ORNL REPORT

ORNL/TM-2013/282
Unlimited Release
Printed August 2013

Quantifying the Impact of Single Bit Flips on Floating Point Arithmetic

J. Elliott, F. Mueller, M. Stoyanov, C. Webster

Prepared by
Oak Ridge National Laboratory
One Bethel Valley Road, Oak Ridge, Tennessee 37831

ORNL/TM-2013/282

Computer Science and Mathematics Division

QUANTIFYING THE IMPACT OF SINGLE BIT FLIPS ON FLOATING POINT ARITHMETIC

J. Elliot [*] F. Mueller [†] M. Stoyanov [‡] C. G. Webster [§]

Date Published: August 2013

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6283
managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

[*]Computer Science Department, North Carolina State University, Raleigh, NC, (`jjellio3@ncsu.edu`)

[†]Computer Science Department, North Carolina State University, Raleigh, NC, (`mueller@cs.ncsu.edu`)

[‡]Computer Science and Mathematics Division, Oak Ridge National Laboratory, One Bethel Valley Road, P.O. Box 2008, MS-6367, Oak Ridge, TN 37831-6164 (`stoyanovmk@ornl.gov`).

[§]Computer Science and Mathematics Division, Oak Ridge National Laboratory, One Bethel Valley Road, P.O. Box 2008, MS-6164, Oak Ridge, TN 37831-6164 (`webstercg@ornl.gov`)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In high-end computing, the collective surface area, smaller fabrication sizes, and increasing density of components have led to an increase in the number of observed bit flips. If mechanisms are not in place to detect them, such flips produce silent errors, i.e. the code returns a result that deviates from the desired solution by more than the allowed tolerance and the discrepancy cannot be distinguished from the standard numerical error associated with the algorithm. These phenomena are believed to occur more frequently in DRAM, but logic gates, arithmetic units, and other circuits are also susceptible to bit flips. Previous work has focused on algorithmic techniques for detecting and correcting bit flips in specific data structures, however, they suffer from lack of generality and often times cannot be implemented in heterogeneous computing environment.

Our work takes a novel approach to this problem. We focus on quantifying the impact of a single bit flip on specific floating-point operations. We analyze the error induced by flipping specific bits in the most widely used IEEE floating-point representation in an architecture-agnostic manner, i.e., without requiring proprietary information such as bit flip rates and the vendor-specific circuit designs.

We initially study dot products of vectors and demonstrate that not all bit flips create a large error and, more importantly, expected value of the relative magnitude of the error is very sensitive on the bit pattern of the binary representation of the exponent, which strongly depends on scaling. Our results are derived analytically and then verified experimentally with Monte Carlo sampling of random vectors. Furthermore, we consider the natural resilience properties of solvers based on the fixed point iteration and we demonstrate how the resilience of the Jacobi method for linear equations can be significantly improved by rescaling the associated matrix.

ACKNOWLEDGEMENTS

The first two authors were supported by NSF grants 1058779, 0958311, 0937908 and DOE DE-AC05-00OR22725. The third and fourth authors were supported by the ASCR and DOE DE-AC05-00OR22725. The ORNL is operated by UT-Battelle, LLC, for the United States Department of Energy under Contract DE-AC05-00OR22725.

1 Introduction

Supercomputers have become an essential instrument to push the limits of complex simulations and large-scale data analysis for sciences, industry, and government. High-Performance Computing (HPC) systems have reached multi-petascale capabilities with exascale on the horizon. But recent work shows that at such scale faults are becoming the norm rather than the exception [24, 28]. Existing fault tolerance schemes (checkpoint/restart [9, 22, 32, 33]) are reaching their scalability limitations [18, 23, 25, 27], while scalable approaches (e.g., redundant execution [11]), may only become feasible at extreme scale and only via capacity computing (increasing job throughput) rather than capability computing (exploiting all resources for an application) [10].

More significant for this work, hardware protection for memory (ECC) detects and corrects the vast majority of bit flips due to radiation from space and decreasing fabrication sizes of semiconductors [28]. Recent work has shown that multi-bit upsets are rare events and chipkill functionality is extremely effective in reducing node failure rates due to DRAM errors [31]. However, processing cores remain largely unprotected [12, 17]. Thus, novel approaches for scalable resilience are required to address the challenge not only at hardware level, but also via the software.

Contributions: This work calls into question an assumption that the field of computational science has taken for granted for quite some time, namely the assumption that computer arithmetic is reliable. As systems continue to grow in size and density, and as fabrication sizes continue to shrink, silent faults in the hardware will present a challenge never before anticipated by users. The goal of this work is to understand the implications of faulty arithmetic and to do so in manner that is theoretically sound and experimentally reproducible. We seek to quantify the impact of a single bit flip on specific numerical methods. From this analysis, we take a first step in characterizing numerical methods by their resilience to silent data corruption (SDC) caused by bit flips.

We analyze the IEEE-754 binary representation and observe the effects of scaling on the relative error introduced by bit flips into the vector dot product. We verify our results via Monte Carlo sampling of random vectors and random bit flips. Furthermore, we explore the natural resilience properties of fixed point iterative solvers and those can be improved by proper scaling.

This document is structured into three major themes: 1) Fault characterization; 2) modeling and sampling the impact of a single bit flip in dot products; 3) analyzing the impact of a silent bit flip on the fixed point iterative algorithms; 4) give a numerical example of how the resilience of the Jacobi iterative method can be significantly improved by simple rescaling of the associated matrix.

