Available online at http://ijim.srbiau.ac.ir/
Int. J. Industrial Mathematics (ISSN 2008-5621)
Vol. 8, No. 1, 2016 Article ID IJIM-00642, 15 pages
Research Article

An approach to fault detection and correction in design of systems using of Turbo codes

H. Hamidi [*][†]

Received Date: 2014-01-06 Revised Date: 2015-09-01 Accepted Date: 2015-09-30

Abstract

We present an approach to design of fault tolerant computing systems. In this paper, a technique is employed that enable the combination of several codes, in order to obtain flexibility in the design of error correcting codes. Code combining techniques are very effective, which one of these codes are turbo codes. The Algorithm-based fault tolerance techniques that to detect errors rely 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, can apply turbo codes for the redundancy. The goal is to describe new protection techniques that are easily combined with normal data processing methods, leading to more effective fault tolerance. 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. This method is a new approach to concurrent error correction in fault-tolerant computing systems. In this paper we present methods for employ turbo codes into systematic forms and evaluation them with class of Convolutional codes, which is based on burst-correcting codes, and bounds on the fault tolerance redundant computations are given. The methods and analysis of the fault tolerance for the data processing systems are presented. A new technique is presented for protecting against both hard and soft errors at the data sample level using the error-detecting properties of turbo codes. The data processing system is surrounded with parallel parity defined by a turbo code. Erroneous behavior is detected by comparing externally the calculated and regenerated parity values.

Keywords : Turbo codes; Fault Detection; Error correction; Redundancy; Computing Systems.

1 Introduction

In the case of fault tolerance, turbo codes are primarily used for error detection, providing the vector space separations, and detected abnormal behavior leads to re-computation of the corrupted results. While the theory of real number coding is similar to codes over finite fields, the decoding for error-correcting purposes is more complicated. Algorithm based fault tolerance, proposed by Huang and Abraham [1], is a fault tolerance scheme that uses Concurrent Error Detection (techniques at a functional level). ABFT techniques are most effective when applied in a systematic form. The redundancy necessary for the ABFT method is commonly defined by real number codes, generally of the block type [2]-[8].

[*]Corresponding author. *h_hamidi@kntu.ac.ir*
[†]Information Technology Engineering Group, Department of Industrial Engineering, K. N. Toosi University of Technology, Tehran, Iran.

It has been used to reduce redundant hardware. ABFT methodologies used in [9], [10] present parity values dictated by a turbo code for protecting linear processing systems. These codes provide error detection in a continuous mode using the same computational resources as the algorithm progresses.
For the first time, in this paper we present methods for employ turbo codes into systematic forms. ABFT techniques are most effective when applying a systematic form. The redundancy necessary for the ABFT method is commonly defined by real number codes, generally of the block type [2]-[6].
ABFT technique is distinctive by three characteristics:

(a) encoding the input sequence,

(b) Plan again of the algorithm to act on the encoded input sequence,

(c) Distribution of the redundant computational steps among the individual computational units in order to adventure maximum parallelism.

The input sequences are encoded in the form of error detecting or correcting codes. The modified algorithm operates on the encoded data and produces encoded data output, from which useful information can be recovered very easily. Obviously, the modified algorithm will take more time to operate on the encoded data when compared to the original algorithm; this time redundant must not be excessive.
In order to use ABFT techniques efficiently, a systematic form is desirable [11]-[13]. Performing error correction when infrequent intermittent errors appear in the protected output values is appealing in several settings. If corrupted output values are recomputed after error detection, the necessary control structure becomes very complicated and the overall processing speed throughput is degraded accordingly. In another situation, data and the related real-number parity values are located in storage and when they are required again, the occurrence of data errors is detected. Then correcting a few errors may be much simpler and faster than re-computing the original data, even if the same processes are still active. However, error correction would probably be employed if a viable error-correcting procedure were available.
The ABFT error detection technique relies on the comparison of parity values computed in two ways. Number data processing errors are detected by comparing parity values associated with a convolution code. This article proposes a new computing paradigm in order 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.
We make the following contributions in this paper: In section 2, we discuss the related work, In Section 3, we propose the usage of codes, turbo codes and burst-correcting Convolutional codes, for ABFT technique, In Section 4, we discuss The error performance along with evaluation and simulation of turbo codes, In Section 5, simulations and results are presented, In Section 6, discussion of the conclusions.

2 RELATED WORK

High performance computers are in great demand in modern data processing systems which involve processing of a large amount of data. The application areas of these high-speed computers demand a large degree of reliability of the computed results. However, the probability of errors in the result increases with the amount of computation. In order to accommodate the contradictory requirements, high complexity and high reliability, the system has to be designed to be fault tolerant. Conventional fault tolerance techniques such as triple modular redundancy (TMR) and triple time redundancy (TTR) suffer from either a high hardware overhead (cost) or time overhead (degraded performance). The Algorithm Based Fault-Tolerance (ABFT) approach transforms a system that does not tolerate a specific type of faults, called the fault-intolerant system, to a system that provides a specific level of fault tolerance, namely recovery and/or safety. The advantage of Algorithm Based Fault-Tolerance is that errors which are caused by permanent or transient failures in the system can be detected and corrected by using a very low overhead and at the original throughput. Real number codes involve symbols that have real or integer values as opposed to classic binary codes. The real number turbo codes hold great promise of protecting many data processing subsystems. There are times when the error detection capabilities of ABFT methods are not enough. Concurrent error correction at the data-level for compensating the effects of intermittent failures avoids disrupting the data flow to react to detected errors. Turbo codes which employ real-number symbols are difficult to decode because of the size of the alphabet and the numerical and round-off noise inherent in arithmetic operations. Such codes find applications in both fault-tolerance support for signal processing subsystems and in channel coding for communication systems. The previous researches do not contain any realistic decoding algorithms for real-number turbo codes particularly when there is inherent numerical and round-off error in processing operations. Any decoding algorithm must function properly within these levels of error. Moreover, the presence of numerical round-off error is a second form of disruptive influence not present in decoding finite-field-based turbo codes. An important challenge is separating numerical round-off effects from internal hardware failure effects.

In this paper, a framework for ABFT methods is presented in the design of fault tolerant computing systems. The ABFT error detection technique relies on the comparison of parity values computed in two ways. Parity comparisons, syndromes, provide error detection, while output data correction is affected by a threshold decoding that includes both round-off error and computer-induced errors. The detection performance in the data processing system depends on the detection threshold, which is determined by round-off tolerances. A majority logic decoder can easily detect and correct single errors by observing the syndrome sequence. The simulations show that the great difference between the round-off error and the computer-induced error is large enough to be distinguished. This allows the range of error detection thresholds to be chosen. Also, the detection and miss probabilities are demonstrated for some high-rate turbo codes. Examples showing the correction behavior and mean-square error performance is presented.

