ALGORITHM-BASED FAULT TOLERANCE: A PERFORMANCE PERSPECTIVE BASED ON ERROR RATE

Ahmad A. Al-Yamani, Nahmsuk Oh and Edward J. McCluskey
Center for Reliable Computing, Stanford University
{alyamani, nsoh, ejm@crc.stanford.edu}

Abstract

In Algorithm-based fault tolerance (ABFT), the fault tolerance scheme is tailored to the algorithm performed. Most of the previous studies that compared various ABFT schemes considered only their error detection and correction capabilities. Some previous studies looked at the overhead in general but no previous work –as far as we know– compared different ABFT schemes considering performance as the main metric. In this work, we compare the performance of two ABFT error recovery schemes: recomputing vs. correction, for different error rates. We consider errors that happen during computation as well as those that happen during the error detection, location and correction process. The metrics we use are *success ratio* and *completion time*. Results show that multiple error correction using ABFT has worse performance than single error correction. They also show that error rate is an essential factor in making one scheme better than another in terms of performance.

1. Introduction

Errors can occur with different rates depending on the environments where computing systems are operated. For example, satellites experience error rates based on their altitude and location. This variation makes different schemes more appropriate in different environments from a performance perspective.

Performance is a critical metric for data processing applications. Errors influence the rate at which these applications run. The influence of errors on the performance of these systems depends on the recovery scheme applied. ABFT was invented for providing low-overhead recovery in data processing applications.

In this work we model ABFT for matrix operations considering transient faults in hardware. We consider three matrix operations that span a wide range of applications. These operations are matrix multiplication, LU decomposition and matrix inversion. We compare recovery by recomputing vs. ABFT correction in terms of performance for various error rates.

The main contributions of this work are: 1) A performance-based comparison between error recovery by recomputing vs. error correction for various error rates, 2) The effect on performance of correction capability (code distance) in ABFT is quantified by simulation, and 3) It is shown by simulation that multiple error correction has a negative impact on both *completion time* and *success ratio.*

Definition 3.1: Success Ratio (*SR*) is defined as the fraction of iterations in which correct results were produced. An iteration corresponds to processing one frame of input data.

Definition 3.2: Completion Time (*CT*) is defined as the amount of time it took the program to perform a fixed number of iterations.

In Sec. 2, Algorithm-based fault tolerance is explained. In Sec. 3, we review the literature related to this work. Section 4 discusses our fault injection scheme. Section 5 discusses and analyzes the simulation results. Section 6 is the summary and conclusions.

2. Algorithm-Based Fault Tolerance

ABFT can be tuned to provide the desired fault tolerance e.g., single error detection, single error correction, etc. For some computations, ABFT can be implemented with low overhead as shown in [1] and [2]. ABFT applies error control codes to the data such that errors are detected and in some cases located and corrected.

An example of ABFT is to encode matrices by adding checksum rows or columns as discussed in [3]. Checksum encoding is used to generate what is called a “checksum matrix” from the original matrix.

Let A be an n×n matrix. Define d unique linearly independent n×1 weights vectors $w^{(i)}$, $i = 1, 2, \dots d$, with elements $w^{(i)}{}_j$, $j = 1, 2, \dots, n$.

Definition 2.1: Define an n×(n+d) weights matrix W as

$$W = \begin{bmatrix} 1 & \cdots & 1 & w_1^{(1)} & \cdots & w_1^{(d)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cdots & 1 & w_n^{(1)} & \cdots & w_n^{(d)} \end{bmatrix}$$

Define a weighted row checksum matrix $A_{rw} = A \times W$, a weighted column checksum matrix $A_{cw} = W^T \times A$, and a weighted full checksum matrix $A_{fw} = W^T \times A \times W$.

With a proper selection of weights in W, up to d errors can be detected and up to $\lfloor d/2 \rfloor$ errors can be corrected in every column (row) of A_{cw} (A_{rw}) [4].

3. Previous Work

Hudak et al., compared several software fault-tolerance techniques including ABFT [5]. The evaluation was based on error coverage for Launch Interceptor Program. The techniques were also ranked based on reliability and cost. Prata et al. compared ABFT to “Result checking” for matrix operations [6]. The criteria considered for comparison were error coverage, overhead and ease of use. In [2], detection only was compared to detection and correction in terms of error coverage, memory size and execution time.

The work presented here is unique in that it is designed to optimize the performance of an application executing in the presence of errors. We compare the performance of recovery by recomputing vs. ABFT correction considering *success ratio* and *completion time*.

4. Fault Injection

The fault model assumed is a bit flip in memory, cache, a register or in ALU during a computation. Fault injection is performed by making a random change in a randomly chosen data entry of a given matrix before or after computation. Faults are injected at a high level (C code). We inject faults only in data entries because that is sufficient for the purpose of this work since we are comparing the performance of different recovery schemes that use the same detection mechanism. Faults can happen before, during or after the computation. The error arrival process is assumed to be Poisson with different rates.

5. Simulation Results

Three ABFT applications: matrix multiplication, LU decomposition and matrix inversion were implemented in C. Matrices of floating point numbers were generated randomly to be used as inputs to the algorithms.

We used matrix sizes (10×10 – 50×50) and error rates of (10^{-9} – 10^{-3}) per entry per second. To avoid round off errors in floating point computations we used a range of 10^{-6} within which two numbers were considered equal.