2 Related Work

The interest in SDC isn't new and a number of studies have already been conducted. Most commonly, a black-box approach is assumed, where soft errors are injected into an existing code and the resulting error is observed. Recently, [14,15] analyzed the behavior of various Krylov methods and observed the variance in iteration count based on the data structure that experiences the bit flip. Shantharam et al. [29] analyzed how bit flips in a sparse matrix vector multiply (SpMV) impact the L^2 norm and observe the error as CG is run. Bronevetsky et al. [4, 30] analyzed several iterative methods documenting the impact of randomly injected bit flips into specific data structures in the algorithms and evaluated several detection/correction schemes in terms of overhead and accuracy. Malkowski et al. [21] analyzed SDC from the perspective of the L1 and L2 caches and proposed an eviction and prefetching scheme that minimizes the amount of time data sits in the unprotected cache. Hoemmen and Heroux proposed a fault tolerant GMRES algorithm based on the principles of flexible preconditioners and demonstrated that their method is resilient to soft errors [13]. Exemplifying the concept of black-box analysis of bit flips, [20] presents BIFIT for characterizing applications based on their vulnerability to bit flips.

Algorithm-based fault tolerance (ABFT) provides an approach to detect (and optionally correct) faults, which comes at the cost of increased memory consumption and reduced performance [8,16]. The ABFT work by Huang et al. [16] was proven by Anfinson et al. [2] to work for several matrix operations, and the checksum relationship in the input checksum matrices is preserved at the end of the computation. Consequently, by verifying this checksum relationship in the final computation results, errors can be detected at the end of the computation. Costs in terms of extra memory and computation required for ABFT may be amortized for dense linear algebra, and such overheads have been analyzed by many (e.g., [1, 3, 19]). The more subtle problem is that algorithms have to be manually redesigned for ABFT support taking numerical properties (e.g., invariants) into account. Recent work has looked at extending ABFT to additional matrix factorization algorithms [8] and as an alternative to traditional checkpoint/restart techniques for tolerating fail-stop failures [5–7].

3 Fault Characterization

We first establish clear definition of bit flip and hardware faults. According to Hoemmen's abbreviated taxonomy [13], our focus is best described as one on *transient silent faults*. We extend this taxonomy by introducing a classification called **silent** and present the following definitions:

(1) **Silent faults** are a subset of soft faults, meaning they do not cause immediate program interruption and are **not** detected. These silent faults may occur in hardware units that do not have any safeguards in place to detect bit flips, such as the Arithmetic Logic Unit or registers (and occasionally L1 caches, such as in BlueGene/L). We do not consider *not a number* (NaN) or *infinity* (Inf) to be silent faults because 1) NaN and Inf may be trapped using floating point exceptions, and, more importantly, 2) for the methods presented in this paper NaN or Inf will propagate to the solution where they are a clear indicator that something is incorrect.

(2) **Transient silent faults** are faults that do not persist in the data, i.e., they corrupt data but do persist in the output of the operation that used the data. For instance, a flip in an adder can be modeled as a corrupt input, but the corrupted input is never saved. Hence, the corruption only manifests itself in the output of the adder, which may be saved or used in another operation.

3.1 Single Bit Flips vs. Multiple Bit Flips

Bit flips are commonly thought to only occur extremely rarely during arithmetic operations inside of arithmetic and logic units (ALUs) and floating-point units (FPUs). This belief is corroborated by years of experience with ALUs/FPUs units on systems providing solutions that match analytic solutions. However, recent work indicates there may be higher bit flip rates than previously thought, not just in memory but also in ALUs/FPUs [12, 17].

In this study, we analyze a single bit flip on the input to some numerical methods, but this single flip on the input may be viewed as multiple flips on the output. Consider the multiplication of two integers, $9 \times 3 = 27$, and assume a bit is flipped, e.g., $9 \rightarrow 13$. This results in the following output:

$$\begin{aligned} 9 \times 3 = 27 &= 1001 \times 0011 = 011011, \\ 13 \times 3 = 39 &= 1101 \times 0011 = 100111. \end{aligned}$$

Note how a single flip on the input results in multiple incorrect bits in the output. Hence, even though we consider only the case of a single bit flip, most of our results trivially extend to multiple flips.

Table 1: When a bit flips, a number v is perturbed to $\tilde{v} = v + z$; the error z depending on the index i of the flipped bit and whether the original value was 0 or 1.

Original bit value	Mantissa bit $0 \le i \le 51$	Exponent bit $52 \le i \le 62$	Sign bit $i = 63$
0	$2^e 2^{i-52}$	$(2^{2^{i-52}} - 1)v$	$-2v$
1	$-2^e 2^{i-52}$	$(2^{-2^{i-52}} - 1)v$	$-2v$

4 IEEE 754 Specification

The IEEE 754 specification describes various floating point standards, from which we chose to analyze the 64-bit *double precision* specification called *Binary64* that is widely used in scientific computing today. Given 64 bits $\{b_i\}_{i=0}^{63}$, a *Binary64* number is represented as

$$v = (-1)^{b_{63}} 2^{e-1023} \left(1 + \sum_{i=0}^{51} b_i 2^{i-52} \right), \tag{4.1}$$

where $\{b_i\}_{i=0}^{51}$ are the bits of the mantissa, $e = \sum_{i=52}^{62} b_i 2^{i-52}$ is the exponent, and b_{63} is the sign bit. A visual representation of the format is given on Figure 1. Note that the exponent is stored using a bias of 1023. This bias may be exploited to enhance resiliency as demonstrated in Section 5.