ABFT for arithmetic and numerical processing operations is based on linear codes. G. Bosilca et al. [7] for high-performance computing (HPC), propose a new ABFT method based on a parity check coding. In [8] is the application of Low Density Parity Check (LDPC) based ABFT, it compare and analyses the use of LDPC to classical Reed-Solomon (RS) codes with regards to different fault models. But, [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 due to the miss of appropriate structure. ABFT methodologies used in [9] present parity values dictated by a real convolutional code for protecting linear processing systems. Paper [10] introduces a class of Convolutional codes which is called burst-correcting Convolutional codes; these codes provide error detection in a continuous mode using the same computational resources as the algorithm progresses. Redinbo [11] presented a method to Wavelet Codes into systematic forms for Algorithm-Based Fault Tolerance applications. This method employ high-rate wavelet codes along with low-redundancy which use continuous checking attributes to detect the errors, in this paper since their descriptions are at the algorithm level can be applied in hardware or software. But, this technique is suited to image processing and data compression applications and is not a general method. Also, other constraint is on burst-error due to computational load high relatively. Moreover, there is onerous analytical approach to exact measures of the detection performances of the ABFT technique applying wavelet codes.

For error correction purposes, redundancy must be inserted in some form and, using the ABFT, turbo parity codes will be employed. A systematic form of turbo codes is especially profitable in the ABFT detection plan because no redundant transformations are needed to achieve the processed data after the detection operations. To achieve fault detection and correction properties of turbo code in data processing with the minimum additional computations, we propose the block diagram in Fig. 1. This figure summarizes an ABFT technique employing a systematic turbo code to define the parity values. The k is the basic block size of the input data, and n is block size of the output data, new data samples are accepted and $(n - k)$ new parity values produced.

The upper way, Fig. 1, is the Process data flow

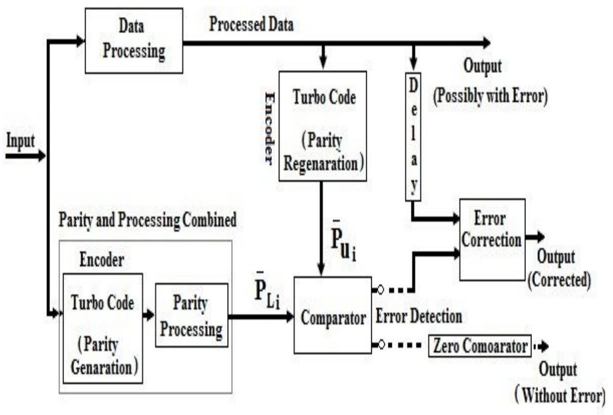

Figure 1: Block diagram of the Algorithm-based fault tolerance technique.

which passes through the process block (data processing block) and then fed to the turbo encoder (parity regeneration) to make parity values. On the other hand, the comparable parity values are generated efficiently and directly from the inputs, parity and processing combined, without producing the original outputs. The ABFT method detects errors whenever these two parity values do not compare within a tolerance threshold. 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 turbo codes structure is designed to produce distinct syndromes for a large class of errors appearing in the processing outputs. Fig. 1 employs turbo code parity in detecting and correcting processing errors.

3 Using of codes for fault tolerance technique

3.1 Turbo Encoder

Early Most of the researchers tend to focus on methods are structured, such as RM and BCH codes with very strong algebraic structures, or topological, such as Convolutional codes [12] Anyway, structures does not always result in the best distance properties for code, and can be produced very complex operations. Special types of Convolutional codes, called recursive systematic Convolutional codes (RSC), are used as the building blocks of a turbo code encoder, Fig. 2 (a). The basic turbo code encoder is built using two identical recursive systematic Convolutional (RSC) codes with parallel concatenation [1]. The two component encoders are separated by an interleaver (p), only one of the systematic outputs from the two component encoders is used, because the systematic output from the other component encoder is just a permuted version of the chosen systematic output, Fig. 2 (b). A turbo code encoder with two component codes is shown in the Fig. 2 (b).

In the general case, the code consists of two

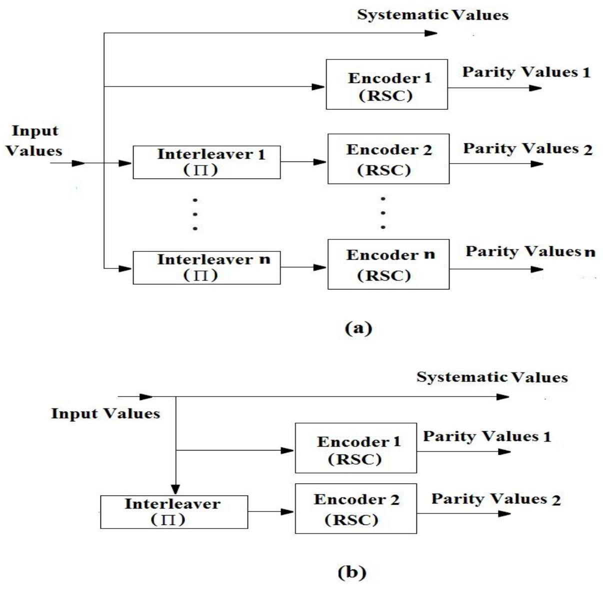

Figure 2: (a) Block diagram of the generalized turbo encoder, (b) Block diagram of the two-component turbo encoder.

parts: the un-coded input values and a set of parity sequences generated by passing interleaved versions of the information bits through Convolutional encoders. Typically, the encoders used are Recursive Systematic encoders; also, in most turbo codes the encoders used are the same (making the Turbo code symmetric), and two sets of parity values are used, one which is generated from the non-interleaved data sequence, and one which is generated from an interleaved sequence. This structure is shown schematically in Fig. 2(b). The parity values are usually punctured in order to raise the code rate to, $R = k/n, 1/2$. The data sequence may or may not be terminated, usually depending on the kind of interleaver used. We will assume that the input sequence contains k input values and is represented by $X^{(0)}$.

$$X^{(0)} = (x_0^{(0)}, x_1^{(0)}, \cdots, x_{k-1}^{(0)}) \qquad (3.1)$$

The input values, $X^{(0)}$, are the first transmitted values, which is $Y^{(0)}$.

$$Y^{(0)} = X^{(0)} = (y_0^{(0)}, y_1^{(0)}, \cdots, y_{k-1}^{(0)}) \quad (3.2)$$

The first encoder generates the parity values, $Y_p^{(1)}$.

$$Y_p^{(1)} = (y_0^{(1)}, y_1^{(1)}, \cdots, y_{k-1}^{(1)}) \quad (3.3)$$

The interleaver reorders or permutes the k bits in the input block so that the second encoder receives $X'^{(0)}$ different from the first. The parity values generated by the second encoder are represented by $Y_p^{(2)}$.

$$Y_p^{(2)} = (y_0^{(2)}, y_1^{(2)}, \cdots, y_{k-1}^{(2)}) \quad (3.4)$$

The output, final transmitted sequence, corresponding to an input symbol input is $(Y^{(0)} = X^{(0)}, Y_p^{(1)}, Y_p^{(2)})$, which $Y_p^{(1)}$ and $Y_p^{(2)}$ are represented the redundant symbols. We employ the systematic form of turbo codes to define parity values associated with groups of data samples. These parity values are computed by a finite impulse response of several groups of previous data samples, [9], [12].
Assume that the component codes are two binary systematic linear block $C_1(n_1, k_1)$ and $C_2(n_2, k_2)$. Let $G_i = [I_i|H_i]$ denote the generator of codes C_1 and C_2.Then the generator matrix of parallel concatenated code $G_p = [I_{k_1k_2}|H_1|H_2]$ can be put in the following form, which π is the permutation matrix associated with the interleaver, and H_i is the parity matrix of code $C_i, i = 1, 2$, [13], [14]:

$$G_p = \left[[I_{k_1k_2}] . \begin{bmatrix} h_1 & & \\ & \ddots & \\ & & h_1 \end{bmatrix} .\pi \begin{bmatrix} h_2 & & \\ & \ddots & \\ & & h_2 \end{bmatrix} \right] \quad (3.5)$$

