A New Method for Transformation Techniques in Secure Information Systems

Hodjatollah Hamidi*
Department of Industrial Engineering, K. N. Toosi University of Technology, Tehran, Iran
h_hamidi@kntu.ac.ir

Received: 04/Jan/2015 Revised: 03/Jan/2016 Accepted: 23/Jan/2016

Abstract

The transformation technique relies on the comparison of parity values computed in two ways. The fault detection structures are developed and they not only detected subsystem faults but also corrected faults 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 fault can propagate to many output faults. Parity values are the most effective tools used to detect faults occurring in the code stream. In this paper, we present a methodology for redundant systems that allows to detect faults. Checkpointing is the typical technique to tolerate such faults. This paper presents a checkpointing approach to operate on encoded data. The advantage of this method is that it is able to achieve very low overhead according to the specific characteristic of an application. The numerical results of the multiple checkpointing technique confirm that the technique is more efficient and reliable by not only distributing the process of checkpointing over groups of processors. This technique has been shown to improve both the reliability of the computation and the performance of the checkpointing.

Keywords: Transformation Techniques; Information Systems; Redundancy; Checkpointing.

1. Introduction

The checkpointing of approaches considers the specific characteristic of an application and designs fault tolerance schemes according to the specific characteristic of an application [1].

Transformation techniques is a class of approaches which tolerant byzantine failures, in which failed processors continues to work but produce incorrect calculations [2]. The transformation techniques approach transforms a system that does not tolerate a specific type of fault, called the fault-intolerant system, to a system that provides a specific level of fault tolerance, namely recovery [3]. In transformation techniques, applications are modified to operate on encoded data to determine the correctness of some mathematical calculations. The Transformation techniques of approaches can mainly be applied to applications performing linear algebra computations and usually achieves a very low overhead. One of the most important characteristics of this research is that it assume a fail-continue model in which failed processors continues to work but produce incorrect calculations [4-5].

Transformation techniques can be tuned to provide the desired fault tolerance e.g., single error detection, single error correction, etc. For some computations, transformation techniques can be implemented with low overhead as shown in [6] and [7]. Transformation techniques applies error control codes to the data such that errors are detected and in some cases located and corrected. An example of transformation techniques is to encode matrices by adding checksum rows or columns as discussed in [1], [8]. Checksum encoding is used to generate what is called a "checksum matrix" from the original matrix.

Faults in numerical data processing may be detected efficiently by using parity values associated with real number codes, even when inherent round off errors are allowed in addition to failure disruptions. The basic approach for protection was discussed in [9] and has been expanded in many ways since, e.g., [10], [11], [12]. The real number convolutional codes discussed here allow data and parity values to be processed in a continuous fashion unlike block real number codes which require data and parity segmentation. There are many applications where such convolutional codes can provide protection including satellite, communication, signal processing, and large data processing systems.

Transformation techniques for arithmetic and numerical processing operations is based on linear codes. G. Bosilca et al. [13] for high-performance computing, propose a new transformation techniques method based on a parity check coding. Redinbo [14-16] 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

* Corresponding Author

onerous analytical approach to exact measures of the detection performances of the transformation techniques technique applying wavelet codes.

The paper is organized as follows: In section 2, we discuss architecture of the transformation techniques technique. In section 3, we propose the error correction system. In section 4, we discuss Factorization. In section 5, we Analysis Check pointing. In section 6, to be discussed conclusions.

2. Architecture of Transformation Techniques

To achieve fault detection and correction properties of this code in a linear process with the minimum overhead computations [15], we propose the architecture in Fig. 1.

The advantage of transformation technique 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 convolutional codes hold great promise of protecting many data processing subsystems. There are times when the error detection capabilities of transformation techniques 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. Convolutional codes which employ real-number symbols are difficult to decode because of the size of the alphabet such codes find applications in both fault-tolerance support for signal processing subsystems and in channel coding for communication systems. In order to achieve fault detection and correction properties of convolutional code in a linear process with the minimum overhead computations, the architecture is proposed. For error correction purposes, redundancy must be inserted in some form and convolution parity codes will be employed, using the transformation technique. A systematic form of convolutional codes is especially profitable in the transformation technique 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 convolution code in data processing with the minimum additional computations, the block diagram is proposed in. The data processing operations are combined with the parity generating function to provide one set of parity values.

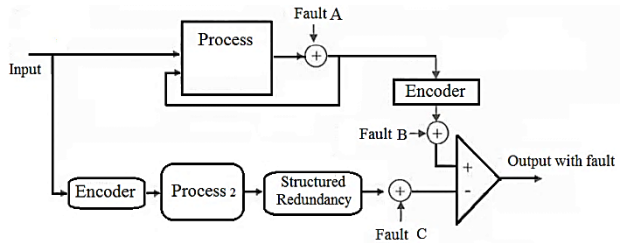

Fig. 1. Our architecture of transformation techniques

We have modeled faults in a linear process block with module fault A while the encoder and structured redundancy faults are modeled with modules B and C. Since these two last faults contribute in syndrome additively we can delete one of them without any degradation. Convolutional codes are usually used over the transmission channels, through which both information and parity bits are sent. The main architecture is similar to a normal transformation techniques scheme except of the structured redundancy and delay line in the information pass which replace the parity generator part of a systematic Convolutional encoder. The upper way is the normal Process data flow which passes through the nonlinear process block and then fed to the Convolutional encoder to make parity sequence the structured redundancy. So the syndrome sequence is a stream of zero or near zero values in normal operation [17].

