as an n-dimensional subspace of a full $(2n)$-dimensional space comparable with the size of the row space. Using this notation, the syndrome

$$[S]_m = [E]_m\left[H^T\right]_m = e_0H_0 + e_1H_1 + \ldots + e_mH_m = e_0H_0 + e_1H_0T + \ldots + e_mH_0T^m = \begin{bmatrix} S_i \\ S_{i+1} \\ \cdot \\ \cdot \\ \cdot \\ S_{i+n} \end{bmatrix} \quad (9)$$

$[S]_m$ is a syndrome vector with $(l+1)$ values, in this class of codes $(n - k)$ equal 1. The design properties of this class of codes assure any contribution of errors in one observed vector, $[E]_m$, appearing in syndrome vector $[S]_m$ is linearly independent of syndromes caused by ensuing error vectors $[E]_{i+1}$, $[E]_{i+2}$, ..., $[E]_{i+1}$ in adjacent observed vectors. At any time, a single burst of errors is limited to set $[E]_m$, correction is possible by separating the error effects. These errors in $[E]_m$ are recognized with the top n items in $[S]_m$.

$$[E]_m = \begin{bmatrix} e_{i,1} \\ e_{i,2} \\ \cdot \\ \cdot \\ \cdot \\ e_{i,n} \end{bmatrix} \quad (10)$$

then error values recognition

$$e_{i,n} = S_i, \quad e_{i,n-1} = S_{i+1}, \ldots, e_{i,1} = S_{i+n+1} \quad (11)$$

If there are non-zero error bursts in $[E]_{i+1}$, $[E]_{i+2}$, ..., $[E]_{i+l}$, their accumulate contribution is in a separate subspace never permitting the syndrome vector $[S]_m$ to be all zeros. The beginning of errors, even if they overwhelm the correcting capability of the code, can be detected. This distinction between correctable and only detectable error bursts is achieved by applying an annihilating matrix, denoted F_0^T, which is $n \times 2n$ and has a defining property, $F_0^T H_0 = 0_n$. Hence, it is possible to check whether a syndrome vector $[S]_m$ represents correctable errors, F_0^T. $[S]_m = 0$, then $[S]_m$ obtain correctable model. From (1) for an optimum burst-error correcting code, $b/a = (1 + R)/(1 - R)$. For the preceding case with $R = (n - 1)/n$ and $b = m.n = m.a$, this implies that

$$\frac{b}{a} = m = 2n - 1 \quad (12)$$

i.e., H_0 is an $n \times 2n$ matrix. We must choose H_0 such that the conditions for burst-error correction are satisfied. If we choose the first n columns of H_0 to be the skewed $n \times n$ identity matrix, then (9) implies that each error sequence with $e_0 \neq 0$ and $e_d = 0$, $d = 1, 2, \ldots, m$ will yield a distinct syndrome. In this case, we obtain the estimate of simply by reversing the first n bits in the $2n$-bit syndrome. In addition, for each $e_0 \neq 0$, the condition

$$e_0H_0 \neq e_dH_0T^d, d = 1, 2, \ldots, m, \quad (13)$$

must be satisfied for $e_d \neq 0$. This ensures that an error in some other blocks will not be confused for an error in block zero. For any $e_d \neq 0$ and $d \geq n$, the first n positions in the vector $e_d H_0T^d$ must be zero, since T^d shifts H_0 such that H_0T^d has all zero in its first d columns; however, for any $e_d \neq 0$, the vector cannot have all zeros in its first positions. Hence, condition (13) is automatically satisfied for $n \leq d \leq m$, $m = 2n$ -1, and we replace (13) with the condition that for each $e_0 \neq 0$,