The number of times that h_1 appears in the middle part h_1' of G_p is k_2; while the number of times that h_2 appears in the leftmost portion h_2' of G_p is k_1, all other entries in h_1' and h_2' are zero. It follows that code word of G_p are of the form $G = [X^{(0)}|X^{(0)}h_1'|X^{(0)}h_2']$, and the final transmitted code word is given by Y.

$$Y = (y_0^{(0)} y_0^{(1)} y_0^{(2)}, y_1^{(0)} y_1^{(1)} y_1^{(2)}, \cdots, y_1^{(0)} y_2^{(1)} y_3^{(2)}) \quad (3.6)$$

The rate of the overall code has length $N = 3K$ and rate $R = K/N \approx 1/3$ for large K. One method to explain a parallel turbo code is as a punctured product code in which the redundant values matches to the checks on checks are deleted, Fig.3 . The redundancy of the code is $r_1 = n_1 - k_1$. After encoding the rows, the columns are encoded using another block code (n_2, k_2), where the check bits of the first code are also encoded. The overall block size of such a product code is $n = n_1n_2$, the total number of information bits $k_1 \times k_2$.
This explanation is shown in Fig. 3. The only

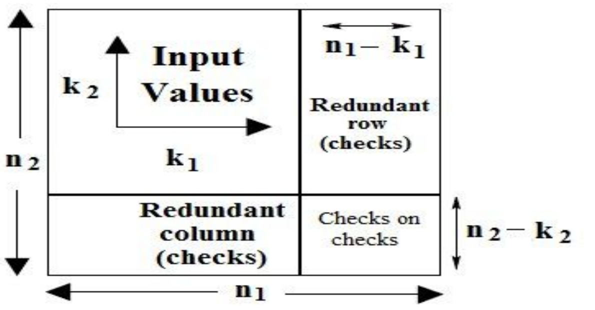

Figure 3: A punctured product code to explain a parallel turbo code.

essential difference between a turbo code and a block product code is that the inter-leaver is not clearly a row by row, column by column interleaver, but one that introduces enough disorder in the input values so that iterative decoding works [14].

3.2 Class of Convolutional Code

Considering that turbo encoder is combined of the two Convolutional encoders, Error probability of turbo code is related to Convolutional code. We consider only systematic forms of Convolutional codes because the normal operation of Process block is not change and there is no need to decoding for obtaining true outputs. In addition Convolutional codes have good correcting characteristics because of memory in their encoding structure [13]. A burst of length d is explained as a vector whose nonzero components are confined to d consecutive numeral situation, the first and last are nonzero [14], A burst apply to a group of errors which is characteristic of unforeseeable effects of errors in data computation. Costello [13] has shown that a sequence of error bits $e_{d+1}, e_{d+2}, \cdots, e_{d+a}$ is called a burst of length a concerning a guard space of length b if $e_{d+1} = e_{d+a} = 1$,and the b bits preceding e_{d+1} and the b bits following e_{d+a} are zero; and the

a bits from e_{d+1} through e_{d+a} include no subsequence of b zero. Also, for any Convolutional code of rate R that corrects all bursts of length a or less relative to a guard space of length b, $\frac{b}{a} \geq \frac{1+R}{1-R}$. Binary burst-correcting Convolutional codes at structure of the turbo codes are appropriate and efficient in detecting and correcting errors from internal computing failures. Binary burst-correcting Convolutional codes need guard bands (error-free regions) before and after bursts of errors, particularly if error correction is needed [[13], Chapter 20]. One class of burst-correcting codes is the Berlekamp-Preparata (BP) codes [13], [14] that have many appropriate characteristic with regard to failure error-detecting. Their design properties vouch for detecting the onset of errors due to failures, regardless of any error-free region following the beginning of a burst of errors. Consider designing an $(n, k = n-1, m)$ systematic Convolutional 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. One parity value is assigned for each input of $(n-1)$ values. Their constraint length is $l = 2n-1$. To design such a code assure that each correctable error value $[E]_m = [e_0, e_1, \cdots, e_m]$ results in a distinct syndrome $[S]_m = [s_0, s_1, \cdots, s_m]$ .This infer that each error values whit $e \neq 0$ and $e_d = 0$, $d = 1, 2, \cdots, m$ must produce a separate syndrome and that each of these syndromes must be separate from the syndrome caused by any error value with $e_0 = 0$ and a single block $e_d \neq 0, d = 1, 2, \cdots, m$. Therefore, the first error block e_0 can be correctly decoded if first $(m+1)$ blocks of e contain at most one nonzero block. An $(n, k = n-1, m)$ systematic code is depicted by the set of generator polynomials $g_1^{(n-1)}(D), g_2^{(n-1)}(D), \cdots, g_{n-1}^{(n-1)}(D)$. The generator matrix of a systematic Convolutional code, G, is a semi finite matrix evolving m finite sub matrixes as:

$$G = \begin{bmatrix} 1P_0 & 0P_1 & 0P_2 & \cdots & 0P_0 & & \\ & 1P_0 & 0P_1 & \cdots & 0P_{m-1} & 0P_m & \\ & & 1P_0 & \cdots & 0P_{m-2} & 0P_{m-1} & 0P_m \\ & & & \ddots & & \ddots & \ddots \end{bmatrix} \tag{3.7}$$

Where I and 0 are identity and all zero $k \times k$ matrixes respectively [15] and Pi with $i = 0$ to m is a $k \times (n-k)$ matrix . The parity-check matrix, 3.8, 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, 3.8.

$$H_m = [H_0, H_1, \cdots, H_m] \tag{3.8}$$

Where H_0 is an $n \times (m+1)$ matrix, 3.9.

$$H_0 = \begin{bmatrix} g_{1,0}^{(n-1)} & g_{1,1}^{(n-1)} & \cdots & g_{1,m}^{(n-1)} \\ \vdots & \vdots & & \vdots \\ g_{n-1,0}^{(n-1)} & g_{n-1,1}^{(n-1)} & \cdots & g_{n-1,m}^{(n-1)} \\ 1 & 0 & \cdots & 0 \end{bmatrix} \tag{3.9}$$

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 column. In a mathematical form, this operation can be expressed as:

$$H_0 = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} = H_{d-1} \cdot T \tag{3.10}$$

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 [16], [17]. 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[16]. For systematic codes, the parity check matrix sub-matrices H_m in 3.8 have special forms that control how these equations are formed.

$$\begin{aligned} H_0^T &= [P_0 | I_{n-k}], \\ H_i^T &= [P_i | 0_{n-k}] \quad i = 1, 2, \cdots, L. \end{aligned} \tag{3.11}$$