2.1 Redundant Implementation

In order to avoid replication when constructing fault tolerant dynamic systems, we replace the original system with a larger, redundant system that preserves the state, evolution and properties of the original system - perhaps in some encoded form. We impose restrictions on the set of states that are allowed in the larger dynamic system, so that an external mechanism can perform error detection and correction by identifying and analyzing violations of these restrictions. The larger dynamic system is called a redundant implementation and is part of the overall.

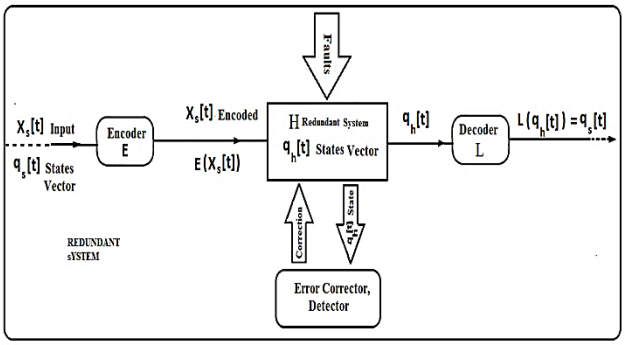

Fig. 2. Fault tolerant structure

Fault tolerant structure shown in Fig. 2,[18], the input to the redundant implementation at time step t, denoted by e(x[t]), is an encoded version of the input x [t] to the original system; furthermore, at any given time step t, the state $q_s[t]$ of the original system can be recovered from the corresponding state $q_h[t]$ of the redundant system through a decoding mapping L (i.e., $q_s[t] = L(q_h[t])$). Note that we require the error detection/correction procedure to be input-independent; so that we ensure the next-state function is *not* evaluated in the error-correcting circuit [19].

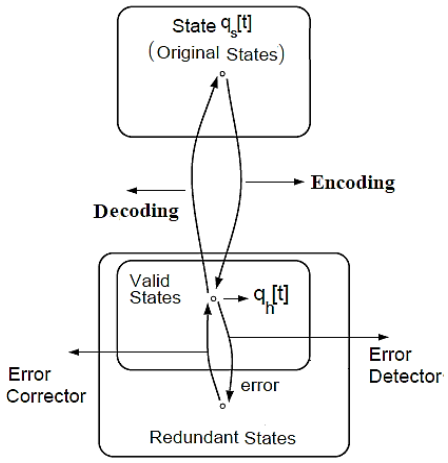

Fig. 3. Reliable state evolution using unreliable error-correction

This approach uses the scheme shown in Fig. 3 but allow failures in both the redundant implementation and the error-correcting mechanism. Clearly, since all components of this construction are allowed to fail, the system will not necessarily be in the correct state at the end of a particular time step. What we hope for, however, is for its state to be within a set of states that *correspond* to the correct one: in other words, if a fault-free error corrector/decoder was available, then we would be able to obtain the correct state from the possibly corrupted state of the redundant system. This situation is shown in Fig. 3, [20]: at the end of each time step, the system is within a set of states that could be corrected /decoded to the actual state (in which the underlying system would be, had there been no failures). Even when the decoding mechanism is not fault-free, our approach is still desirable because it guarantees that the probability of a decoding failure will not increase with time in an unacceptable fashion.

3. Error Correction System

There is no easy analytical approach to accurate measures of the detection performances of the transformation technique using convolution codes. Thus, a series of simulations provide estimates of the probability of detection and miss. Convolutional codes offer a natural protection method for lossless compression systems. A high rate Convolutional code over the ring of real corresponding to the arithmetic format can be used to dictate parity numbers that are inserted periodically in the input to the source encoder, as shown in Fig. 4, [16]. The parity values are compressed along with the normal data and the compressed stream is passed through the channel to the decoder. The decoder extracts the data and the inserted parity numbers. These parity values are compared with locally regenerated parity values. These comparisons detect error conditions and, when appropriate, error correction is engaged. A large class of burst-correcting Convolutional codes produces a single parity value for each group of data numbers. This protection method has a small impact on the overall compressing efficiency, especially for high rate codes.

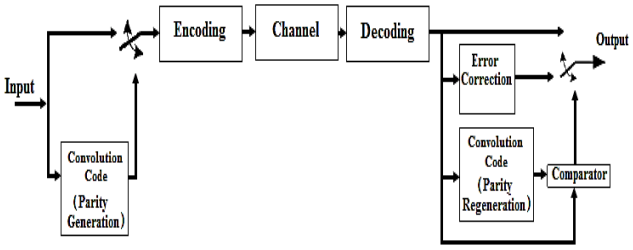

Fig. 4. Joint source-channel coding: embedding Convolutional code parity in compressed and transmitted data.