$$e_0H_0 \neq e_dH_0T^d, d = 1, 2, \ldots, n - 1 \quad (14)$$

5. Decoding and corrector system

The BP codes can be decoded using a general decoding technique for burst-error-correcting convolution codes according to Massey [20]. We recall from (9) that the set of possible syndromes for a burst confined to block 0 is simply the row space of the $n \times 2n$ matrix H_0. Hence, $e_0 \neq 0$ and $e_d = 0$, $d = 1, 2, \ldots, m$ $[S]_m$ are codeword in the $(2n, n)$ block code generated by H_0; however, if $e_0 = 0$ and a single block $e_d \neq 0$ for some d, $1 \leq d \leq m$, condition (13) ensures that $[S]_m$ is not a codeword in the block code generated by H_0. Therefore, e_0 contains a correctable error pattern if and only if $[S]_m$ is a codeword in the block code generated by H_0. This requires determining if $[S]_{\mathrm{m}}$. $H_0^T = 0$ is the $n \times 2n$ block code parity check matrix corresponding to H_0. If $[S]_m$ $H_0^T = 0$, the decoder must then find the correctable error pattern that produced the syndrome $[S]_m$. Because in this case $[S]_m = e_0H_0$, we obtain the estimate of simply by reversing the first n bits in $[S]_m$. For a feedback decoder, the syndrome must then be modified to remove the effect of e_0. But, for a correctable error pattern, $[S]_m = e_0H_0$ depends only on e_0, and hence when the effect of e_0 is removed the syndrome will be reset to all zeros. Error correction system provides a more detailed view of some subassemblies in Figure 2 (see Figure 5). The processed data $\bar{d}_i$ can include errors $\bar{e}_i$ and the error correction system will subtract their estimates $\bar{e}'_i$ as indicated in the corrected data output of the error correction system. If one of the computed parity values, $\bar{p}_{u_i}$ or $\bar{p}_{l_i}$ in Figure 5, comes from a failed subsystem, the error correction system's inputs may be incorrect. Since the data are correct under the single failed subsystem assumption, the data contain no errors and the error

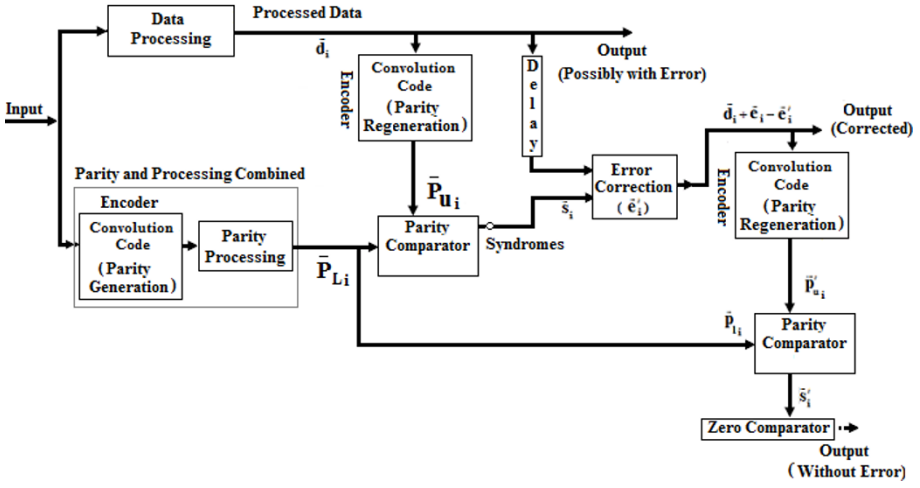

Figure 5 Block diagram of the ABFT technique along with error correction system.

correction system is operating correctly. The error correction system will observe the errors in the syndromes and properly estimate them as limited to other positions. In addition, an excessive number of error estimates $\{\bar{e}_i'\}$ could be deducted from correct data, yielding $\{\bar{d}_i - \bar{e}_i'\}$ values at the Error Correction System's output, which the regeneration of parity values produces $\{\bar{p}_{u_i}'\}$. There are several indicators that will detect errors in the error correction system's input syndromes $\{\bar{s}_i\}$.

6. Simulations and results

6.1. Design evaluation

The methods discussed in this article are programmed using the MATLAB programming tool. The MATLAB code forms the basis for a simulation program that explores the role of the threshold τ , $S = \bar{p}_{l_i} - \bar{p}_{u_i}$ if $|\bar{p}_{l_i} - \bar{p}_{u_i}| < \tau$ then there are no errors. If the threshold τ is set too low, even occasional round-off errors will exceed it, indicating failures leading to recomputation unnecessarily. It is generally permissible to accept a few small errors that are in the range of round-off levels. Nevertheless, the simulations examine how the threshold choice impacts undetected errors. Errors are detected by examining the magnitude of the respective syndromes and comparing against thresholds five times the standard deviation of syndrome values when only low levels of round-off error appear. The simulation program randomly selects the line in a magnitude error is superimposed. The magnitude of each error is chosen from a Gaussian population with zero mean and fixed variance. For small thresholds, large errors always lead to detection, whereas large thresholds increase the undetected error performance. The threshold was varied over a wide range so as to see the transition between low detected errors and high levels of missed errors. However, for a simulation, the error-detecting capabilities are interrelated with the variance of the simulated computer-induced errors. The probability of undetected errors when errors are present is evaluated as the ratio of threshold to error variance is varied over several orders of magnitude. The results are shown in Figure 6. The input data size is k = 100 samples. The error magnitude variance is taken as 10^{-3} so that, probabilistically, only small errors are superimposed. At very low thresholds, the experimental probability of undetected errors is zero. The values are not displayed on the smallest part of the abscissa. The curves shown in Figure 6 never have any undetected error until the threshold 5, when the first undetected probability is 1.1×10^{-4}. Two longer simulations using 10^6 samples are performed for two low thresholds of 2×10^{-3} and 2×10^{-5}. The undetected error rate is 4.86×10^{-7} when the threshold is 2×10^{-5}. For the slightly higher threshold of 2×10^{-3} this error rate is 4.724×10^{-5}.