Where I_{n-k} and 0_{n-k} are identity and all zero $k \times k$ matrixes respectively and P_i is a $(n-k) \times k$ matrix . However, in an alternate view, the respective rows of H_0 contain the parity sub matrices P_i needed in H^T, 3.8 and 3.11.

$$H_0 = \begin{bmatrix} P_0 & | & I_1 \\ P_1 & | & 0 \\ P_2 & | & 0 \\ \vdots & \vdots & \vdots \\ P_{L-1} & | & 0 \\ P_L & | & 0 \end{bmatrix} \tag{3.12}$$

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

$$\begin{aligned}[S]_m &= [E]_m \cdot [H^T]_m \\ &= e_0H_0 + e_1H_1 + \cdots + e_mH_m \\ &= e_0H_0 + e_1H_0T + \cdots + e_mH_0T^m \\ &= \begin{bmatrix} s_i \\ s_{i+1} \\ \vdots \\ s_{i+n} \end{bmatrix}\end{aligned} \tag{3.13}$$

$[S]_m$ 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 $[E]_{i+1}, [E]_{i+2}, \cdots, [E]_{i+l}$ 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 $[S]_m$.

$$[E_m] = \begin{bmatrix} e_{i,1} \\ e_{i,2} \\ \vdots \\ e_{i,n} \end{bmatrix} \tag{3.14}$$

Then error values recognition

$$e_{i,n} = S_i, \quad e_{i,n-1} = S_{i+1} \tag{3.15}$$

If there are nonzero error bursts in $[E]_{i+1}, [E]_{i+2}, \cdots, [E]_{i+l}$, 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 using 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, $F_0^T \cdot [S]_m = 0$, then $[S]_m$ contain correctable model. For optimum burst error correcting code, $b/a = (1 + R)/(1 - R)$. For the preceding case with $R = (n - 1)/n$ and $b = m \cdot n = m \cdot a$, this implies that $\frac{b}{a} = m = 2n - 1$, and H_0 is an $n \times 2n$ matrix.

3.3 Iterative Turbo Code Decoder

In this paper, the turbo code decoder is based on a modified Viterbi algorithm that includes reliability values to improve decoding performance. The Viterbi algorithm produces the majority logic (ML) output value for Convolutional codes. This algorithm provides optimal sequence estimation for one stage Convolutional codes. For concatenated Convolutional codes, there are two main disadvantages to conventional Viterbi decoders. First, the inner Viterbi decoder produces bursts of bit errors which reduce the performance of the outer Viterbi decoders [18], [19]. Second, the inner Viterbi decoder produces hard decision outputs which prevent the outer Viterbi decoders from deriving the advantage of soft decisions [1], [20], [21], [22]. Both of these drawbacks can be reduced and the performance of the overall concatenated decoder can be improved if the Viterbi decoders are able to produce reliability (soft-output) values [1]. The reliability values are passed on to subsequent Viterbi decoders as apriori information to improve decoding performance. The Viterbi algorithm was modified to output bit reliability information [22]. The soft-output Viterbi algorithm (SOVA) computes the reliability of the input values as a log-likelihood ratio (LLR),

$$\Lambda(X^{(0)}) = \log\left(\frac{\Pr[X^{(0)} = 1|\bar{r}]}{\Pr[X^{(0)} = 0|\bar{r}]}\right) \tag{3.16}$$

Where $\bar{r}$ denotes the received sequence. In an iterative decoding procedure, the output information provided by $\Lambda_{i,e}(X^{(0)})$ can be fed back to the decoder as a priori probability for a second round of decoding. The output LLR can be written as, Let $D_i^{(j)}$ be the set of branches connecting state $S_{i-1}^{(l')}$ to state $S_i^{(l)}$ such that the associated information bit $X^{(0)} = j$, with $j \in \{0, l\}$.

$$\Lambda_{i,e}(X^{(0)}) = \log\left(\frac{\sum\limits_{(l,l')\in D_i(0)} \zeta_{i-1}(l')\gamma_i(l,l')\eta_i(l)}{\sum\limits_{(l,l')\in D_i(1)} \zeta_{i-1}(l')\gamma_i(l,l')\eta_i(l)}\right) \tag{3.17}$$

Where $\zeta_{i-1}(l')$,$\eta_i(l)$ and $\gamma_i(l, l')$ are given by

$$\zeta_i(l) = \Pr\{S_i^{(l)}, r_i'\} \tag{3.18}$$

$$\eta_i(l) = \Pr\{r_i'|S_i^{(l)}\} \tag{3.19}$$

$$\gamma_i(l, l') = \delta_{ij}(l \cdot l') \exp\left(\frac{E}{N_0}\sum_{m=1}^{n-1} r_{i,m}x_{i,m}\right) \tag{3.20}$$

Where

$$\delta_{ij}(l \cdot l') = \begin{cases} 1 & \text{if } (l', l) \in D_i^{(j)} \\ 0 & \text{else} \end{cases} \qquad (3.21)$$

This output LLR, for an information position i, does not contain any variable directly related to $X_i(0)$, for $i = 1, 2, \cdots, N$. It should be noted that because of the assumption that encoding is systematic, and therefore the sum in the modified branch metric [20] starts at $m = 1$. In the more general case of a two-dimensional product coding scheme, the first decoder produces $\Lambda_{i,e}^{(1)}(X^{(0)})$ which is given to the second decoder as a priori probability $\Lambda_{i,e}^{(2)}(X^{(0)})$ to be used in the computation of the LLR of input values $X_i{}^{(0)}$. In other words, the output information provides a soft output that involves only reliabilities that are not directly related to the information symbol $X_i^{(0)}$. The basic structure of an iterative decoder with two component codes is shown in Figure 4; each iteration consists of two phases, one decoding phase per component decoder. First phase, In the first decoding iteration the soft-in soft-out decoder for the first component code computes the a posteriori LLR ,3.17.This decoder computes the extrinsic information for each information symbol,$\Lambda_{i,e}^{(1)}(X^{(0)})$, on the basis of the part of the received sequence that corresponds to the parity symbols,$\bar{r}_{pl}$,and sends the result to the second decoder.

In the second phase of the first decoding itera-

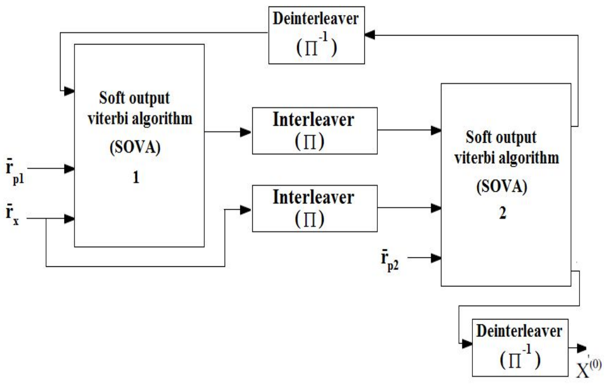

Figure 4: An iterative decoder for a parallel turbo code.

tion, the permuted (or interleaved) output information from the first decoder is used as a priori LLR, $\pi \cdot \Lambda_{i,e}^{(1)}(X^{(0)})$. Extrinsic information $\Lambda_{i,e}^{(2)}(X^{(0)})$ is computed on the basis of the part of the received sequence that corresponds to the parity values of the second component code,$\bar{r}_{p2}$, thus conclude the first decoding iteration. At this point, a decision can be made on an information symbol, on the basis of its a posteriori LLR $\Lambda_{i,e}(X^{(0)})$. In subsequent iterations, the first decoder uses the de interleaved extrinsic information from the second decoder, $\pi^{-1} \cdot \Lambda_{i,e}^{(2)}(X^{(0)})$, as a priori LLR for the computation of the soft-output (the a posteriori LLR), $\Lambda_{i,e}(X^{(0)})$. This procedure can be repeated until either a stopping criterion is met [23], [24] or a maximum number of iterations is performed. It should be noted that making decisions on the information symbols after the first decoder saves one deinterleaver.