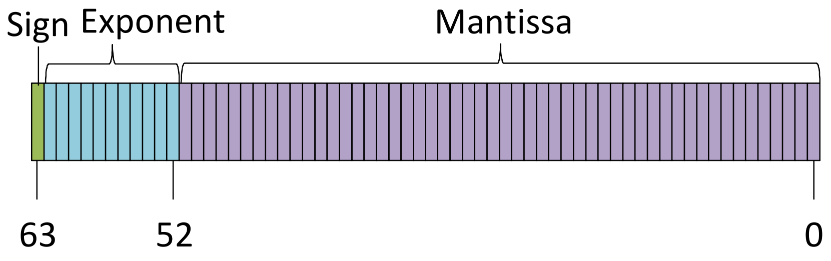


Figure 1: Representation of the Binary64 IEEE format.

Suppose a single bit flips, i.e. one of b_i changes from $0 \to 1$ or $1 \to 0$, then v is perturbed to a new value

$$\tilde{v} = v + z,$$

where z depends on the location and value of the flipped bit. We consider all possible cases in Table 4. We can bound the relative error for all cases as

$$\frac{|z|}{|v|} \le \begin{cases} 2^{i-52}, & 0 \le i \le 51, \\ (2^{2^{i-52}} - 1), & b_i = 0 \text{ and } 52 \le i \le 62, \\ (2^{-2^{i-52}} - 1), & b_i = 1 \text{ and } 52 \le i \le 62, \\ 2, & i = 63. \end{cases} \tag{4.2}$$

The relative error is largest when a zero bit in the exponent is flipped, in all other cases the relative error is bounded by $\frac{|z|}{|v|} \le 2$, therefore, zero-bits in the exponent are the most "volatile" bits. Without prior knowledge about the hardware, we can improve the resilience of an algorithm if we reduce the number of "volatile" bits in the double precision structures of the algorithm's implementation.

Table 2: Binary patterns in the exponent.

Base10	Exponent Bits $b_{62} \quad \dots \quad b_{52}$	Bias Relation	Effective Exp.
$\approx 10^{308}$	11111111110	$2^{2047-1023}$	2^{1024}
5	10000000001	$2^{1025-1023}$	2^{2}
2	10000000000	$2^{1024-1023}$	2^{1}
1	01111111111	$2^{1023-1023}$	2^{0}
.2	01111111100	$2^{1020-1023}$	2^{-3}
.5	01111111110	$2^{1022-1023}$	2^{-1}
$\approx 10^{-308}$	00000000000	2^{0-1023}	2^{-1023}

Table 2 presents several cases for the bit pattern in the exponent and the resulting scaling. The most volatile case is close to machine precision, while the most resilient operates at very large numbers. Usually, numerical code avoids working with numbers at the extremes of the double-precision representation range due to numerical stability. Away from the extremes, the most volatile representation is associated with numbers slightly bigger than 2, while numbers between 1 and 2 have the largest number of resilient bits. This observation is fundamental in assessing the effects of bit-flips and can also be used to increase the resilience of numerical algorithms.

5 Case Study: Vector Dot Products

5.1 Analytic Error Propagation: Deterministic Case

First we study the effect of single bit flips on the dot product of two N-dimensional vectors. We make this choice since many linear algebra operations can be decomposed into dot products, e.g. the matrix-vector product.

Let $\vec{u}, \vec{v} \in \mathbb{R}^N$ be two N-dimensional vectors. The dot product between u and v is defined as

$$\vec{u} \cdot \vec{v} = \sum_{j=1}^{N} u_j v_j,$$

where u_j and v_j are the components of $\vec{u}$ and $\vec{v}$. Suppose we encounter a bit flip in bit i and w.l.o.g. assume that the j-th element of $\vec{u}$ was perturbed to $u_j + z$, where z is the error described in the previous section. The relative error introduced by the bit-flip is

$$e = \frac{|\tilde{u} \cdot \vec{v} - \vec{u} \cdot \vec{v}|}{|\vec{u} \cdot \vec{v}|} = \frac{\frac{|z|}{|u_j|}|u_j v_j|}{|\vec{u} \cdot \vec{v}|} = \frac{|z|}{|u_j|} \frac{|u_j v_j|}{|\vec{u} \cdot \vec{v}|} = \frac{|z|}{|u_j|}|d_j|,$$

where $d_j = \frac{u_j v_j}{\vec{u} \cdot \vec{v}}$. Thus, the error is bound by two components, the relative error term described in 4.2 and the relative magnitude of $|u_j v_j| / |\vec{u} \cdot \vec{v}|$.

If $|\vec{u} \cdot \vec{v}| \ll |u_j v_j|$ then the error becomes effectively unbounded, however, this case will arise only when there is a significant cancellation in the components of the dot product sum and such cancellation is numerically unstable even if executed on reliable hardware. In the converse case $|u_j v_j| \ll |\vec{u} \cdot \vec{v}|$, the relative magnitude significantly dampens the effects of the hardware error. The scaling of $|\vec{u} \cdot \vec{v}|$ and $|u_j v_j|$ are problem specific and hence we want to focus our attention on the effect of $\frac{|z|}{|u_j|}$. We consider the balanced case, where the components $u_j v_j$ have similar sign and magnitude and hence $|d_j| \approx \frac{1}{N}$.

According to 4.2, when a bit flips from 1 to 0 or if the bit is not in the mantissa, then $\frac{|z|}{|u_j|} \leq 2$. Otherwise, the error has the form $(2^{2^{i-52}} - 1)$, thus if the i-th bit is flipped

$$e = \frac{|z|}{|u_j|}|d_j| \leq \begin{cases} (2^{2^{i-52}} - 1)/N, & b_i = 0 \text{ and } 52 \leq i \leq 62, \\ 2/N, & \text{otherwise}, \end{cases} \tag{5.1}$$

where $\{b_i\}_{i=0}^{63}$ are the bits that represent u_j.