By comparing the differences between the two parity values $\bar{p}_{u_i}$ and $\bar{p}_{l_i}$, we can show the checking system responding to error.

Figure 7 shows how the errors are reflected at the checker output (comparator). The top figure shows a very small difference between the two parity values $\bar{p}_{u_i}$ and $\bar{p}_{l_i}$. The reason for the non-zero differences is round off errors because of the finite answer of computing system. In the bottom figure, the values of $|\bar{p}_{l_i} - \bar{p}_{u_i}|$ reflect errors occurred. If the error threshold is setup low enough, then most of the errors can be detected by the comparator; however, if we set the

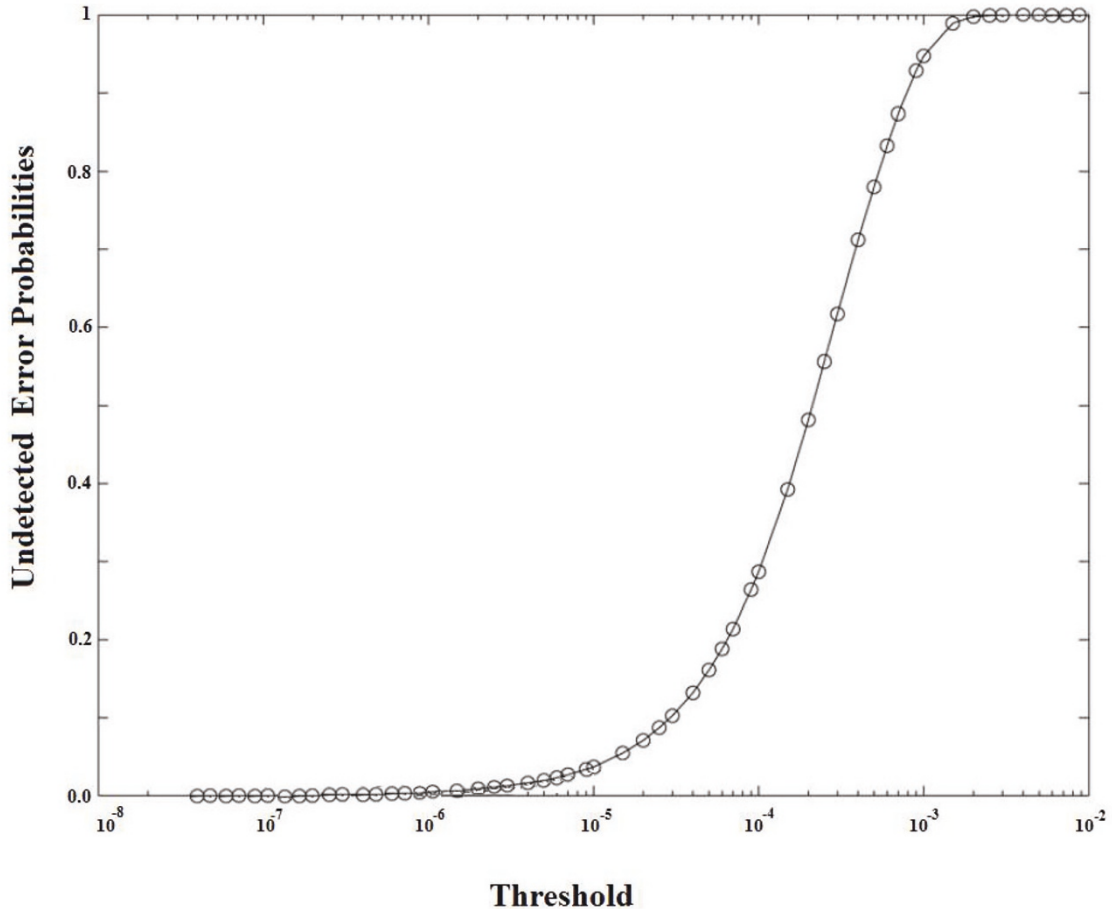

Figure 6 Undetected error probabilities versus threshold choices.