4 Performance and Evaluattion

4.1 Error Performance

With iterative decoding, [1], the error performance improves. Typical of turbo coding schemes is the fact that increasing the number of iterations results in a monotonically decreasing improvement in coding gain. Increasing the number of iterations from 2 to 6 gives an improvement in SNR of 1.7 dB, whereas going from 6 to 18 iterations yields only a 0.3 dB improvement in coding gain. Since the appearance of turbo codes, advances have taken place in understanding the bit error rate (BER) behavior of turbo codes [25]. There appears to be a consensus among researchers on why turbo codes offer such an superior error performance, 3.1 Turbo codes have a weight distribution that approaches, for long interleavers, that of random codes, 3.2 Recursive Convolutional encoders and proper interleaving map most of the low-weight information sequences into high-weight coded sequences, and 3.3 Systematic encoders allow the effective use of iterative decoding techniques utilizing constituent SOVA decoders. Information symbol estimates are available directly from the channel.

4.2 Evaluation and Simulation of Turbo Codes

Considering that turbo encoder is combined of the two Convolutional encoders, Error probability and performance of turbo code is related to Convolutional code, the number of iterations, and interleaver. My simulations are based on use of the SOVA decoder and constraint lengths of 3-5.

We have considered that, the use of 1/2 rate codes degrades the BER performance by only 0.5 to 0.7 dB relative to 1/3 rate codes,10 decoding iterations are adequate. This computational are made using standard MATLAB, version 2010a. In our research the performance of turbo codes several simulations were run on a PC, CPU: Core 2 Dou 2.2 GHz 2MB RAM: 4 GB DDR 2 HDD: 640 GB. For the purposes of this simulation a punctured turbo code at rate $R = 1/2$ is used. The data block length is $k = 400$ bits, and a SOVA decoder is used in the simulation. The results shown at Fig. 5 are the BER versus E_b/N_o (channel signal to noise ratio) curves for different numbers of iterations from $n = 1, 2, 5$ and 10.

It can observe that BERs of the order of 10^{-5}

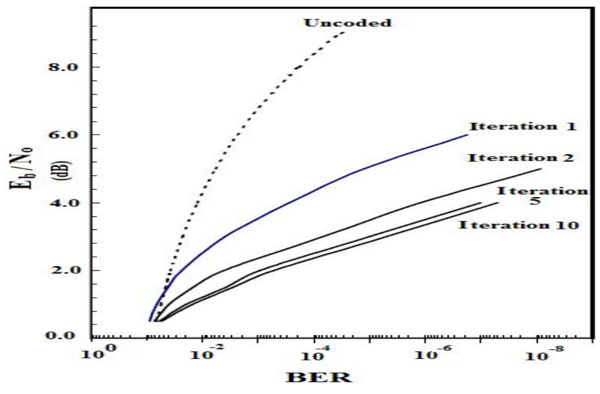

Figure 5: BER for Turbo Code.

are achievable with $E_b/N_o > 3$ dB with modest numbers of iterations. A typical coding gain of $E_b/N_o > 3$ dB, relative to an un-coded channel, is observed at a BER of 10^{-5}. Fig. 5 derive that the BER should improve with each iteration, so a series of simulations are run to evaluate the improvement. It can observe from Fig. 6 that the first few iterations yield the most significant improvements in BER for any given E_b/N_o. Thereafter the results appear to converge onto a BER for each value of E_b/N_o. It is obvious that there is a tradeoff to be made between the number of iterations, processing power, and E_b/N_o when looking for a given BER.

4.3 Turbo Code Error-Performance Example

Performance results using turbo codes have been presented in [3] for a rate 1/2, $K = 5$ encoder implemented with generators $G1 = \{11111\}$ and $G2 = \{10001\}$, using parallel concatenation and a 256 . 256 array interleaver. The modified algorithm was used with a data block length of 65,536 bits. After 18 decoder iterations, the bit-error probability PB was less than 10^{-5} at $E_b/N_o = 0.7$ dB. The error-performance improvement as a function of the number of decoder iterations is seen in Fig. 5. Note that, as the Shannon limit of -1.6 dB is approached, the required system bandwidth approaches infinity, and the capacity (code rate) approaches zero. Therefore, the Shannon limit represents an interesting theoretical bound, but it is not a practical goal. For binary modulation, several authors use $PB = 10^{-5}$ and $E_b/N_o = 0.2$ dB. Thus, with parallel concatenation of RSC convolutional codes and feedback decoding, the error performance of a turbo code at $PB = 10^{-5}$ is within 0.5 dB of the Shannon limit. A class of codes that use serial instead of parallel concatenation of the interleaved building blocks has been proposed.

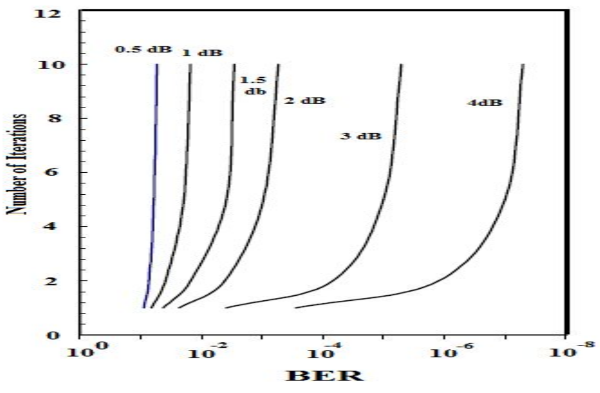

Figure 6: BER versus as number of iterations varies.

5 Simulation and Results

5.1 Error Correction System

Error correction system, Fig. 6, provides a more detailed view of some subassemblies in Fig. 1, in Fig. 7. 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 Fig. 7, comes from a failed subsystem, the error correction systems inputs may be incorrect. Since the data are correct under the single failed subsystem assumption, the data contain no errors and the error 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. Moreover, an excessive number of error estimates $\{\bar{e'}_i\}$ could be deduct from correct data, yielding $\{\bar{d}_i - \bar{e'}_i\}$ values at the Error Correction Systems output, which the regeneration of parity values produces $\{\bar{p'}_{u_i}\}$, as shown in Fig. 7 at the final output.

$$\bar{p'}_{u_i} = \sum_{j=0}^{l} p_j(\bar{d}_{i-j} - \bar{e'}_{i-j}) \tag{5.22}$$

Simultaneously, if the errors do not affect the parallel parity values $\{\bar{p'}_{l_i}\}$, its value is correct.

$$\bar{p}_{l_i} = \sum_{k=0}^{l} p_k \bar{d}_{i-k} \tag{5.23}$$

The output checking syndromes $\{\bar{s'}_i\}$ will become nonzero at the beginning of errors because of the burst-detecting nature of the code.

$$\bar{S'}_i = \bar{p'}_{u_i} - \bar{p}_{l_i} = \sum_{j=0}^{l}(-1)p_j\bar{e}_{i-j} \tag{5.24}$$

Therefore, there are several indicators that will detect errors in the error correction systems input syndromes $\{\bar{S}_i\}$. The checking syndromes $\{\bar{S'}_i\}$ must indicate the beginning of errors, so the error correction system cannot subtract incorrect, even overwhelming errors from otherwise correct data without observation. The limited checking features inserted in and around the corrector will always detect its unsuitable behavior.