5.2 Analytic Error Propagation: Probabilistic Case

Bit flips are a random phenomena, hence, we seek a probabilistic description of the error. We associate each bit b_i with the probability of flip in that bit p_i, so that $\sum_{i=0}^{63} p_i = 1$ [1]. Numerical

[1] We also assume that every double precision number is equally susceptible to a flip

solvers always generate an approximate solution to a problem, hence a bit flip will not have a significant effect unless the error introduced exceeds the numerical accuracy. In other words, we are interested in the probability of failure $P(e > \epsilon)$, i.e. the probability that e would exceed some pre-defined ϵ. Define B as the set of bit indexes that would result in a failure:

$$B = \{i : 0 \leq i \leq 63, \text{ and } e > \epsilon\}.$$

Then we have the probability of error

$$P(e > \epsilon) = \sum_{i \in B} p_i.$$

If p_i for different bits doesn't vary in a significant way (i.e. p_i are distributed uniformly), then the major factor affecting the probability of failure is the cardinality of B.

Suppose that ϵ and the vector size N are so that $2/N < \epsilon$, then according to (5.1) failure may occur only if the flipped bit is a zero in the exponent of the binary representation

$$B_{2/N<\epsilon} = \{i : 53 \leq i \leq 62 \text{ and } b_i = 0\}.$$

In that case, $P(e > \epsilon)$ is only affected by the bit pattern illustrated on Table 2.

On the other hand, if $2/N > \epsilon$, then every bit flip in the exponent would result in failure

$$B_{2/N>\epsilon} = \left\{i : i \leq 52, \text{ and } 2^{i-52}/N > \epsilon\right\} \cup \{53, 54, \cdots, 63\}.$$

Conclusion: For sufficiently large problem size N, the probability of a bit flip causing a catastrophic failure is affected only by the number of zero bits in the mantissa of the IEEE 754 representation.

5.3 Numerical Experiment

Next, we experimentally verify the analytic conclusion from the previous section. Suppose we are given two N-dimensional vectors of IEEE 754 double-precision numbers. We can systematically flip every one of the $2 \times 64 \times N$ bits and observe the resulting relative error in the dot product. We assume that every bit is equally likely to flip, then for a given ϵ we can compute the probability of failure

$$P(e > \epsilon) = \frac{number\ of\ bits\ resulting\ in\ error\ more\ than\ \epsilon}{2 \times 64 \times N}.$$

We want to observe the effects of the bit pattern on $P(e > \epsilon)$, however, the total number of possible patterns is prohibitively large. We use Monte Carlo sampling where we generate a series of random vectors using *C stdlib rand()* and compute $P(e > \epsilon)$ for each sample. Then we take the mean and standard deviation of $P(e > \epsilon)$ for each sample set. Each vector is scaled so that the elements have approximately the same magnitude and we consider a range for possible magnitudes each corresponding to a different bit pattern of the exponent.

We consider the range of magnitudes between 2^{-50} and 2^{50}, which corresponds to 101 bit patterns. For each power of 2, we consider vector lengths $N = 10^2, 10^4, 10^5, 10^6$. For each case we generate 10^6 Monte Carlo samples. We take $\epsilon = 10^{-4}$. The results of the experiment are visualized as four plots on Figure 5.3. As the vector size increases, we observe a decrease in the probability of failure. More importantly, we observe very strong dependence on the scaling, in particular in the transition region between 2^0 and 2^1. Whenever one or two of the vectors transitions to a bit pattern with significantly more 1-bits in the exponent, the probability of failure sharply increases.

Next we observe the expected value of the error as a function of the scaling. We generate random vectors in the same manner, however, we only consider equal scaling. As before, we consider $N = 10^2, 10^4, 10^5, 10^6$. We compare the results of Monte Carlo sampling with 10^6 samples to the analytically computed expected value. Whenever the expected value exceeds 1, we truncate the plot, since we assume that 100% error would be catastrophic for most applications. Figure 5.3 shows the results of the sampling. Again, we observe the sharp transitions between 2^0 to 2^1. Furthermore, we observe that for large vectors N, the expected value can be rather small for scaling 2^0 and slightly below.

Conclusion: The expected magnitude of the error introduced by bit flips depends on the bit pattern of the associated floating point data structures. If a numerical method is capable of correcting small errors introduced into the floating point operations, then the method will be able to withstand some bit flips (i.e. converge even if executed on unreliable hardware). Furthermore, problem scaling will have significant impact on the resilience of the method.