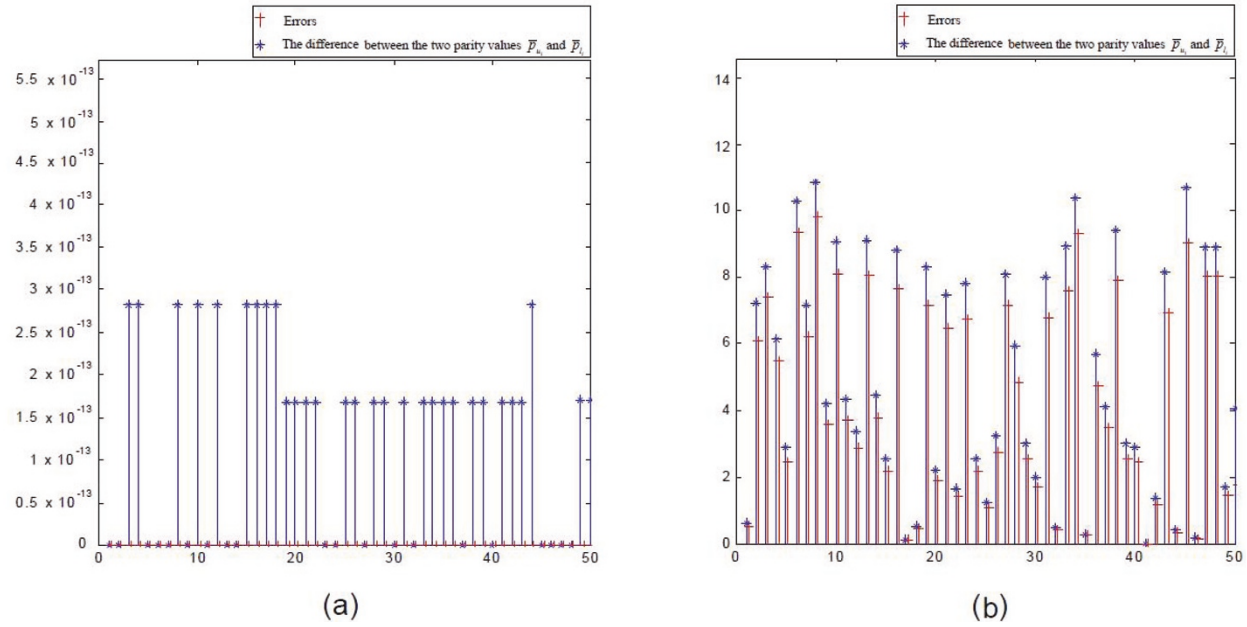

Figure 7 The responding to errors (computer-induced errors): (a) no errors, (b) errors and the difference between the two parity values $\bar{p}_{u_i}$ and $\bar{p}_{l_i}$.

threshold too low, the comparator may pick up the round-off errors and consider those to be the errors because of the computer-induced errors. Thus, we need to find a good threshold, which separates the errors because of computer analysis limited and the computer-induced errors.

Figure 8 gives the error detection performance versus the setup threshold. At the small setup threshold, the checker picks up most the errors occurred. The performance is getting worse when the threshold is getting larger.

6.2. Mean square error performance

The correction procedures are governed by a minimum mean square error (MSE) criterion. This section examines the MSE performance through MATLAB simulations. Errors are inserted additively, both in the code symbols and syndrome values to model failures. Simulation runs for the code (4, 3), rate 3/4 is performed for each standard deviation of the inserted errors, 10^{-3} to 10^{-8}. The insertion error rate is $p = 5 \times 10^{-3}$. The average MSE plots shown in Figure 9 display the values for input errors as well as those for corrected code. The input mean-squared values for input errors are very similar by statistical regularity while the corrected MSEs are much lower since large errors have been eliminated. Furthermore, the code seems quite capable of correcting all errors. The differences between input error mean-squared values and its corrected version can be evaluated by taking a ratio of their mean-squared levels.

6.3. Examples and simulations

A BP burst-correcting convolution code (6, 5, 11) is constructed [16] for use with a fault-tolerant processing situation. A rate 1/3 (3, 1, 10) code is chosen from a standard text [16] which have a constraint parameter m = 10. Long simulations involving 250, 000 blocks of data over a wide range of variances are performed. For the