Powerful efficient Convolutional codes can be used to define arithmetic Convolutional codes that operate on the computational structures of many data processing systems. The encoding and detecting operations employ standard computational resources. When errors at the value level are detected, standard binary decoding algorithms are used in an iterative feedback manner to correct values using syndrome processing methods. The general flow of the feedback decoding method is shown in Fig. 5, [14].

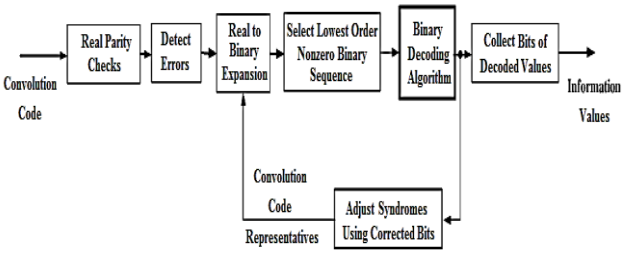

Fig. 5. Iterative syndrome decoding

Random data were encoded into convolutional code words and low level error values with variances' σ_g^2 were added. The high level error values variances' σ_b^2 and probability p were added also. The syndromes of the resulting corrupted code words were analyzed and if any syndrome's component exceeded a threshold ($r = 5\sqrt{\sigma_g^2}$). These error locations were positioned at exact integer index values. The simulations evaluated three convolutional codes, all with 7 parity positions indicating triple error correction capability, of lengths 32, 39, and 50: (32,25), (39,32), (50,43). The low-level error variances σ_g^2 were set at 10^{-6}, a very high choice so as to examine the detrimental effects on the detection and correction operations. The probability of code word error is plotted in Fig. 6 for the three selected convolutional codes. The fixed point input data used m=8 bits while the floating point input was to the precision of MatLab representation. The fixed-point encoded data always had better code word error probabilities.

This section provides typical detection performance results using the designed error detection strategies. The simulation process indicated that round off errors were on the order of 10^{-11} so the necessary thresholds were chosen well above this level. Consider the convolutional code with error injection values modeled by a Gaussian noise source. The experiment results show that when errors

were injected into the system, is on the order of 10^{-10}. The error detection performance is dependent on the error variance and the selected detection threshold. Fig. 7(a) displays three detection performance curves of the 9/7 convolutional code corresponding to single errors with three different variances: $10^{-4}\sigma_0^2$, $10^{-3}\sigma_0^2$, $10^{-2}\sigma_0^2$, where σ_0^2 is the variance of input data.

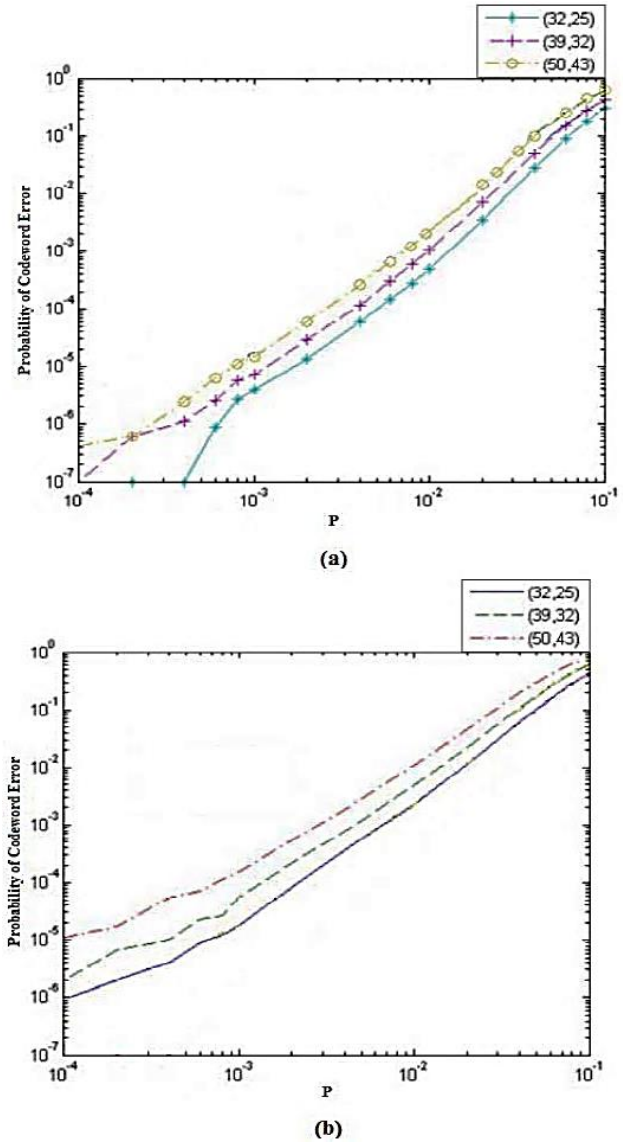

Fig. 6. Probability of Code word Error for Three Codes
(a) The fixed point input data (b) The fixed-point encoded data
($\sigma_g^2 = 10^{-6}$, $\sigma_b^2 = 50$).