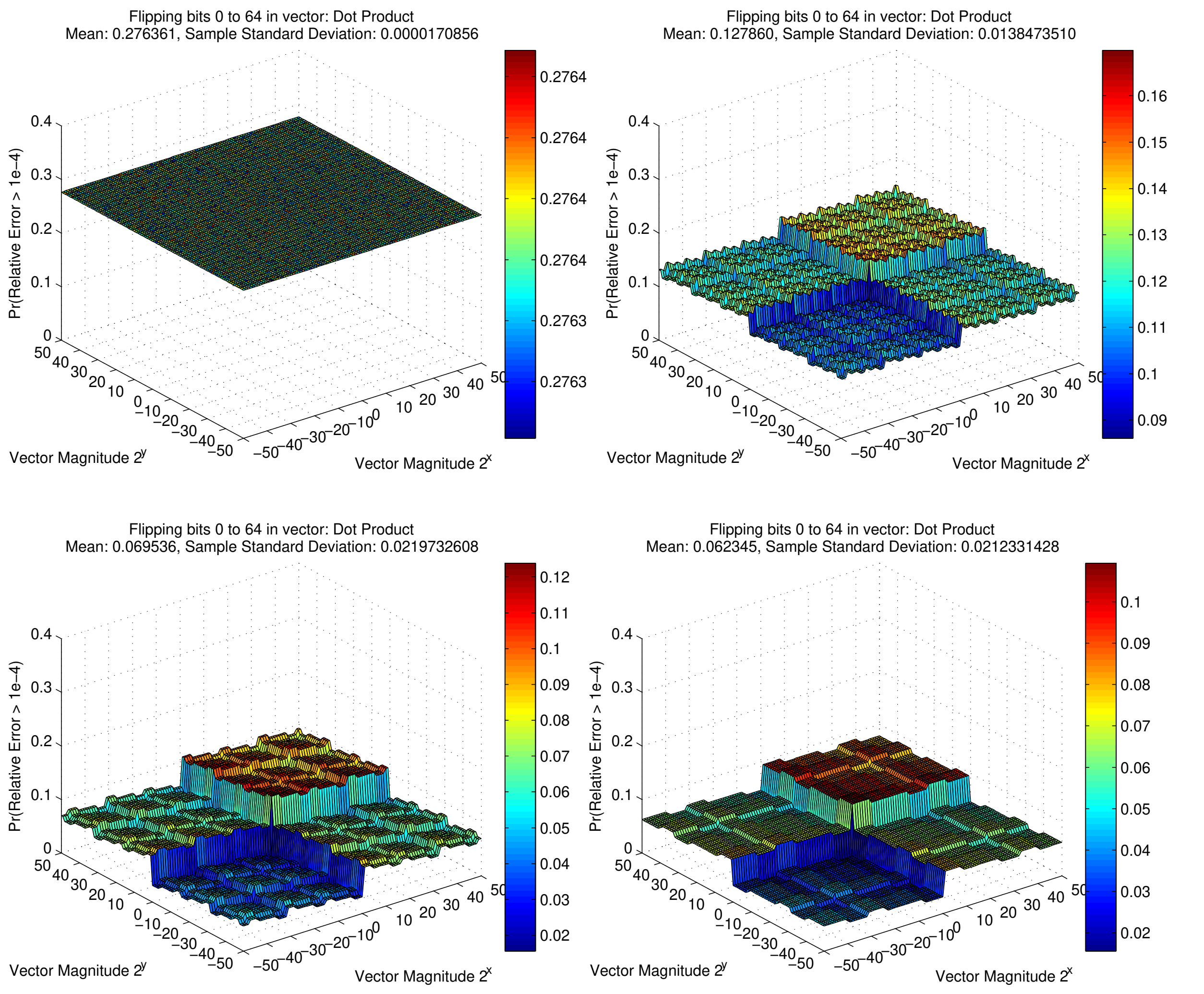


Figure 2: Probability of failure for different vector lengths as a function of the scaling of the two vectors. Top-right, $N = 100$, flipping any bit other than the least significant bits in the mantissa would cause failure, hence, $P(e > \epsilon)$ is highest with lowest variance and no sensitivity to scaling. The other three plots correspond to $N = 10^4$ (top right), $N = 10^5$ (bottom left) and 10^6 (bottom right), when the vector size is large enough, the number of zeros in the exponent become the determining factor for $P(e > \epsilon)$ and we see the four plateaus correspond to the sudden change of bit patterns as predicted by Table 2.

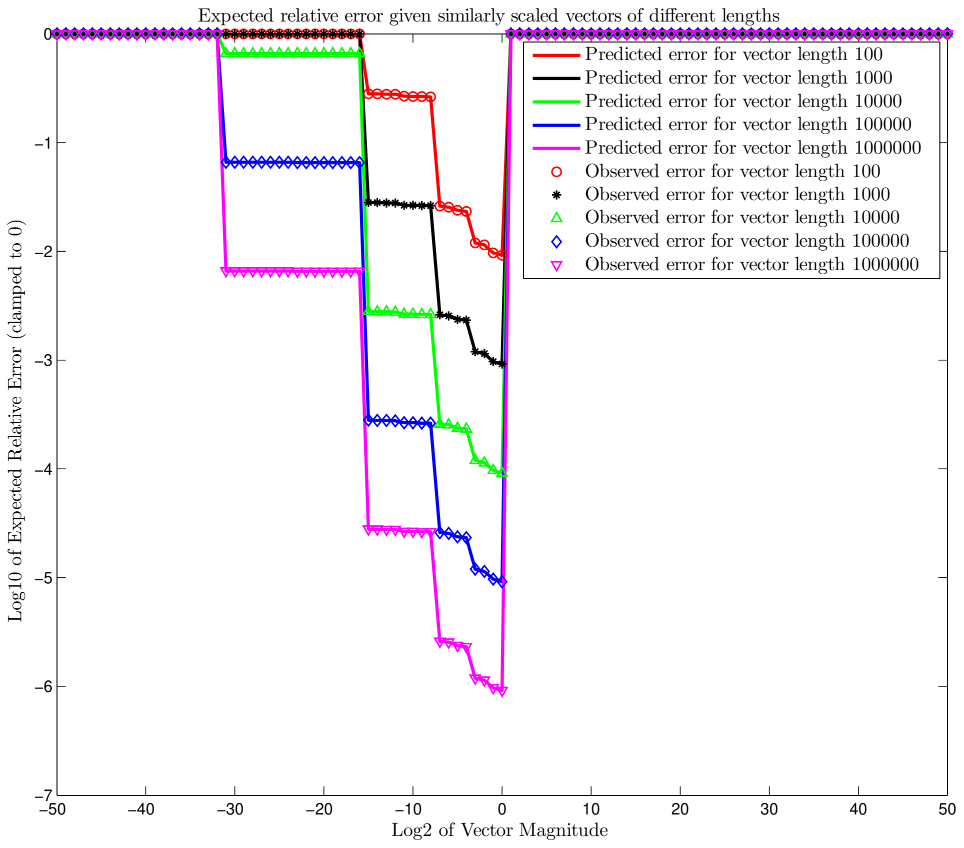


Figure 3: Expected value of the error e for different vector lengths as a function of the vector scaling, both vectors have the same scaling. To emphasize the dependence on the scaling, we truncate the expected error at 100%. We see decrease in the expected value as the vector size increases, but more importantly we see the sharp jump as scaling moves from 2^0 towards 2^1.