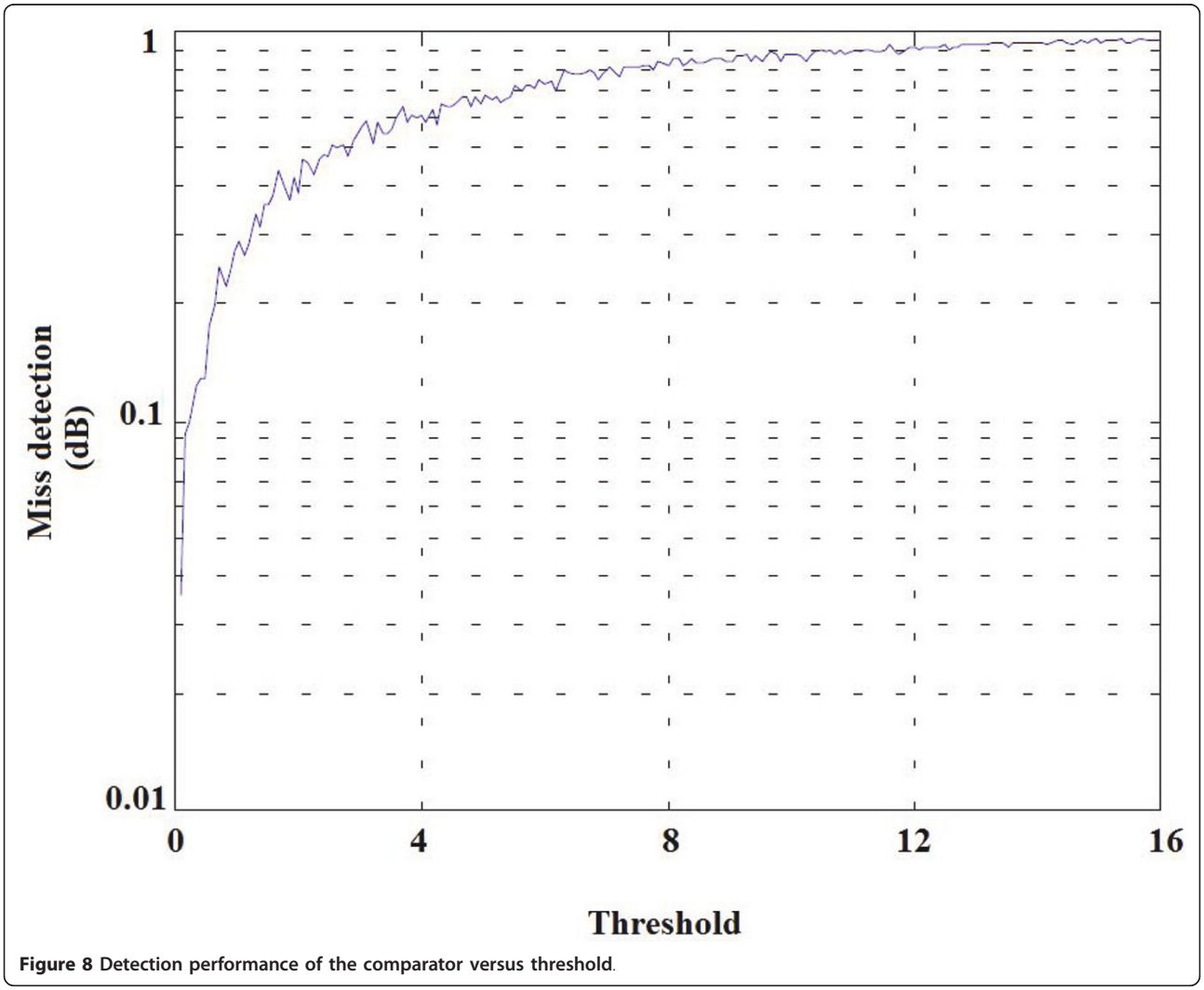

Figure 8 Detection performance of the comparator versus threshold.