The results show that the system has high detection performance when the threshold is in the range from 10^{-10} to 10^{-4} but decreases as the detection threshold increases. Fig. 7(b) shows the range of excellent performance is smaller than that of the forward transform (10^{-10} to 10^{-4}). Table 1 summarizes these simulation results. The detection performance depends on the power of the code supporting the checking capabilities.

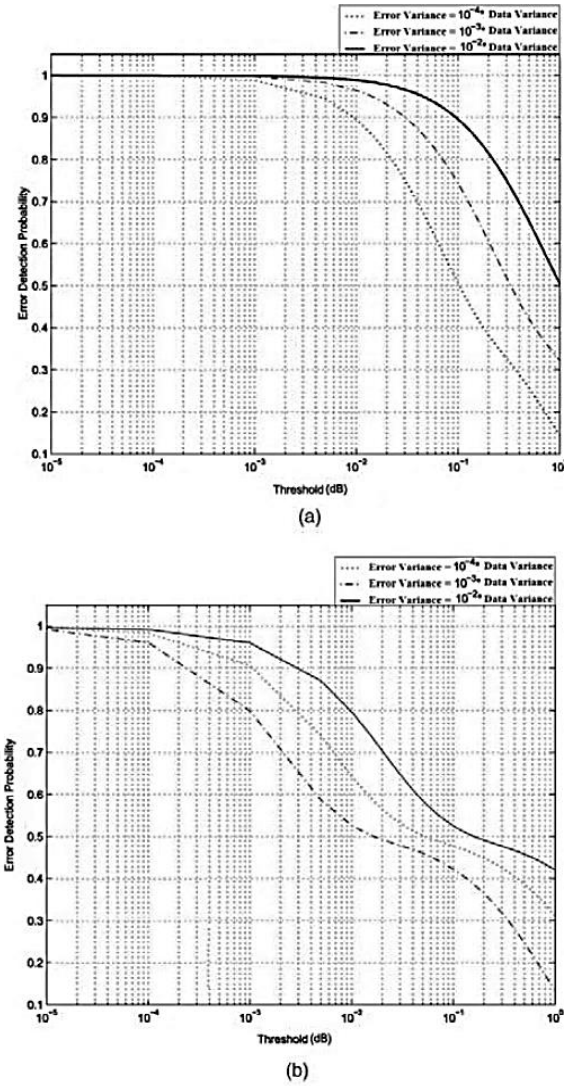

Fig. 7. Error detection performance of the proposed design model for: (a) 9/7 Convolutional code, single error (b) 9/7 Convolutional code, double error. ($10^{-4}\sigma_0^2$, $10^{-3}\sigma_0^2$, $10^{-2}\sigma_0^2$) three different variances:

Table 1. Detection Performance

Single Error			Double Error		
Injection	Detection	Percent	Injection	Detection	Percent
4096	4096	100%	4096	4096	100%
4096	4096	100%	4096	4070	99.3%
4096	4096	100%	2390	1304	54.6%

4. Factorization

4.1 LU Factorization

In LU factorization, an m×n real matrix A is factored into a lower triangular matrix L and a upper triangular matrix U, i.e. PA = LU, where P is a permutation matrix, at each iteration one column block is factored and a permutation matrix P is generated, if necessary, The LU factorization is performed in place, and P is stored as a one-dimensional array of the pivoting indices. Three variants exist for implementing LU factorization on

sequential machines. These three block algorithms of LU factorization can be constructed as follows. Suppose that we have factored A as A = LU. We write the factors in block form as follows:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{21} & U_{31} \\ 0 & U_{22} & U_{32} \\ 0 & 0 & U_{33} \end{bmatrix} \quad (1)$$

With these relationships, we can develop three variants by manipulating the order in which computations are formed and maintaining the final result of computations in place. These variants are called ijk variants [21] or, more specifically, right-looking, top-looking, and left-looking, respectively. They differ in which regions of data are accessed and computed during each reduction step.

4.2 Cholesky Factorization

Cholesky factorization factors an n×n real, symmetric, positive definite matrix A into a lower triangular matrix L and its transpose L^T, i.e., $A = LL^T$ or U^TU where U is upper triangular). Because of the symmetric, positive definite property of the matrix A, Cholesky factorization is also performed in place on either an upper or lower triangular matrix and involves no pivoting. Three different variants of the Cholesky factorization can be developed as above [22].

4.3 QR Factorization

Given an m×n real matrix A, QR factorization factors A such that

$$A = Q\begin{bmatrix} R \\ 0 \end{bmatrix} \quad (2)$$

Where Q is an m×m orthogonal matrix and R an n×n upper triangular matrix Q is computed by applying a sequence of householder transformations to the current column block of the form, $H_i = 1 - \tau_i v_i v_i^T$ where i =1,…., b. In one block QR algorithm Q can be applied or manipulated through the identity $Q = H_1H_2 \ldots H_b = 1 - VTV$ where V is a lower triangular matrix of "householder" vectors V_i and T is an upper triangular matrix constructed from the triangular factors V_i and τ_i of the householder transformations. When the factorization is complete, V is stored in the lower triangular part of the original matrix A, R is stored in the upper triangular part of A, and the $\tau_i' S$ are stored in the diagonal entries of A. The complete details of this algorithm are described in [23] and [24]. Both left- looking and right- looking variants can be constructed [25].