6 Fixed Point Iterative Methods

Iterative solvers are usually the methods of choice for large scale problems, e.g. Jacobi and Gauss-Siedel methods for linear equations or Newton method for non-linear equations. A sequence $\{v_k\}_{k=1}^{\infty} \subset \mathbb{R}^N$ of approximate solutions is created so that $v_k \rightarrow v^*$, where v^* is the exact solution to the linear or non-linear equation of interest. A special class of iterative methods are the fixed point methods.

Theorem 1 *Fixed Point Theorem*

Suppose $\Gamma \subset \mathbb{R}^N$ is complete in some norm $\|\cdot\|$ and let $\psi : \Gamma \rightarrow \Gamma$ be a contraction operator, i.e. there is a constant $c < 1$ so that

$$\|\psi(v) - \psi(w)\| \leq c\|v - w\|, \qquad \textit{for all } v, w \in \Gamma.$$

Then, there is a unique fixed point $v^* \in \Gamma$ so that $\psi(v^*) = v^*$. Furthermore, for any $v_0 \in \Gamma$ the sequence $\{v_k\}_{k=0}^{\infty}$ defined by $v_{k+1} = \psi(v_k)$ converges to v^* (i.e. $v_k \rightarrow v^*$) and the error $\|v_k - v^*\|$ is bounded by

$$\|v_k - v^*\| \leq c^k \|v_0 - v^*\|.$$

A proof of the theorem can be found in [26].

For a properly chosen operator ψ, the fixed point of the above theorem is the solution to a linear or non-linear equation of interest. Examples of such methods are Jacobi and Gauss-Seidel iterations for linear equations and the Newton method for non-linear problems. In practice, one seeks an initial guess v_0 that is close the fixed point of interest and the sequence $\{v_k\}_{k=0}^{\infty}$ is terminated at some K with $\|v_K - v^*\| \leq tol$.

Fixed point methods have natural resilience. Suppose that we encounter error at step k and the iterate x_k is perturbed to

$$\tilde{v}_k = v_k + \tilde{v},$$

so long as $\tilde{v}_k \in \Gamma$, the fixed point iteration will converge and the error at iteration step n ($n > k$) can be bound by

$$\|v_n - v^*\| \leq c^k \|v_0 - v^*\| + c^{n-k}\|\tilde{v}\|.$$

Therefore, fixed point methods can withstand a range of hardware faults by correcting the error with additional iterations. Furthermore, the number of additional iterations needed for convergence is strongly dependent on the magnitude of the added hardware error.

6.1 Example: Jacobi Method

One of the simplest examples of a fixed point method for solving linear equations is the Jacobi iteration. While the Jacobi method is more often used as a preconditioner rather than a solver, its simplicity makes it the perfect example for an algorithm that can benefit from resilience enhancing scaling.

Consider the system of linear equations described in matrix form as

$$Av = b, \tag{6.1}$$

where $A \in \mathbb{R}^{N\times N}$ and $b \in \mathbb{R}^N$ are given. If A is non-singular, then there exist a unique solution $v^* \in \mathbb{R}^N$ that satisfies (6.1).

Let $D = diag(A)$ (i.e. $R \in \mathbb{R}^{N\times N}$ is a diagonal matrix with the elements of A on the diagonal) and define $R = A - D$. The Jacobi operator is defined as

$$J(v) = D^{-1}(b - Rv),$$

and for a given initial guess v_0 we have the Jacobi iteration

$$v_{k+1} = J(v_k) = D^{-1}\left(b - Rv_k\right).$$

The solution v^* to equation (6.1) is a fixed point of $J(v)$ (i.e. $J(v^*) = v^*$). If $J(v)$ is a contraction in any norm then according to the Fixed Point Theorem the Jacobi method will converge. Denote by $\rho(D^{-1}R)$ the spectral radius of $D^{-1}R$, if $\rho(D^{-1}R) < 1$, then for any initial guess the Jacobi iteration will converge to the solution.

The most computationally expensive step of the Jacobi algorithm is computing the vector matrix product Rv_k, which consists of N dot products. We want to enhance the resilience of the Jacobi method by utilizing the relation between scaling and the magnitude of error introduced by bit flips.[2].

6.2 Numerical Example: Diffusion Equation

Let $\Omega = [0,1] \times [0,1] \subset \mathbb{R}^2$ and consider the partial differential equation

$$\begin{aligned} \frac{d}{dt}u(t,x,y) &= -\frac{\partial^2}{\partial x^2}u(t,x,y) - \frac{\partial^2}{\partial y^2}u(t,x,y), \qquad (x,y) \in \Omega, t > 0, \\ u(t,x,y)|_{\partial\Omega} &= 0, \\ u(0,x,y) &= xy(x-1)(y-1). \end{aligned}$$

We seek a numerical approximation to the solution $u(t,x,y)$. We discretize the problem in Ω using Finite Difference scheme with uniformly distributed nodes. Define

$$\{x_i\}_{i=1}^n, \quad x_i = \frac{i}{n+1}, \qquad \{y_j\}_{j=1}^n, \quad y_j = \frac{j}{n+1},$$

