5th SASTech
Iran |Mashhad
May 12 - 17 | 2011
5th Symposium on Advances in Science & Technology

# An Approach of Algorithm Based Fault Tolerance for High Performance Computing Systems

H.Hamidi
Department of Electronics Engineering
Islamic Azad University -Doroud Branch
hamidi@iau-doroud.ac.ir
Paper Reference Number: 1112-331
Name of the Presenter: H.Hamidi

**Abstract**

We present a new approach to algorithm based fault tolerance (ABFT) for High Performance Computing system. The Algorithm Based Fault Tolerance 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. We have implemented a systematic procedure for introducing structured redundancy into ABFT. Algorithm Based Fault Tolerance has been recommending as a cost-effective concurrent error detection scheme. It proposes a novel computing paradigm to provide fault tolerance for numerical algorithms. To that end, a matrix-based model has been developed and, based on that, algorithms for both the design and analysis of ABFT systems are formulated.

**Key words:** Algorithm Based Fault Tolerance (ABFT), Checkpointing, Error Correction, Matrix operations.

---

## 1. INTRODUCTION

Algorithm-based fault tolerance (ABFT), proposed by Huang and Abraham (1984), is a fault tolerance scheme that uses Concurrent Error Detection (techniques at a functional level). System level applications of ABFT techniques have also been investigated (Acree, Ullah, Karia, Rahmeh, & Abraham, 1993; Banerjee, Rahmeh, Stunkel, Nair, Roy, & Abraham, 1990). These techniques assume a general fault model which allows any single module in the system to be faulty (Huang, & Abraham, 1984). ABFT is widely applicable and it has proved its cost-effectiveness especially when applied to array processors (Jou, & Abraham, 1986). Fault detection and diagnosis are integral parts of any fault tolerance scheme. An algorithm executing on a multiple process system is specified as a sequence of operations performed on a set of processes in some discrete time steps.

**CHARACTERISTICS OF ABFT:** 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.

*Signal* processing has been the major application area of ABFT until now, even though the technique is applicable in other types of computations as well. Since the major computational requirements for many important real-time signals processing tasks can be formulated using a common set of matrix computations, it is important to have fault tolerance techniques for various matrix operations (Nair, 1990; Nair, & Abraham, 1990). Coding techniques based on ABFT have already been proposed for various computations such as matrix operations (Huang, & Abraham, 1984; Chen, &Dongarra, 2008; Ashouei, & Chatterjee, 2010), FFT (Jou ,& Abraham,1988; Biernat,2010), QR factorization, and singular value decomposition (Salfner , Lenk, & Malek,2010). Real number codes such as the Checksum (Huang, & Abraham, 1984) and Weighted Checksum codes (Jou ,& Abraham,1986) have been proposed for fault-tolerant matrix operations such as matrix transposition, addition, multiplication and matrix-vector multiplication. Application of these techniques in processor arrays and multiprocessor systems has been investigated by various researchers (Banerjee, Rahmeh, Stunkel, Nair, Roy, & Abraham, 1990). Figure 1, (Moosavie Nia, & Mohammadi, 2007), shows the basic architecture of an ABFT system. Existing techniques use various coding schemes to provide information redundancy needed for error detection and correction. As a result this encoding/decoding must be considered as the overhead introduced by ABFT. The coding algorithm is closely related to the running process and is often defined by real number codes generally of the block types (Baylis, 1998). Systematic codes are of most interest because the fault detection scheme can be superimposed on the original process box with the least changes in the algorithm and architecture. In most previous ABFT applications, the process to be protected is often a linear system. In this paper we assume a more common case consisting linear or nonlinear systems but still constrain ourselves to static systems.

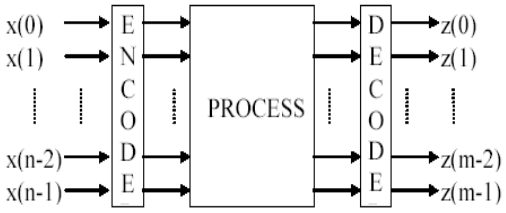



Fig.1**:** General architecture of ABFT

Various methods such as checksum encoding, weighted checksum encoding and average checksum codes have been proposed for fault-tolerant matrix operations. These encoding schemes are especially suitable for computations in processor arrays (Hakkarinen, & Chen, 2010). The use of the checksum codes is limited due to the inflexibility of the encoding schemes and also due to potential numerical problems. Numerical errors may also be misconstrued as errors due to physical faults in the system. A generalization of the existing schemes has been suggested as a solution to these shortcomings .