5. Analysis of Checkpointin

The basic checkpointing operation works on a panel of blocks, where each block consists of X floating- point numbers, and the processors are logically configured in a P × Q mesh (See Fig.8,[26]) .The processors take the checkpoint with a combine operation of XOR or addition. This works in a spanning- tree fashion in three parts. The checkpoint is first taken row wise, then taken column wise, and then sent to PC. The first part therefore takes [log P] steps, and the second part takes [log Q] steps. Each step consists of sending and then performing either XOR or addition on X floating- point numbers. The third part consists of sending the X numbers to PC.

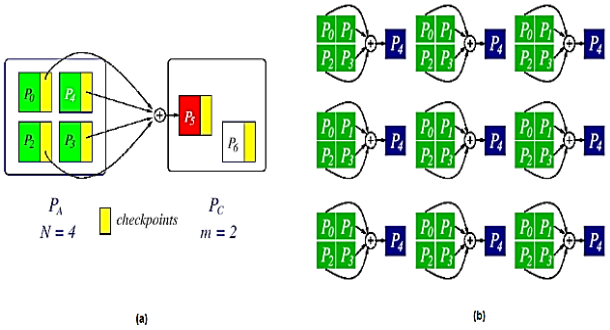

Fig. 8. (a) Single- failure recovery model: after a failure, (b) Checkpointing the matrix of (a).

We define the following terms:

γ: The time for performing a floating- point addition or XOR operation.

α: The startup time for sending a message.

β: The time to transfer one floating- point number.

The first part takes [log P] $(\alpha + X(\beta + \gamma))$, the second part takes [log Q] $(\alpha + X(\beta + \gamma))$, and the third takes $(\alpha + X\beta)$.

5.1 Implementations and Performance Evaluation

For all of the implementations, the following set of tests was performed and timed:

- Failure free algorithm without checkpointing.
- Fault tolerant implementation with single checkpointing.
- Single checkpointing implementation with one random failure.
- Fault tolerant implementation with multiple checkpointing.
- Multiple checkpointing implementations with multiple failures.

Note that the failures were forced to occur at the last iteration before the first checkpoint. The performance results of the implementations are evaluated in terms of the following parameters:

- Total elapsed wall-clock times of the algorithms in seconds (T_A, T).
- Checkpointing and recovery overheads in seconds (T_C, T_R).
- Checkpointing interval in iterations $(K, N_C = n/Kb)$.
- Average checkpointing interval in seconds $((T - T_{init})/N_C)$.
- Average checkpointing overhead in seconds $(\Delta T_C = T_C - T_{init})$.
- Total size of checkpoints in bytes (M).

- Extra memory usage in bytes (M_C).
- Checkpointing rate in bytes per second (R)

The checkpointing performed in these implementations consists of data communication and either XOR or addition of floating-point numbers. We define the checkpointing rate R as the amount of data checkpointed in bytes per second. This metric has been used to evaluate the performance of various checkpointing schemes [29-34]. In our case, the checkpointing rate is determined experimentally based on our analytic models of the fault-tolerant implementations. The total checkpointing overhead of the left-looking variant is too high compared with the right-looking variant without checkpointing (See Figures 9, 10 and 11). This checkpointing rate is used to compare the performance of the different fault tolerance techniques, Figures 12 and 13 plots the checkpointing rate for each implementation.

Parity-Based Technique: For the parity-based matrix operations, the total percentage overhead of checkpointing decreases as the problem size n increases. The total overhead of recovery is dominated by the time for taking the bitwise exclusive- or of each processor's entire data. The time it takes to recover does not depend upon the location of the failure. The multiple checkpointing implementations show performance improvement. LU factorizations benefit relatively more from the multiple checkpointing because of pivoting. Figure 10 shows the checkpointing rate experimentally determined for each implementation .This presents the overall performance of the parity-based technique for matrix operations.

	n	T_A (sec)	N_C+1	T (sec)	$\frac{T-T_{init}}{N_C}$ (sec)	T_C (sec)	T_{init} (sec)	$\frac{\Delta T_C}{N_C}$ (sec)	T_R (sec)
single checkpointing	1500	220	32	232	7.3	19	8	0.5	8
	3000	1530	66	1430	21.6	62	29	0.9	27
	4500	2610	102	2010	19.2	151	71	1.4	69
	6000	4900	144	5100	35	280	103	1.9	92
	7500	7100	180	6900	38	489	183	1.0	151
	n	T_A (sec)	N_C+1	T (sec)	$\frac{T-T_{init}}{N_C}$ (sec)	T_C (sec)	T_{init} (sec)	$\frac{\Delta T_C}{N_C}$ (sec)	T_R (sec)
multiple checkpointing	1500	220	32	220	6.9	17	5	0.5	6
	3000	1530	66	1200	18.1	46	21	0.7	20
	4500	2610	102	1902	18.3	101	55	1.0	46
	6000	4900	144	4800	33	236	79	1.6	70
	7500	7100	180	6480	35.5	409	132	0.8	111