[2]In most large sale applications the matrix A is sparse. In order to reduce the memory footprint, the sparse data structure is often times represented in a compact form by a mixture of integers and floating point numbers (e.g. row compact form). The integers are susceptible to bit flips, however, discrete data structures require different resilience approach which is beyond the scope of this document.

and approximate

$$u(t, x_i, y_j) \approx u_{i,j}(t), \qquad u_{i,j}(0) = x_i y_j (x_i - 1)(y_j - 1).$$

We discretize the diffusion operator as

$$-\frac{\partial^2}{\partial x^2} u_{i,j}(t) - \frac{\partial^2}{\partial y^2} u_{i,j}(t) \approx \frac{u_{i-1,j}(t) - 2u_{i,j}(t) + u_{i+1,j}(t)}{\Delta x^2} + \frac{u_{i,j-1}(t) - 2u_{i,j}(t) + u_{i,j+1}(t)}{\Delta y^2},$$

where $\Delta x = \Delta y = \frac{1}{n+1}$ and $u_{0,j}(t) = u_{n+1,j}(t) = u_{i,0}(t) = u_{i,n+1}(t) = 0$. The spacial discretization results in n^2 ordinary differential equations that can be written in a matrix form

$$\dot{v}(t) = Lv(t),$$

where the k-th component of $v(t)$ is associated with $u_{i,j}(t)$ by $v_{in-n+j}(t) = u_{i,j}(t)$ and L is the matrix representation of the discretized operator.

We evolve the system in time using backward Euler method. We select a time step Δt and approximate

$$v(t + \Delta t) \approx (I - \Delta t L)^{-1} v(t).$$

At each time step we need to solve a system of linear equations and we use the Jacobi method.

For the purpose of this study, we only consider the linear system associated with the initial step for $t = 0$ that approximates $v(\Delta t)$. We take $n = 50$ which results in $2,500$ degrees of freedom and we use $\Delta t = 2^{-12}$.

6.3 Numerical Example: Scaling Comparison

We split the matrix L as

$$L = -\frac{4}{\Delta x^2} I + \frac{1}{\Delta x^2} R = -\frac{4}{\Delta x^2} I + S,$$

where R is a sparse matrix with ones corresponding to the off-diagonal non-zeros of L, $S = \frac{1}{\Delta x^2} R$ and I is the identity matrix. We are interested in solving a linear system of equations of the form

$$(I - \Delta t L)\, v = b,$$

where b is the vector of initial conditions for the diffusion problem and the solution v^* is the first step of the Euler integration scheme. We apply the Jacobi method to the above system and we present two alternative implementations of the solver.

Algorithm 1 *Regular-Jacobi-Iteration*

```
Given $\Delta t$, $S$ and $b$, let $v_0 = b$ and $k = 0$
repeat
    $k \leftarrow k + 1$
    $v_k \leftarrow S v_{k-1}$
    $v_k \leftarrow \frac{1}{\Delta t} b + v_k$
    $v_k \leftarrow \frac{\Delta t \Delta x^2}{4\Delta t - \Delta x^2} v_k$
    $e_k \leftarrow \|v_k - v_{k-1}\|_2$
until $e_k < 10^{-10}$
```

Algorithm 2 *Scaled-Jacobi-Iteration*

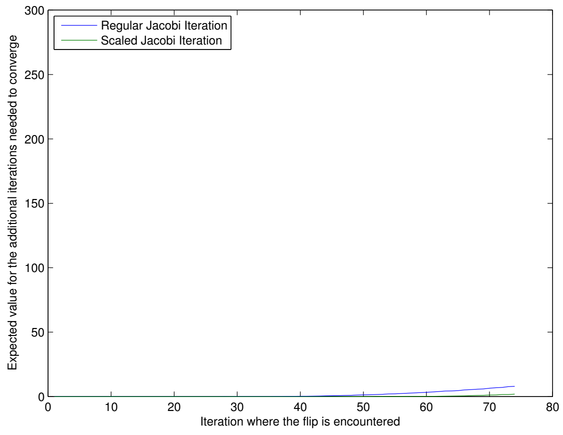
```
Given $\Delta t$, $S$ and $b$, let $v_0 = b$ and $k = 0$
repeat
    $k \leftarrow k + 1$
    $v_k \leftarrow R v_{k-1}$
    $v_k \leftarrow \frac{1}{\Delta x^2 \Delta t} b + v_k$
    $v_k \leftarrow \frac{\Delta t \Delta x^2}{4\Delta t \Delta x^2 - 1} v_k$
    $e_k \leftarrow \|v_k - v_{k-1}\|_2$
until $e_k < 10^{-10}$
```

The main difference between the two algorithms is that one uses R, while the other one works with S. All entries of R are equal to 1, which according to Table 2 has binary representation with very few zeros in the exponent and hence a bit flip in the entries of R will result in error with smaller relative magnitude. On the other hand, the entries of S have bit patterns that will result in larger bit-flip error (in expectation).

If executed without errors, both algorithms 1 and 2 converge to the solution in 74 iterations. We introduce random bit flips in the floating point numbers representing R and S and we observe the number of additional iterations that each algorithm would need to converge. In some cases, the error introduced by the bit-flip is too large and it requires a prohibitive number of additional iterations, we limit the total number of steps to $10,000$ to avoid such stagnation. We consider the additional number of iterations needed to converge as a function of the iteration on which the bit-flip took place. Given an iteration for the bit flip, we randomly select one of the numbers in R or S and we change one of its bits. For each of the 74 iterations, we take $1,000,000$ realizations of the possible bit flips and we take the average for the number of additional iterations needed to converge.