## 2. ARCHITETURE OF ABFT

To achieve fault detection and correction properties of this code in a linear process with the minimum overhead computations (Moosavie Nia, & Mohammadi, 2007), we propose the architecture in figure 2.

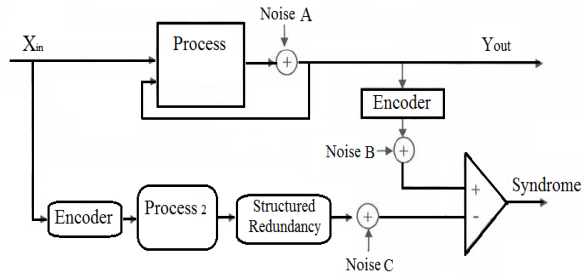



Fig.2**:** Our architecture of ABFT

We have modeled faults in a linear process block with module noise A while the encoder and structured redundancy noises are modeled with modules B and C. Since these two last noises contribute in syndrome additively we can delete one of them without any degradation (Moosavie Nia, & Mohammadi, 2007). 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 ABFT 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 (Moosavie Nia, & Mohammadi, 2007).

## 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

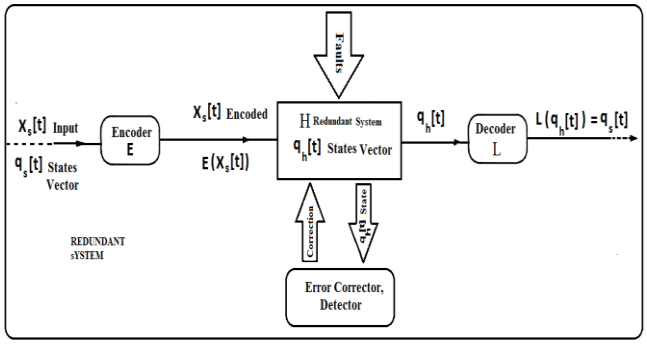



Fig.3**:** Fault tolerant structure

Fault tolerant structure shown in Figure 3,( Hadjicostis ,& Verghese,2002; Hadjicostis,1999), 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, (Hadjicostis, & Verghese, 2002; Hadjicostis, 1999).

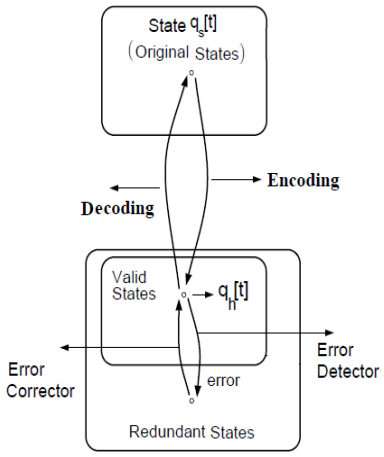



Fig. 4**:** Reliable state evolution using unreliable error-correction

This approach uses the scheme shown in Figure 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 Figure 4, (Hadjicostis, 1999, 2002): 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. As long as the redundant state is within the set of states that represent the actual (underlying) state, the decoding at each time step will be incorrect with a *fixed* probability, which depends only on the reliability of the decoding mechanism and does not diminish as the dynamic system evolves in time.

## 3. FACTORIZATION

### 3.1. 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.

### 3.2. 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 (Kim, 1996; Plank *et al.*, 1995).

### 3.3. QR FACTORIZATION

Given an m ×n real matrix A, QR factorization factors A such that $A = Q\begin{bmatrix} R \\ 0 \end{bmatrix}$ ، here 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 $Q = H_1 H_2 ... H_b = 1 - VTV^T$ 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 $\tau_i$ 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.

## 4. ANALYSIS OF CHECKPOINTING

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 Figure 5, (Kim, 1996)) .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.

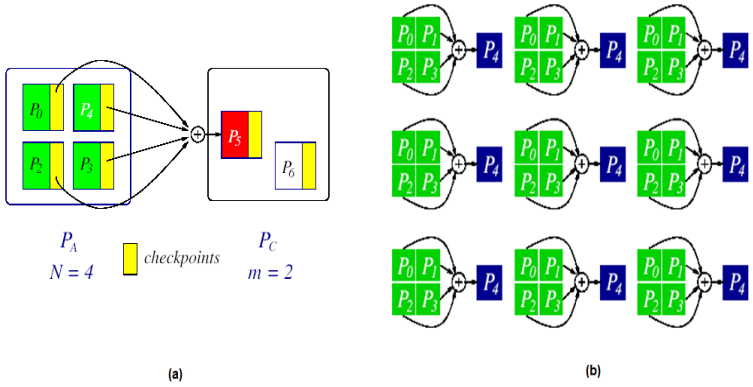



Fig. 5: (a) Single- failure recovery model: after a failure, (b) Checkpointing the matrix of (a).