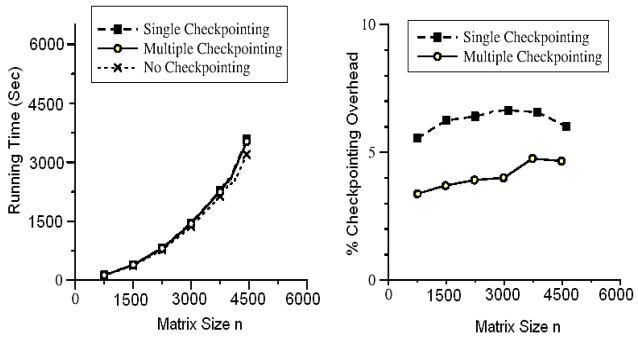

Fig. 9. Left-looking LU, timing results

	n	T_A (sec)	N_C+1	T (sec)	$\frac{T-T_{init}}{N_C}$ (sec)	T_C (sec)	T_{init} (sec)	$\frac{\Delta T_C}{N_C}$ (sec)	T_R (sec)
single checkpointing	1500	220	32	228	7.2	8	4	0.1	4
	3000	1530	66	1558	23.7	28	15	0.2	13
	4500	2610	102	2675	26.2	65	32	0.3	28
	6000	4900	144	5022	34.7	122	53	0.4	50
	7500	7100	180	7330	40.5	230	74	0.9	66
	n	T_A (sec)	N_C+1	T (sec)	$\frac{T-T_{init}}{N_C}$ (sec)	T_C (sec)	T_{init} (sec)	$\frac{\Delta T_C}{N_C}$ (sec)	T_R (sec)
multiple checkpointing	1500	220	32	226	7.2	6	2	0.1	1
	3000	1530	66	1548	23.7	18	6	0.2	5
	4500	2610	102	2659	26.1	49	20	0.3	17
	6000	4900	144	5005	34.8	105	31	0.5	27
	7500	7100	180	7291	40.4	191	55	0.8	52

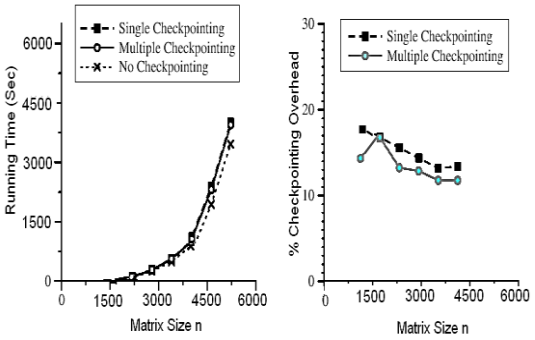

Fig. 10. Left-looking Cholesky, timing results

	n	T_A (sec)	N_C+1	T (sec)	$\frac{T-T_{init}}{N_C}$ (sec)	T_C (sec)	T_{init} (sec)	$\frac{\Delta T_C}{N_C}$ (sec)	T_R (sec)
single checkpointing	1500	220	32	244	7.6	24	8	0.5	6
	3000	1530	66	1603	24.2	73	30	0.6	27
	4500	2610	102	2750	26.7	140	54	0.8	50
	6000	4900	144	5094	38.8	194	78	0.8	71
	n	T_A (sec)	N_C+1	T (sec)	$\frac{T-T_{init}}{N_C}$ (sec)	T_C (sec)	T_{init} (sec)	$\frac{\Delta T_C}{N_C}$ (sec)	T_R (sec)
multiple checkpointing	1500	220	32	240	7.6	20	5	0.5	4
	3000	1530	66	1582	24	52	22	0.5	19
	4500	2610	102	2725	26.6	115	40	0.7	39
	6000	4900	144	5048	34.9	148	55	0.6	51