It is an easy matter to construct MATLAB to implement the BP Convolutional codes and turbo codes. Thus, a series of simulations provide appraise of the probability of detection and failure. Several simulation schemes modeling the ABFT method for detecting numerical level errors were described in MATLAB, version 2010a, where the modeling errors were assumed Gaussian with zero means and statistically independent from symbol to symbol. Errors were allowed in the parity values computed by the combined data parity generator, Fig. 1, and in the processed data symbols. Very researches were performed to verify the iterative decoding technique. The error modeling provides a strong set of conditions of failure effects and is completely general. An example of the G and H^T matrices for $n = 4$ is extracted after execution of these scripts. The encoding matrix G is $4 \times (3.8)$ with its top three rows containing zeros and a 4×3 identity matrix in the rightmost three columns. The last row of G exhibit the additive identity as, $(-1,-1,-1,-1,-1,0,-1,0,-1,0,0,0,-1,$ $0,0,0,-1,0,0,0,-1,0,0,0)$. The significant parity check part H^T, which dictates the parity values, is $1 \times (4.8)$; $(l+1) = 2n$ in this case. Its single row is, $(1,0,1,0,1,1,0,0,1,0,0,0,1,0,0,0,1,0,0,0,$ $0,1,0,0,0,0,1,0,0,0,0,1)$. The simulation code randomly inserts a burst of errors in each block of input symbols, representing an encoded block. The choice of the burst is controlled by probability parameter ρ. Once an error situation is established in the simulation, n symbols representing a burst are determined using a uniform distribution and then are added to the n code symbols to model a burst. Many simulation steps were executed at various error rates ρ. For a high value of ρ, the error bursts are frequent enough that they may sometimes happen so close as to violate the protective band requirement for correction, leading to incorrect. A typical test used $n = 7$ with $\rho = 10^{-2}$ and employed 10^5 code word blocks. The experimental ratio of bursts introduced in one run of 10^5 was 0.000101, and the experimental conditional probability of correction was 0.9832561, whereas that of failure was 0.0167439. When the probability of a burst was lowered to $\rho = 10^{-7}$, there were no failures for long runs. Moreover, statistically defined numerical errors were inserted randomly in the parity and data symbols. The inherent

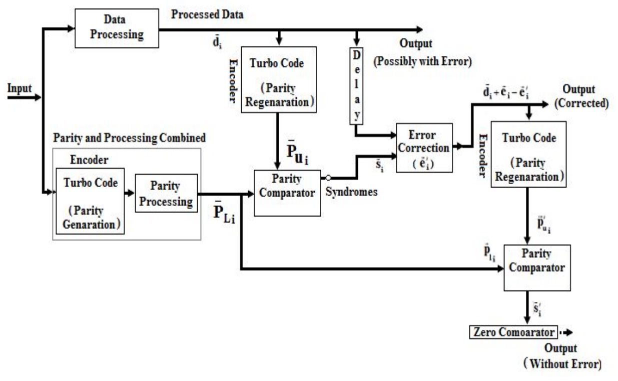

Figure 7: Block diagram of the ABFT technique along with error correction system.

memory in the parity regeneration from data allows errors to produce large syndrome values beyond the first block in which they first appear. The simulation results are shown in Fig. 8. Each estimation point employed 10^5 data blocks of k symbols, k. 10^5 symbols overall. Errors were inserted according to probability p, statistically independently for all positions. The probability of detection is the joint probability of detecting an error that is really present while the probability of miss (failure) is the joint probability of no detection when an error is there. The curves in Fig. 8 for detection and failure probabilities are as the probability of error insertions p is varied, it is shown detection and miss probabilities for two codes with $R = 3/4$ and $6/7$. The detection curves are separated because the rate at which errors occur in any blocking of data and parity symbols depends on n and the symbol error rate p. The two curves have respective block sizes of four and six. At high insertion rates p, there are on average more opportunities for misses (failures) which are masked by adjacent large errors that are detected in the same data block. Hence, the miss probabilities tend to decrease at high insertion rates.

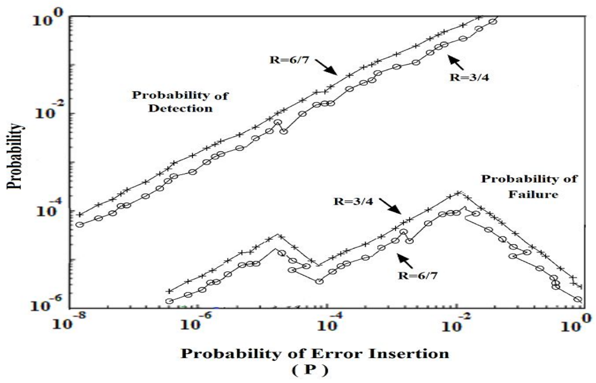

Figure 8: Detection and miss (failure) probabilities for two codes with R=3/4 and 6/7.

Figure 9 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 nonzero differences is round off errors due to 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 threshold too low, the comparator may pick up the round-off errors and consider those to be the errors due to the computer-induced errors. Thus, we need to find a good threshold, which separates the errors due to computer analysis limited and the computer-induced errors.

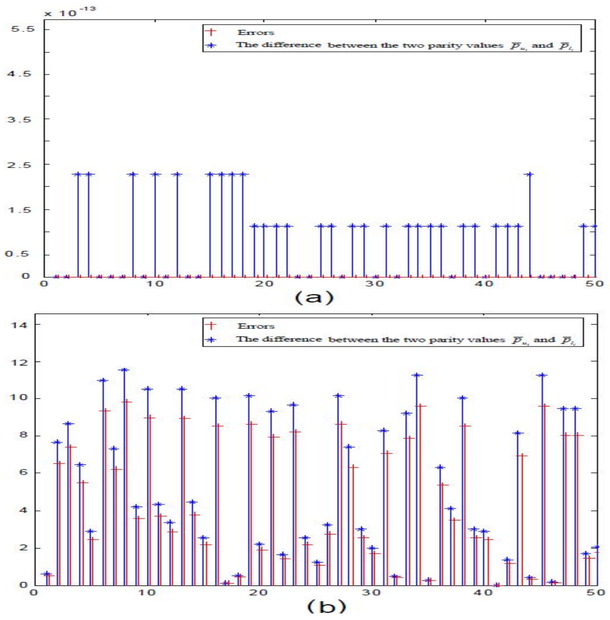

Figure 9: The responding to errors: (a) no errors (b) errors and the difference between the two parity values $\bar{P}_{u_i}$ and $\bar{P}_{l_i}$.