We define the following terms:
$\gamma$ : The time for performing a floating- point addition or XOR operation.
α**:** The start up 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. IMPLEMENTATIONS AND PERFORMANCE EVALUTION

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

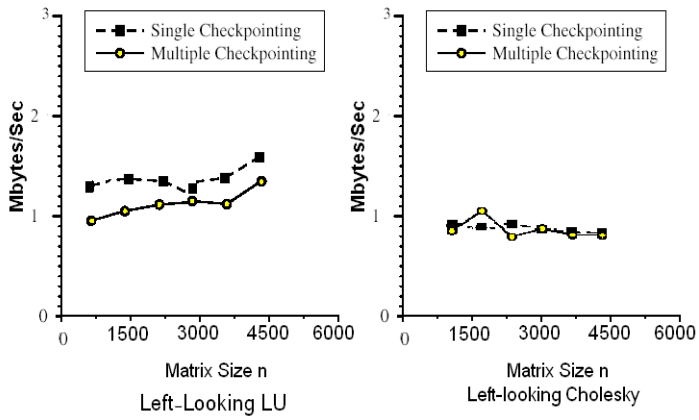
- Failure free algorithm without checkpointing.

- 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 (Elnozahy, Johnson, & Zwaenepoel, 1992). 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 .This checkpointing rate is used to compare the performance of the different fault tolerance techniques, Figures 6 and 7 plots the checkpointing rate for each implementation.



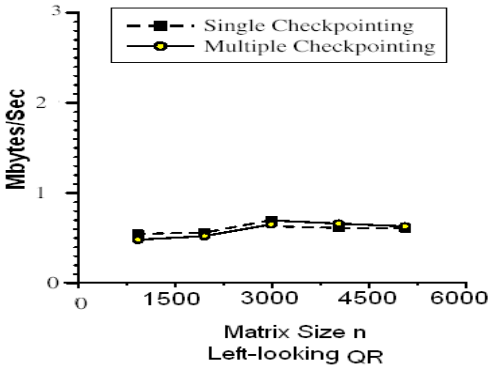



Fig. 6: Experimental checkpointing rate

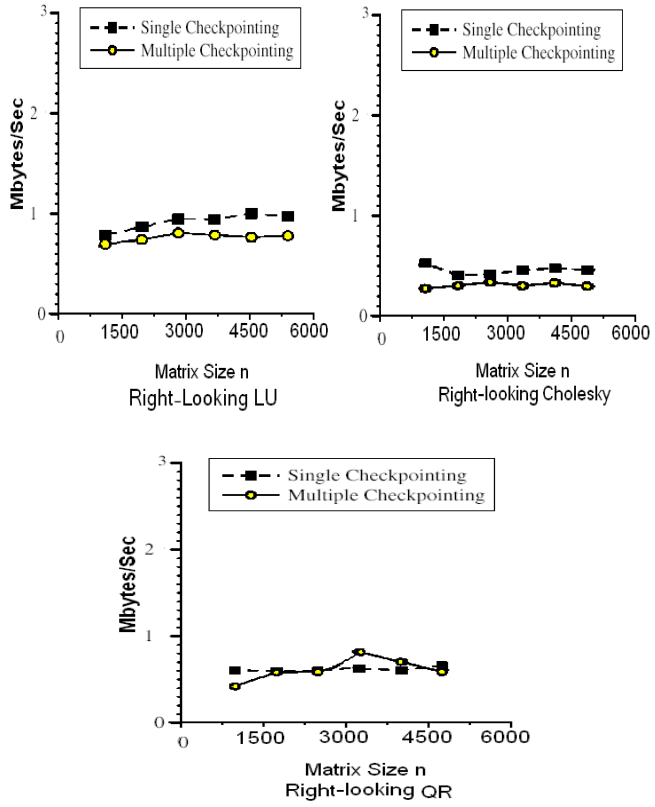



Fig. 7: Experimental checkpointing rate

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 6 and 7 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.