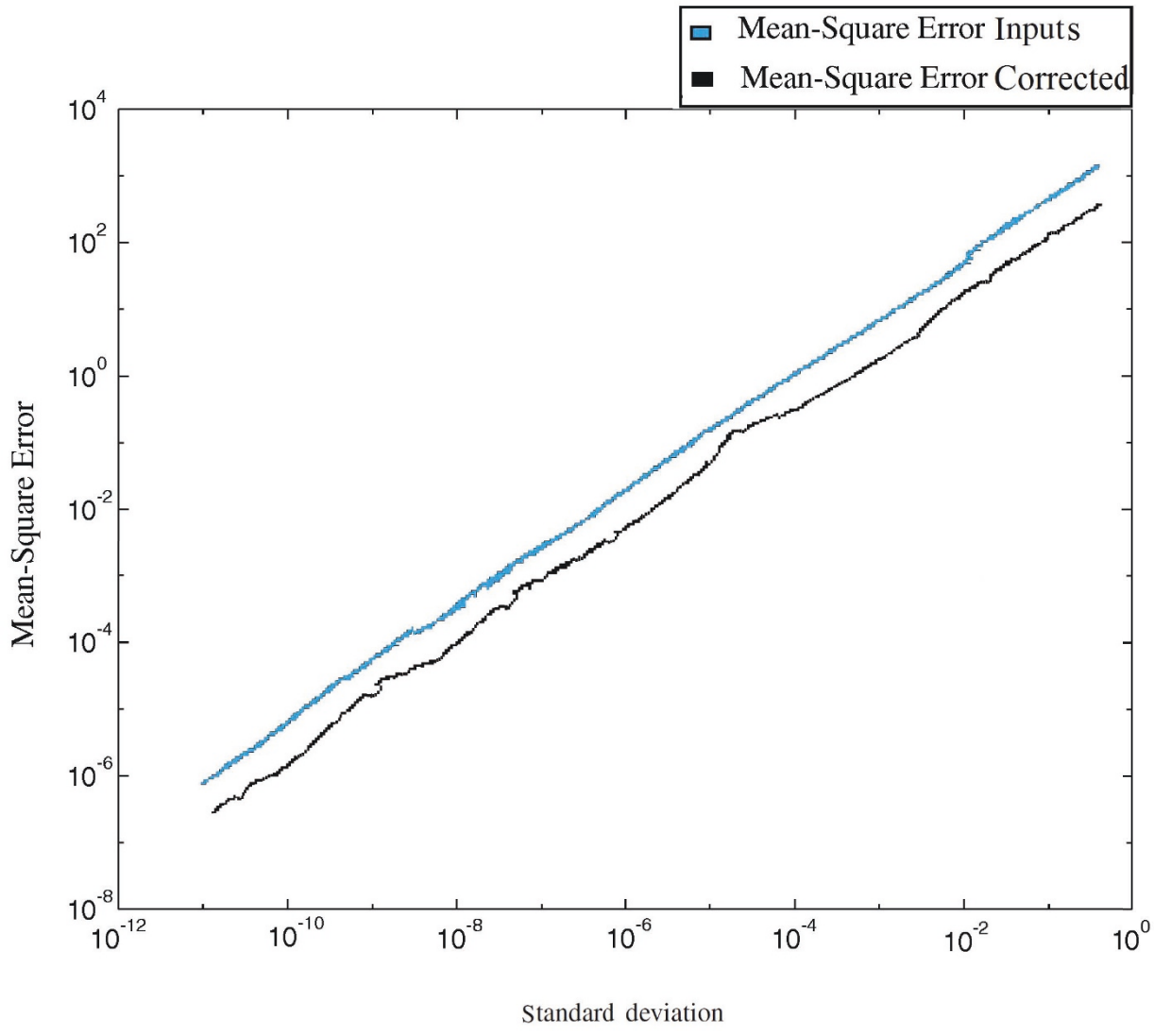

Figure 9 Average MSE values versus standard deviation.

rate 1/3 code, this represented 750, 000 samples, while for the rate 5/6 code case it implied 1.5 million samples. Burst and errors within each block are permitted. A burst in this context means that the standard deviations of all components in a block are raised to 10% of the maximum standard deviation. On the other hand, when a burst is not active, errors are allowed with positions within a block chosen independently at random, and those selected had their standard deviations raised to 10% of their maxima. The probability of a burst is 5×10^{-3}, while intra block errors have probability 10^{-3}. For long simulations, the basic parameter σ^2 (variance of error) is changed from 10^{-9} up to 3.2.

The mean-square error performance for the rate 1/3 example is shown in Figure 10a, while that for the processing system protected by the rate 5/6 BP code is displayed in Figure 10b. These plots show consistent improvement for the coded situations over the wide range of modeling error variances. The corrective actions for both cases are displayed in Figure 11. The input errors and correction values are displayed as labeled, but the important plots represent the absolute value of correction differences.

7. Conclusions

This article addresses new methods for performing error correction when real convolution codes are involved. Real convolution codes can provide effective protection for data processing operations at the data-parity level. Data processing implementations are protected against