Figure 10 shows the quality performance curves of the turbo encoder with and without error correction. The dash line and solid line curves present the reconstruction performance of input with and without encoding request, respectively, versus error rate. As a matter of fact, the fidelity is much improved when the error correction system is implemented.

The simulation results plotted in Fig. 11, show

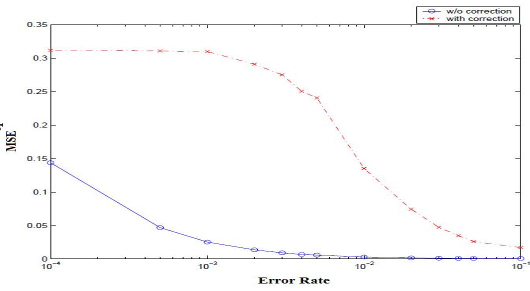

Figure 10: Quality performance curves with and without encoding request (error correction performance vs. error rate).

the performance of the iterative decoder for the Turbo Code (3750, 2550). Figure 11 show that the slope of curves and coding gain are improved by increasing the number of iterations. At 10^{-5} about 1.8 dB coding gain can be obtained after 4 iterations. After the 4th iteration, the amelioration of the coding gain becomes negligible because of the steep slope of the BER curve. The Turbo Code is well established. The curves are done with 4 iterations.

The performance of the continuous iterative de-

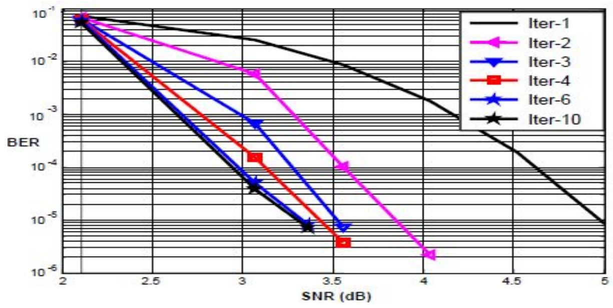

Figure 11: Effect of iterations on Iterative decoding for Turbo Code (3750, 2550) code.

coding algorithm applied to the turbo code is shown in Fig. 12, where we plot the bit-error probability as a function of the number of iterations of the decoding algorithm for various values of the bit signal-to-noise ratio. It can be seen that the decoding algorithm converges down to an error probability of 10^{-5} for signal-to-noise ratios of 0.2 dB with nine iterations. Moreover, convergence is guaranteed also at signal-to-noise ratios as low as 0.05 dB, which is 0.55 dB from the Shannon capacity limit.

The procedures performed by the iterative de-

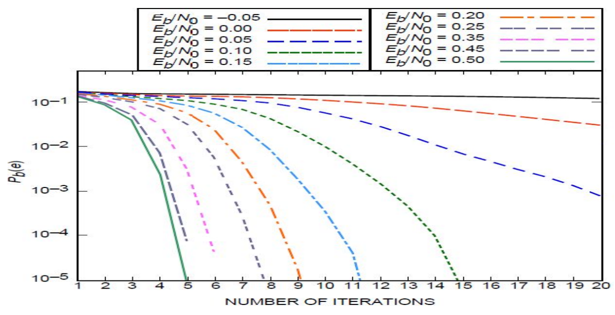

Figure 12: Turbo-decoding: bit error probability versus the number of iterations.

coder algorithm can be classified as follows [15]:

- Branch Metrics Calculation (Proc. A)
- Path Metrics Update (Proc. E)
- Hard Decision Generation (Proc. G)

Moreover, in this case procedure A does not exploit any *a priori* information. Tables 1-3 summarize the computational requirements of the various decoding algorithms as a function of the encoder memory order. Note that here we assume that the constituent RSC encoders for turbo coding, as well as the RSC encoder for convolutional coding are rate 1/2. Note that we also take into account the additional complexity associated with the branch metrics calculations due to *a priori* information exploited by the turbo decoder. Finally, Table 4 summarizes the overall complexity (in terms of the number of equivalent addition operations) of the various decoding algorithms. As an example, let us consider in detail the computational requirements of the iterative decoder for a rate 1/2 convolutional code (see Table 3).

Fig. 13 also compares the performance of various turbo-coded and convolutional-coded systems for both single and multiple antenna configurations again for the case of frames having 2048 encoded bits. However, here we set the antenna envelope correlation coefficient to be equal to zero, i.e., the ideal situation. In this case, as the number of antennas is increased, turbo codes eventually substantially outperform convolutional codes. In fact, as the number of antennas is increased the underlying fading channel will approach a non-fading AWGN channel, where turbo codes are known to substantially outperform convolutional codes.

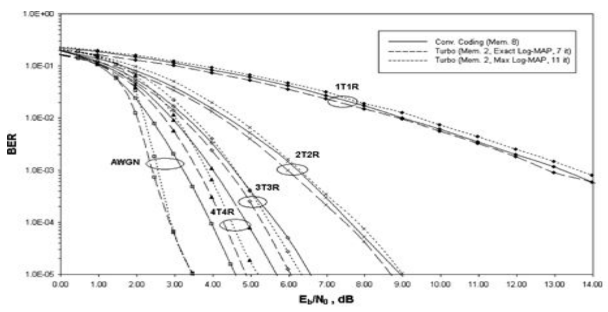

Figure 13: Compares the performance: Error rates for various turbo-coded and convolutional-coded OFDM systems for both single and multiple antenna FWA configurations for frames having 2048 code bits.

Table 1: Computational Requirments of The Log-Map Algorithm

	ADD	SUB	MUL	DIV	MAX	LKUP
Procedure A	4×2^M	-	6×2^M	2×2^M	-	-
Procedure B	3×2^M	2^M	-	-	2^M	2^M
Procedure C	3×2^M	2^M	-	-	2^M	2^M
Procedure D	$6 \times 2^M - 2$	$2 \times 2^M - 1$	-	-	$2 \times 2^M - 1$	$2 \times 2^M - 1$

Table 2: Computational Requirments of The Max-Log-Map Algorithm

	ADD	SUB	MUL	DIV	MAX	LKUP
Procedure A	4×2^M	-	6×2^M	2×2^M	-	-
Procedure B	2×2^M	-	-	-	2^M	-
Procedure C	2×2^M	-	-	-	2^M	-
Procedure D	$4 \times 2^M - 2$	1	-	-	$2 \times (2^M - 1)$	-

Table 3: Computational Requirments of The Viterbi Algorithm

	ADD	SUB	MUL	DIV	MAX	LKUP
Procedure A	2×2^M	-	4×2^M	-	-	-
Procedure E	2×2^M	-	-	-	2^M	-
Procedure G	-	2^M	-	-	-	1

Table 4: Complexity of The Decoding Algorithm

	Number of Equivalent Additions
Log-MAP algorithm	$48 \times 2^M - 13$
Max-log-MAP algorithm	$28 \times 2^M - 3$
Viterbi algorithm	$10 \times 2^M + 3$

6 Conclusion

There are many applications of ABFT; this paper provides a general method and techniques for employing turbo codes in ABFT, and bounds on the ABFT redundant computations are given. This article proposes a new computing paradigm in order to provide fault tolerance for numerical algorithms. Turbo codes can be used efficiently for detecting the errors in numerical processing systems. This advantage of turbo codes over Convolutional methods of coding is moderately typically over the complete range of might be code rates, that is, coding gain can be achieved at moderate BERs whit long turbo codes of the same rate and approximately the same decoding complexity as Convolutional codes. This paper proposes an efficient method to detect the arithmetic errors using turbo codes at the output compared with an equivalent parity value derived from the input data. Number data processing errors are detected by comparing parity values associated with a turbo code. 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.