We show only two samples of the results due to space limitations. Figure 1 shows a comparison of *success ratio* using recovery by recomputing versus ABFT correction in Matrix Multiplication. Det/rec label corresponds to recomputing and cor(d=i) corresponds to correcting $2\times\lfloor i/2 \rfloor$ errors. The ABFT correction is done using various code distance (d) values. Increasing d means higher correction capability. However, it also means that the correction scheme will take longer. This causes a higher possibility for errors hitting the computation during correction and these errors are mostly not correctable.

The figure shows that recomputing has a relatively high *success ratio*. Higher values for d (3 and 4) resulted in considerably low *success ratios*. ABFT correction with d = 1 gave the highest *success ratio*.

Figure 2 shows a comparison of the *completion time* using recovery by recomputing vs. ABFT correction in matrix inversion. ABFT correction with d = 1, had the smallest *completion time* at low error rates. As the error rate increased, recomputing had the smallest completion time. Higher code distances ($d \geq 2$) had high *completion times*.

6. Summary and Conclusions

ABFT was invented to provide fault tolerance with low performance overhead. So it is important to compare different ABFT schemes based on performance. We used *success ratio* and *completion time* as the performance metrics for comparison. It was shown by simulation that multiple error correction in ABFT is inefficient in terms of both completion time and success ratio.

Results showed that ABFT single error correction was the best for matrix multiplication in terms of success ratio and completion time. For LU decomposition, recomputing was better in completion time. In success ratio, recomputing was better at low error rates and correction was better at high error rates. For matrix inversion, correction was better in success ratio. In completion time, correction was better at low error rates and recomputing was better at high rates.

In conclusion, error rate had a direct impact in making on scheme favorable in terms of performance. Generally, correction was better at low error rates and recomputing was better at high error rates especially for computationally intensive algorithms.

Acknowledgement

This work was supported in part by the National Aeronautics and Space Administration and administered through the Jet Propulsion Laboratory, California Institute of Technology under Contract No. 1216777.

References

[1] K. Huang and J. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations", *IEEE Transactions on Computers*, Vol. C-33, No. 6, pp. 518-528, June 1984.
[2] R. K. Acree, Nasr Ullah, A. Karia, J. T. Rahmeh & J. A. Abraham, "An Object-Oriented Approach for Implementing Algorithm-Based Fault Tolerance", *12th Annual International Phoenix Computers and Communications Conference* , pp. 210-216, Mar. 93.
[3] Paul S. Dwyer, *Linear Computations*, John Wiley & Sons, 1951.
[4] J. Jou and J. Abraham, "Fault-Tolerant Matrix Arithmetic and Signal Processing on Highly Concurrent Computing Structures", *Proc. IEEE*, vol. 74, pp. 732-741, May 1986.
[5] J. Hudak, B. Suh, D. Siewiorek and Z. Segall, "Evaluation & Comparison of Fault-Tolerant Software Techniques", *IEEE Transactions on Reliability*, Vol. 42, No. 2, June 1993.
[6] P. Prata and J. Silva, "Algorithm Based Fault Tolerance Versus Result-Checking for Matrix Computations", *International Symposium on Fault Tolerant Computing FTCS-29*, pp. 4 – 11, Jun 99.

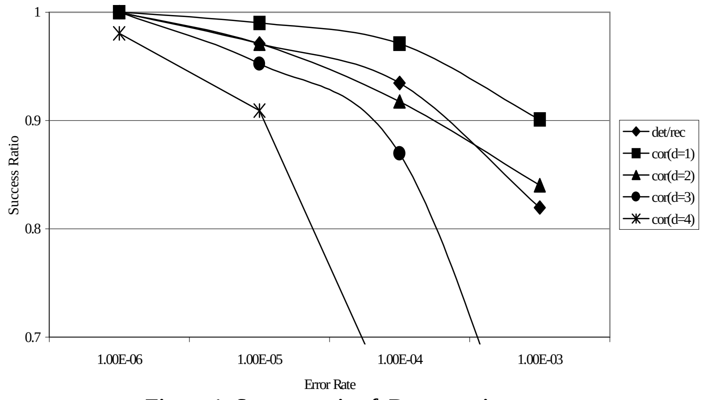

Figure 1: Success ratio of Recomputing vs. ABFT Correction with Different Code Distances

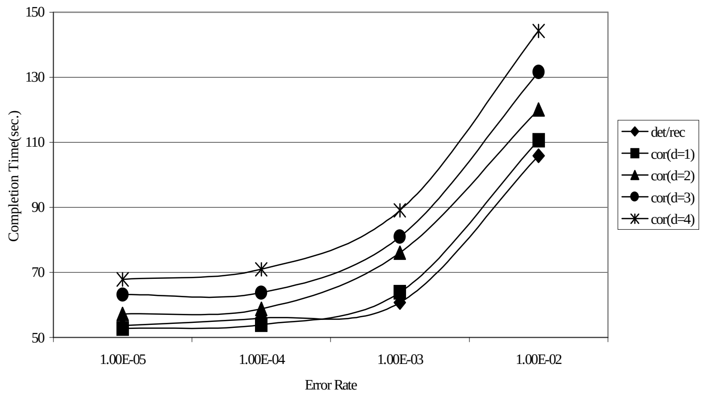

Figure 2: Completion time of Recomputing vs. ABFT Correction with Different Code Distances