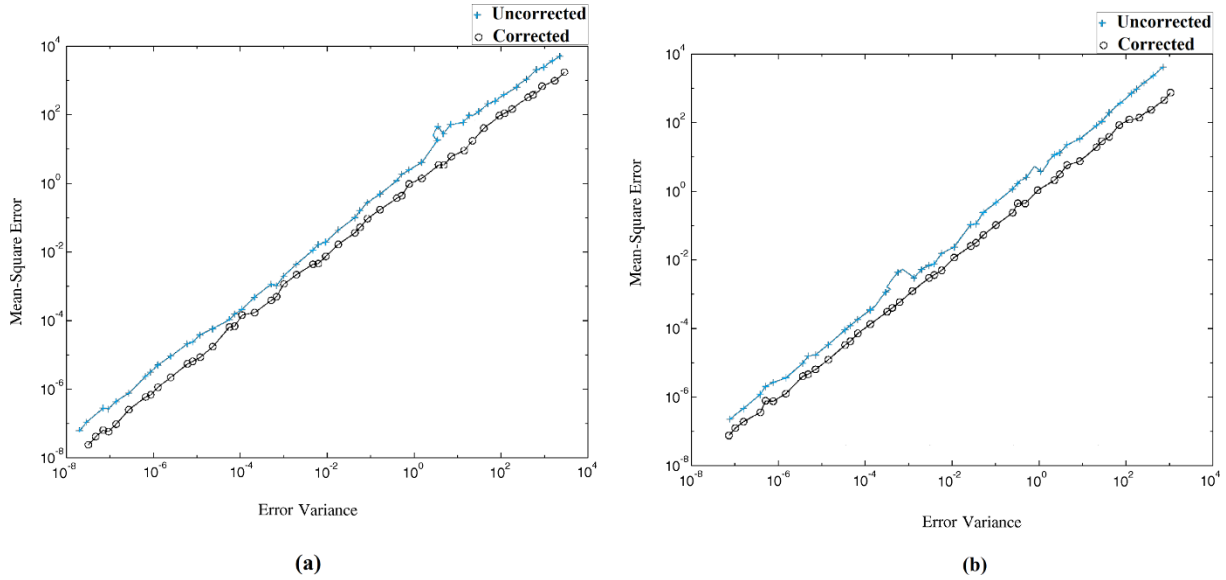

Figure 10 MSE versus error variance: (a) rate 1/3, (b) rate 5/6 BP code.

both hard and soft errors. The data processing system is protected through parity sequences specified by a high rate real convolution code. Parity comparisons provide error detection, while output data correction is affected by a decoding method that includes both round-off error and computer-induced errors. The error detection structures are developed and they not only detected subsystem errors, but also corrected errors introduced in the data processing system. Concurrent parity values techniques are very useful in detecting numerical error in the data processing operations, where a single error can propagate to many output errors. Parity values are the most effective tools used to detect burst errors occurring in the code stream. The detection performance in the data processing system depends on the detection threshold, which is determined by round-off tolerances. The structures have been tested using MATLAB programs and compute error detecting performance of the concurrent parity values method and simulation results are presented.

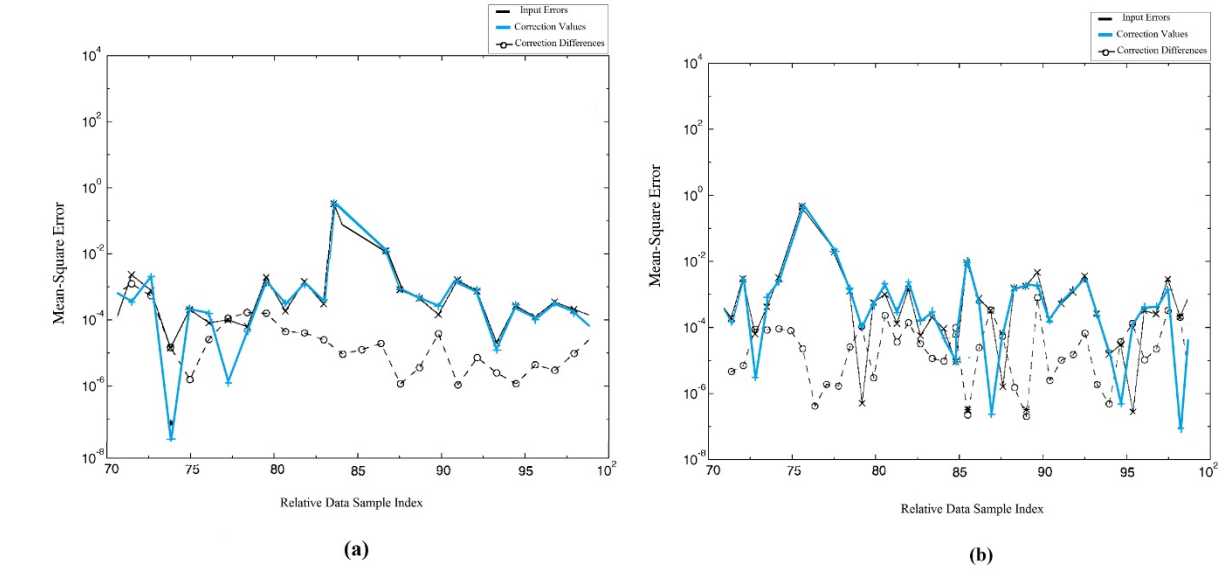

Figure 11 Correction values and differences: (a) rate 1/3, (b) rate 5/6 BP code.