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 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.
The advantage of this paper is that errors which are caused by permanent or transient failures in the system can be detected and corrected by using a very low overhead and at the original throughput.
However, the probability of errors in the result increases with the amount of computation. In order to accommodate the contradictory requirements, high complexity and high reliability, the system has to be designed to be fault tolerant. Conventional fault tolerance techniques such as triple modular redundancy (TMR) and triple time redundancy (TTR) suffer from either a high hardware overhead (cost) or time overhead (degraded performance). An important challenge is separating numerical round-off effects from internal hardware failure effects. For future work, our analysis has taken into consideration only un-punctured rate -1/3 symmetric Turbo codes. The problem of interleaver design for punctured codes also necessitates an appropriate interleaver optimisation. Some work has already been done on punctured codes, particularly indicating the usefulness of an odd-even interleaver structure.

References

[1] C. Berrou, A. Glavieux, P. Thitimajshima, *Near Shannon limit error-correcting coding and decoding: Turbo-codes*, in Proceedings of the IEEE International Conference on Communications 2 (1993) 1064-1070.

[2] K. H. Huang, J. A. Abraham, *Algorithm-Based Fault Tolerance for Matrix Operations*, IEEE Trans. Computers 33 (1984) 518-528.

[3] H. Hamidi, A. Vafaei, A. H. Monadjemi, *Algorithm based fault tolerant and check pointing for high performance computing systems*, J. Applied Sci. 9 (2009) 3947-3956.

[4] Z. Chen, *Extending Algorithm-based Fault Tolerance to Tolerate Fail-stop Failures in High Performance Distributed Environments*, Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium, DPDNS'08 Workshop, Miami, FL, USA, April (2008) 14-18.

[5] C. N. Zhang, X. W. Liu, *An algorithm based mesh check-sum fault tolerant scheme for stream ciphers*, International Journal of Communication Networks and Distributed Systems 3 (2009) 217-233.

[6] S. Sundaram, C. N. Hadjicostis, *Fault-Tolerant Convolutional via Chinese Remainder Codes Constructed from Non-Coprime Moduli*, IEEE Transactions on Signal Processing 56 (2008) 4244-4254.

[7] G. Bosilca, R. Delmas, J. Dongarra, J. Langou, *Algorithm-based fault tolerance applied to high performance computing*, Journal of Parallel and Distributed Computing, Elsevier 69 (2009) 410-416.

[8] T. Roche, M. Cunche, J. L Roch, *"Algorithm-Based Fault Tolerance Applied to P2P Computing Networks*, 2 (2009) 144-149, First International Conference on Advances in P2P Systems.

[9] G. Robert Redinbo, *Generalized Algorithm-Based Fault Tolerance: Error Correction via Kalman Estimation*, IEEE Transactions ON Computers 47 (1998) 1111-1119.

[10] G. Robert Redinbo, *"Failure-Detecting Arithmetic Convolutional Codes and an Iterative Correcting Strategy"*, IEEE Transactions on Computers 52 (2003) 1434-1442.

[11] G. Robert Redinbo, *"Wavelet Codes for Algorithm-Based Fault Tolerance Applications"*, IEEE Transactions on Dependable and Secure Computing 7 (2010) 315-328.

[12] A. Moosavie Nia, K. Mohammadi, *A Generalized ABFT Technique Using a Fault Tolerant Neural Network*, Journal of Circuits, Systems, and Computers 16 (2007) 337-356.

[13] D. Costello, S. Lin, *Error Control Coding Fundamentals and Applications*, 2nd edition, Pearson Education Inc., NJ, U.S.A., 2004.

[14] Robert H. Morelos-Zaragoza, *The Art of Error Correcting Coding*, 2nd Edition, John Wiley & Sons, ISBN: 0470015586, 2006.

[15] Andrew J. Viterbi, J. K. Omura, *Principles of Digital Communication and Coding*, McGrawhill, 2-nd Print 1985.

[16] E. R. Berlekamp, *A Class of Convolutional Codes*, Information and Control 6 (1962) 1-13.

[17] J. L. Massey, *Implementation of Burst-Correcting Convolutional Codes*, IEEE Trans. Information Theory 11 (1965) 416-422.

[18] V. Snasel, J. Platos, P. Kromer, N. Ouddane, *"Genetic Algorithms Searching for Turbo Code Interleaver and Solving Linear Ordering Problem*, cisim, 2008 7th Computer Information Systems and Industrial Management Applications 78 (2008) 71-77.

[19] H. El Gamal, A. R. Hammons, *Analyzing the turbo decoder using the Gaussian approximation*, IEEE Transactions on Information Theory 47 (2001) 671-686.

[20] J. G. Proakis, *Digital Communications*, 4th Edition. New York: McGraw Hill, 2001.

[21] P. H. Y. Wu, *On the complexity of turbo decoding algorithms*, in Proceedings of the IEEE Vehicular Technology Conference-Spring 2 (2001) 1439-1443.

[22] J. Hagenauer, F. Hoher, *A Viterbi Algorithm with Soft-Decision Outputs and Its Applications*, Proc. 1989 IEEE Global Teleconim Con& (GLOBECOM89). pp. 47.1.1- 47.1.7, Dallas, Texas, 1989.

[23] J. Hagenauer, E. Offer, L. Papke, *Iterative decoding of binary block and Convolutional codes*, IEEE Trans. Inform. Theory 42 (1996) 429-445.

[24] H. Sadjadpour, N. Sloane, M. Salehi, G. Nebe, *Interleaver design for turbo codes*, IEEE Journal on Selected Areas in Communications 19 (2001) 831-837.

[25] J. Hokfelt, O. Edfors, T. Maseng, *On the theory and performance of trellis termination methods for turbo codes*, IEEE Journal on Selected Areas in Communications 19 (2001) 838-847.

[26] D. Liao, G. Sun, V. Anand, H.Yu, *Reliable Design for Stochastic Multicast Virtual Network in Data Centres*, IETE Technical Review 31 (2014) 327-341.

[27] T. Ma, Y. Chu, L. Zhao, & O. Ankhbayar, *Resource Allocation and Scheduling in Cloud Computing: Policy and Algorithm*, IETE Technical Review 31 (2014) 4-16.

Hodjat Hamidi, born 1978, in shazand Arak, Iran, He got his PhD in computer engineering. His main research interest areas are Information Technology, Fault-Tolerant systems (fault-tolerant computing, error control in digital designs) and applications and reliable and secure distributed systems and e- commerce. Since 2013 he has been a faculty member at the IT group of K. N. Toosi University of Technology, Tehran Iran.