## 6. CONCUSIONS

The fault tolerant matrix operations can be characterized as follows: Very low overhead while checkpointing at a relatively fine grain interval. Robust and easy to incorporate this technique into numerical algorithms, Checkpointing and recovery does not cause numerical problems such as overflow and underflow. Block size has little impact on checkpointing and recovery overhead. Usefulness is limited to those matrix operations in which a moderate amount of data is modified between two checkpoints. 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 and rollback recovery over groups of processors but also by tolerating multiple failures in one of each group of processors. This technique has been shown to improve both the reliability of the computation and the performance of the checkpointing and recovery. In particular, for the checksum and reverse computation based implementations, multiple checkpointing could reduce the checkpointing and recovery overhead without using more memory. In addition, the probability of overflow, underflow, and cancellation error can be reduced. Finally, it is easier to develop fault tolerant implementations when multiple checkpointing processes are used for checkpointing.

## References

Acree,R. K. and Nasr Ullah, Karia,A. and J. T. and Rahmeh , and Abraham, J. A.( Mar. 1993) , An Object-Oriented Approach for Implementing Algorithm- Based Fault Tolerance, 12th Annual International Phoenix Computers and Communications Conference , (pp. 210-216).

Ashouei ,M. and Chatterjee,A.( October 2009) , Checksum-based probabilistic transient-error compensation for linear digital systems, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.17, No.10, pp.1447-1460 .

Banerjee, P.,and Rahmeh,P. J. T. and Stunkel, C. B. and Nalr,V. S. S. and Roy, K. and Abraham, J. A. (Sept, 1990) , Algorithm-based fault tolerance on a hypercube multiprocessor, IEEE Transactions on Computers, vol. 39, no. 9, 1132-1145.

Baylis, J. (1998.) Error- Correcting Codes: A Mathematical Introduction, Chapman and Hall LTD.

Biernat, J. (2010), Fast fault-tolerant adders, International Journa*l Critical Computer-Based Systems*, Vol. 1, Nos. 1/2/3, pp.117–127.

Chen,Z. and Dongarra,J.( Apr. 2008) , Algorithm-Based Fault Tolerance for Fail-Stop Failures, IEEE Transactions on Parallel and Distributed Systems, vol. 19, no. 12, pp. 1628-1641.

Elnozahy, E. N. and Johnson, D. B. and Zwaenepoel,W. (October 1992) The performance of consistent checkpointing, In 11th Symposium on Reliable Distributed Systems, pages 39-47.

Hadjicostis, C. N. (1999), Coding Approaches to Fault Tolerance in Dynamic Systems, Ph.D. thesis, EECS department, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Hadjicostis, C. N. (2002), *Coding Approaches to Fault Tolerance in Combinational and Dynamic Systems*, Boston, Massachusetts: Kluwer Academic Publishers.

Hadjicostis ,C. N. and Verghese,G. C. (January 2002). Encoded dynamics for fault tolerance in linear £nite-state machines, *IEEE Transactions on Automatic Control*, vol. 47,pp. 189–192.

Hakkarinen , D. and Chen,Z. "Algorithmic Cholesky Factorization Fault Recovery," Proceedings of the 24th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2010), Atlanta, GA, USA, April 19-23, 2010.

Huang, K. H. and Abraham, J. A, (1984), Algorithm-based fault tolerance for matrix operations, IEEE Transactions on Computers, vol. C-33:518-528.

Jou,J. Y. and Abraham, J. A., (May 1986), Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures, Proc. IEEE, vol. 74,no.5, pp. 732-741.

Jou,J. Y. and Abraham, J. A.( May 1988) , Fault-tolerant FFT networks, IEEE Transactions on Computers., vol. 37, pp. 548-561.

Kim,Y.( June 1996) , Fault Tolerant Matrix Operations for Parallel and Distributed Systems, PhD dissertation, Univ. of Tennessee.

Moosavie Nia, A.and Mohammadi, K. (2007): A Generalized ABFT Technique Using a Fault Tolerant Neural Network. Journal of Circuits, Systems, and Computers 16(3): 337-356.

Nair, V. S. S, and. Abraham, J. A, (April, 1990), Real number codes for fault-tolerant matrix operations on processor arrays, IEEE Transactions on Computers, pp. 426-435.

Plank, J. S. and Kim,Y. and Dongarra ,J.( June 1995) , Algorithm-based diskless checkpointing for fault tolerant matrix operations, In 25th International Symposium on Fault-Tolerant Computing, Pasadena, CA.

Plank, J. S. and Li. Ickp, K. (Summer 1994), A consistent check pointer for multicomputer, IEEE Parallel & Distributed Technology, 2(2):62-67.

Salfner, and Lenk, M and Malek (March 2010) , "A survey of online failure prediction methods,"ACM Computing Surveys (CSUR), Volume 42, Issue 3 .