Abbreviations
ABFT: algorithm-based fault tolerance; BP: Berlekamp-Preparata; LDPC: low density parity check; MSE: mean square error; RS: Reed-Solomon.

Acknowledgements
The authors are grateful to the comments from Mrs. Mahbobeh Meshkinfam that significantly improved the quality of this article.

Competing interests
The authors declare that they have no competing interests.

Received: 9 May 2011 Accepted: 22 October 2011
Published: 22 October 2011

References

1. KH Huang, JA Abraham, Algorithm-based fault tolerance for matrix operations. IEEE Trans Comput. **C-33**, 518–528 (1984)
2. JY Jou, JA Abraham, Fault tolerant matrix arithmetic and signal processing on highly concurrent computing structures. Proc IEEE. **74**(5), 732–741 (1986)
3. JY Jou, JA Abraham, Fault-tolerant FFT networks. IEEE Trans Comput. **37**, 548–561 (1988). doi:10.1109/12.4606
4. P Banerjee, JT Rahmeh, CB Stunkel, VSS Nair, K Roy, JA Abraham, Algorithm-based fault tolerance on a hypercube multiprocessor. IEEE Trans Comput. **39**, 1132–1145 (1990). doi:10.1109/12.57055
5. J Rexford, NK Jha, Algorithm-based fault tolerance for floating-point operations in massively parallel systems. Proc Int Symp on Circuits & Systems 649–652 (1992)
6. VSS Nair, JA Abraham, Real number codes for fault-tolerant matrix operations on processor arrays. IEEE Trans Comput. **39**, 426–435 (1990). doi:10.1109/12.54836
7. G Bosilca, R Delmas, J Dongarra, J Langou, Algorithm-based fault tolerance applied to high performance computing. J Parallel Distrib Comput. **69**(4), 410–416 (2009). doi:10.1016/j.jpdc.2008.12.002
8. T Roche, M Cunche, JL Roch, Algorithm-based fault tolerance applied to P2P computing networks. ap2ps, 2009 First International Conference on Advances in P2P Systems 144–149 (2009)
9. GR Redinbo, Generalized algorithm-based fault tolerance: error correction via Kalman estimation. IEEE Trans Comput. **47**(6), 1864–1876 (1998)
10. GR Redinbo, Failure-detecting arithmetic convolution codes and an iterative correcting strategy. IEEE Trans Comput. **52**(11), 1434–1442 (2003). doi:10.1109/TC.2003.1244941
11. GR Redinbo, Wavelet codes for algorithm-based fault tolerance applications. IEEE Trans Depend Secure Comput. **7**(3), 315–328 (2010)
12. GR Redinbo, Systematic wavelet Sub codes for data protection. IEEE Trans Comput. **60**(6), 904–909 (2011)
13. A Moosavie Nia, K Mohammadi, A generalized ABFT technique using a fault tolerant neural network. J Circ Syst Comput. **16**(3), 337–356 (2007). doi:10.1142/S0218126607003708
14. J Baylis, Error-Correcting Codes: A Mathematical Introduction, (Chapman and Hall Ltd, 1998)
15. VS Veeravalli, Fault tolerance for arithmetic and logic unit. IEEE Southeastcon 09 329–334 (2009)
16. D Costello, S Lin, Error Control Coding Fundamentals and Applications, (Pearson Education Inc., NJ, 2004), 2
17. RH Morelos-Zaragoza, The Art of Error Correcting Coding, (Wiley, 2006), 2 ISBN: 0470015586
18. AJ Viterbi, JK Omura, Principles of Digital Communication and Coding, (McGrawhill, 1985), 2
19. ER Berlekamp, A class of convolution codes. Inf Control. **6**, 1–13 (1962)
20. JL Massey, Implementation of burst-correcting convolution codes. IEEE Trans Inf Theory. **11**, 416–422 (1965). doi:10.1109/TIT.1965.1053798
21. LHC Lee, Convolutional Coding: Fundamentals and Applications, (Artech House, 1997)

doi:10.1186/1687-6180-2011-90
Cite this article as: Hamidi *et al.*: A framework for ABFT techniques in the design of fault-tolerant computing systems. *EURASIP Journal on Advances in Signal Processing* 2011 **2011**:90.