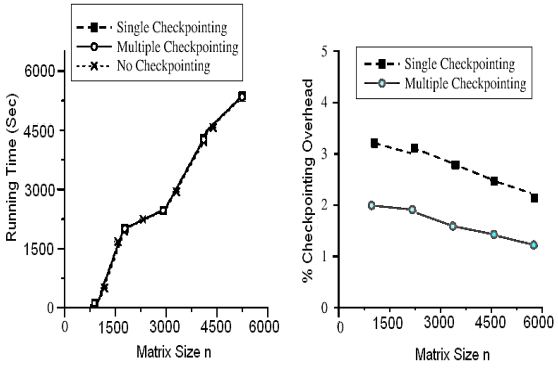

Fig. 11. Left-looking QR, timing results

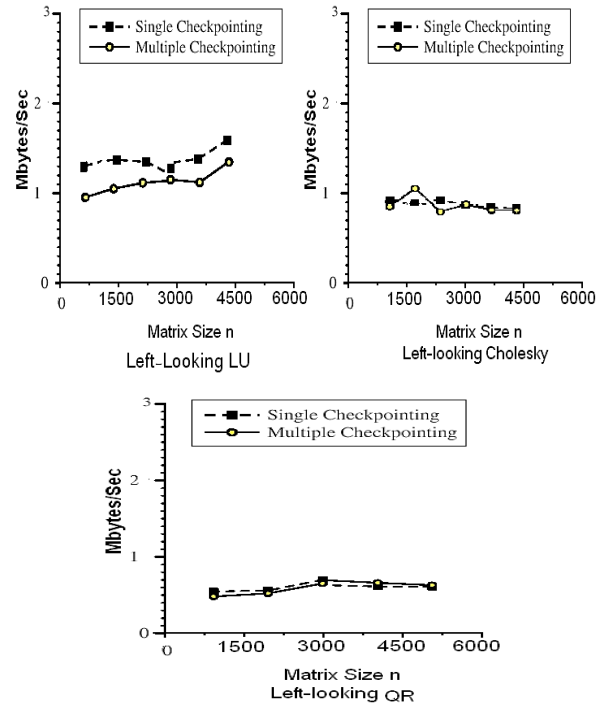

Fig. 12. Experimental checkpointing rate

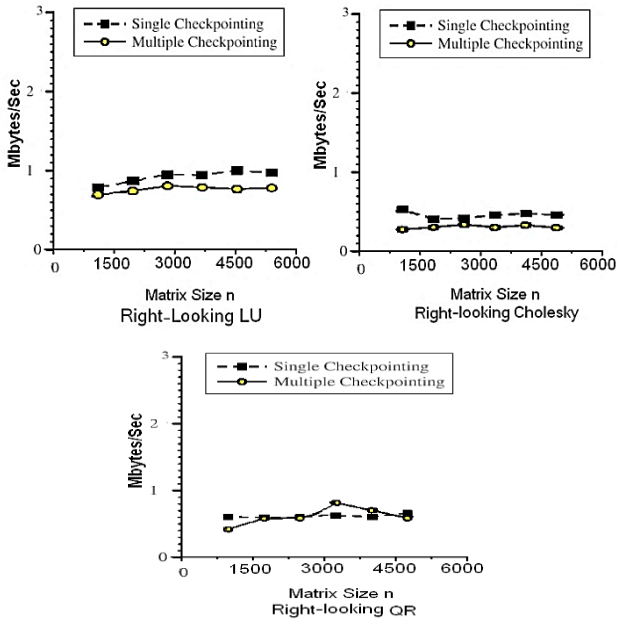

Fig. 13. Experimental checkpointing rate

Since the measured peak bandwidth of the network is 64 Mbits per second, we expect that the checkpointing rate should be somewhat lower than 8 Mbytes per second considering synchronization, copying, performing XOR, and message latency and network contention. As shown in Figures 12 and 13 the checkpointing rate determined experimentally is between 2 and 4 Mbytes per second for all the matrix operations. The right-looking variant performs the best among the failure-free variants of each factorization because it benefits from less communication and more parallelism than the others. However, for the LU and Cholesky factorizations, the left-looking variants with checkpointing perform better than the right-looking variant with checkpointing. For the QR factorization, no top-looking variant exists, and the left-looking variant performs much slower than the right-looking variant. The total checkpointing overhead of the left-looking variant is too high compared with the right-looking variant without checkpointing (Figure 9-11).

6. Conclusions

The transformation technique 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 transformation technique 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.

In This paper presents a model for executing certain scientific computations on a changing distributed computing platform .The model allows a distributed computation to run on a platform where individual processors may leave due to failures, unavailability, or heavy load, and where processors may enter during the computation. The model provides an interesting way to allow reliability in computations per formed on networks of computers.

Systematic codes for data protection using the parity comparison method can be determined from general Convolutional codes by manipulating the matrix associated with such codes.

These operations involve straightforward matrix operations similar to those supporting the normal matrix forms.

The advantage of this method is that it is able to achieve very low overhead according to the specific characteristic of an application. The limitation of this method is that it is non-transparent and has to be designed according to the specific characteristic of an application.

References

[1] H. Hamidi, A. Vafaei, A.H. Monadjemi, “Algorithm based fault tolerant and check pointing for high performance computing systems,”J.Applied Sci., 9:3947-3956, 2009.

[2] H. Hamidi, A.Vafaie, S. A. Monadjemi “Analysis and design of an ABFT and parity-checking technique in high performance computing systems” Journal of Circuits,

Systems, and Computers (JCSC), JCSC Vol.21, No. 3, May 2012.
[3] H. Hamidi, A.Vafaie, S. A. Monadjemi. Analysis and evaluation of a new algorithm based fault tolerance for computing systems. International Journal of Grid and High Performance Computing, 4(1), 37–51. 2012.
[4] H. Hamidi, A.Vafaie, S. A. Monadjemi "A Framework for ABFT Techniques in the Design of Fault-Tolerant Computing Systems", EURASIP Journal on Advances in Signal Processing, Springer, Vol.2011:90, October 2011.
[5] H. Hamidi, A.Vafaie, S. A. Monadjemi, "A Framework for Fault Tolerance Techniques in the Analysis and Evaluation of Computing Systems" International Journal of Innovative Computing, Information and Control (IJICIC), Vol.8, No.7, July 2012.
[6] K.H. Huang, J.A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations." IEEE Trans. Computers, vol. 33, pp. 518-528, 1984.
[7] 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 14-18, 2008.
[8] 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, Vol.3, No.3, pp.217-233, June 2009.
[9] S. Sundaram, C. N. Hadjicostis, "Fault-Tolerant Convolutional via Chinese Remainder Codes Constructed from Non-Coprime Moduli,"IEEE Transactions on Signal Processing, Vol. 56, No. 9, pp. 4244-4254, September 2008.
[10] T.Roche, M. Cunche, J.L Roch, "Algorithm-Based Fault Tolerance Applied to P2P Computing Networks," ap2ps, pp.144-149, 2009 First International Conference on Advances in P2P Systems, 2009.
[11] D. Costello, S. Lin, Error Control Coding Fundamentals and Applications, 2nd edition, Pearson Education Inc., NJ, U.S.A., 2004.
[12] Robert H. Morelos-Zaragoza, The Art of Error Correcting Coding, 2nd Edition, John Wiley & Sons, ISBN: 0470015586, 2006.
[13] G.Bosilca, R.Delmas, J. Dongarra, J. Langou, "Algorithm-based fault tolerance applied to high performance computing,"Journal of Parallel and Distributed Computing, Elsevier, Vol.69, No.4, pp.410-416, April 2009.
[14] G. Robert Redinbo, "Wavelet Codes for Algorithm-Based Fault Tolerance Applications," IEEE Transactions on Dependable and Secure Computing, vol. 7, no. 3, pp. 315-328, July-Sept. 2010.
[15] G.Robert Redinbo, "Generalized Algorithm-Based Fault Tolerance: Error Correction via Kalman Estimation", IEEE Transactions ON Computers, vol. 47, no. 6, Jun 1998.
[16] G. Robert Redinbo, "Failure-Detecting Arithmetic Convolutional Codes and an Iterative Correcting Strategy," IEEE Transactions on Computers, vol. 52, no. 11, pp. 1434-1442, Nov. 2003.
[17] S.Veeravalli, "Fault tolerance for arithmetic and logic UNIT," IEEE SOUTHEASTCON,'09, 2009, pp. 329 – 334.
[18] J. Choi, J.J. Dongarra, S. Ostrouchov, A. P. Petitet, D. W.Walker, and R. C.Whaley. "The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines," Scientific Programming, 1996.
[19] J. Choi, J. J. Dongarra, and D.W.Walker, "PB-BLAS: A set of parallel block basic linear algebra subprograms," Concurrency Practice and Experience, 1996.
[20] J. J. Dongarra, J, Du Croz, S. Hammarling, and I. Duff, "A set of level 3 basic linear algebra subprograms," ACM Transactions on Mathematical Software, 18(1):pp.1-17, 1990.
[21] S. Ekici, S. Yildirim, M.Poyraz, "A transmission line fault locator based on Elman recurrent networks," Applied Soft Computing, Elsevier, Volume 9, Issue 1, pp.341-347, 2009.
[22] X.She, S.Trimberger, "Scheme to minimize short effects of single-event upsets in triple-modular redundancy (TMR), IET Computers & Digital Techniques, 4(1):50, 2010.
[23] J. J. Dongarra and R. C. Whaley, A user's guide to the BLACS v1.0.LAPACK Working Note 94, Technical Report CS-95-281, University of Tennessee, 1995.
[24] A. A.Kumar, A. Makur, "Improved coding-theoretic and subspace-based decoding algorithms for a wider class of DCT and DST codes," IEEE Transactions on Signal Processing Vol. 58, Issue 2, pp. 695-708, 2010.
[25] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The performance of consistent check pointing," In 11th Symposium on Reliable Distributed Systems, 1992, pp. 39-47.
[26] R.H. Morelos-Zaragoza, "The Art of Error Correcting Coding, Second Edition," John Wiley & Sons, Ltd. ISBN: 0-470-01558-6, 2006.
[27] J. S. Plank and K. Li. Ickp, "A consistent checkpointer for multicomputer," IEEE Parallel & Distributed Technology, 2(2):62-67, 1994.
[28] J.Y. Jou and J. A.Abraham, "Fault tolerant matrix arithmetic and signal processing on highly concurrent computing structures," Proc. IEEE, vol, no.5, pp.732-741, 1986.
[29] C. N. Hadjicostis, "Coding Approaches to Fault Tolerance in Dynamic Systems," Ph.D thesis, EECS department, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1999.
[30] C. N. Hadjicostis, "Coding Approaches to Fault Tolerance in Combinational and Dynamic Systems," Boston, Massachusetts: Kluwer Academic Publishers, 2002.
[31] Y. Kim, "Fault Tolerant Matrix Operations for Parallel and Distributed Systems," Ph.D dissertation, Univ. of Tennessee, June 1996.
[32] J. S. Plank, Y. Kim, and J. Dongarra, "Algorithm-based diskless checkpointing for fault tolerant matrix operations," In 25th International Symposium on Fault-Tolerant Computing, Pasadena, CA, June 1995.
[33] H. Hamidi, An Approach to Fault Detection and Correction in Design of Fault Tolerant Computing Systems Using of Turbo codes, International Journal of Industrial Mathematics, 2016, (Accepted for Publication).

Hodjatollah Hamidi, born 1978, in shazand Arak, Iran, He got his Ph.D 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. Department of Industrial Engineering , K. N. Toosi University of Technology.