We show the result from our experiment on Figure 6.3. The expected number of additional iterations is plotted vs the iteration where the bit flip was introduced. For a bit flip in the mantissa (left plot), we observe that the natural resilience properties of the Jacobi algorithm can converge to a solution with very few additional iterations. Scaling changed the maximum number of iterations from 8 to 2, however, this is a small improvement on the scale of 74 total iterations. The biggest difference between the regular and scaled algorithms was observed when the bits were flipped in the exponent. In the best case, the regular Jacobi algorithm requires on average almost 200 additional iterations to converge, while the scaled algorithm is unaffected by a bit-flip in the first 30 iterations. Even in the worst case, the scaled version of the algorithm needs only 50 additional iterations vs 250 for the regular one.

In this example, all the entries of R were 1, which is the most resilient number according to Table 2. For other problems, the benefit from the scaling will be less significant. However, scaling as many numbers as possible to be one (or slightly less than one), will have a significant impact in proving the resilience of all fixed point methods that rely on vector (or even scalar) dot product.



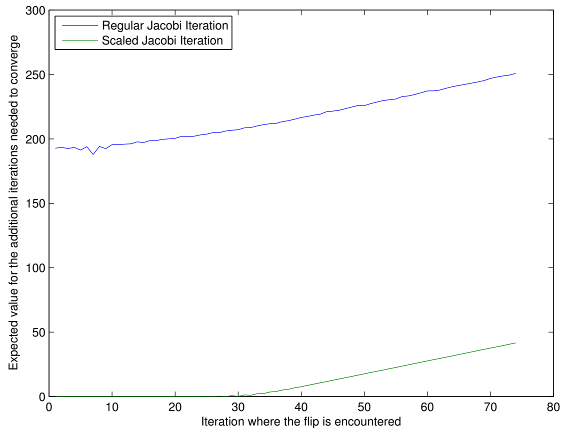


Figure 4: Expected number of additional iterations needed for convergence as a function of the iteration where the hardware fault was encountered. Considering bit flips in the mantissa (left), we see that the natural resilience capabilities of the Jacobi iteration can correct the error with only a few iterations (< 8) and scaling has overall small effect. However, when the bits of the exponent flip (right), then the standard Jacobi iteration requires a huge number of additional iterations to converge (≈ 200), while the scaled version performs much better. For the first 30 iteration, the Scaled Jacobi iteration will not be affected by the bit flip regardless whether it happened in the mantissa or the exponent. In the worst case, Scaled Jacobi took on average only ≈ 50 additional iterations.

7 Conclusions

This work contributes an analysis of how a silent bit flip in floating point arithmetic impacts the elementary linear algebra constructs. We particularly considered the vector dot product and the impact of the bit pattern of the IEEE-754 floating point representation on the relative magnitude of the error introduced by a bit flip. We demonstrated analytically and experimentally that scaling (and the associated bit patterns of the mantissa) has profound effect on the error introduced.

In addition, we also considered a common class of iterative solvers based on a fixed point contraction iteration. Those methods possess natural resilience properties that can be further enhanced by changing the problem scaling. In particular, we considered diffusion equation discretized with finite difference method in space and backward Euler method in time. The implicit time stepping was solved with Jacobi iterative method and we demonstrated how the scaling of the matrix entries can have a significant impact on the resilience of the solver. When the entries of the matrix were scaled to 1 and a bit was randomly flipped, the Jacobi method required much fewer additional iteration to converge to the desired solution.

REFERENCES

[1] A. Al-Yamani, N. Oh, and E. J. McCluskey, *Performance evaluation of checksum-based abft*, in Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2001), Oct. 2001, pp. 461–466. 3

[2] C. J. Anfinson and F. T. Luk, *Linear algebraic model of algorithm-based fault tolerance*, IEEE Transactions on Computers, 37 (1988), pp. 1599–1604. 3

[3] P. Banerjee, J. T. Rahmeh, C. Stunkel, V. S. Nair, K. Roy, V. Balasubramanian, and J. A. Abraham, *Algorithm-based fault tolerance on a hypercube multiprocessor*, Computers, IEEE Transactions on, 39 (1990), pp. 1132–1145. 3

[4] G. Bronevetsky and B. de Supinski, *Soft error vulnerability of iterative linear algebra methods*, in International Conference on Supercomputing, 2008, pp. 155–164. 3

[5] Z. Chen, *Extending algorithm-based fault tolerance to tolerate fail-stop failures in high performance distributed environments*, in International Parallel and Distributed Processing Symposium, Apr. 2008. 3

[6] ——, *Algorithm-based recovery for iterative methods without checkpointing*, in Symposium on High-Performance Parallel and Distributed Computing, June 2011, pp. 73–84. 3

[7] T. Davies, C. Karlsson, H. Liu, C. Ding, and Z. Chen, *High performance linpack benchmark: A fault tolerant implementation without checkpointing*, in International Conference on Supercomputing, May 2011, pp. 162–171. 3

[8] P. Du, A. Bouteiller, G. Bosilca, T. Herault, and J. Dongarra, *Algorithm-based fault tolerance for dense matrix factorizations*, SIGPLAN Not., 47 (2012), pp. 225–234. 3

[9] J. Duell, *The design and implementation of berkeley lab's linux checkpoint/restart*, tr, Lawrence Berkeley National Laboratory, 2000. 2

[10] J. Elliot, K. Kharbas, D. Fiala, F. Mueller, C. Engelmann, and K. Ferreira, *Combining partial redundancy and checkpointing for HPC*, in International Conference on Distributed Computing Systems, 2012. 2

[11] K. Ferreira, J. Stearley, J. H. L. III, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. Bridges, and D. Arnold, *Evaluating the viability of process replication reliability for exascale systems*, in Supercomputing, nov 2011. 2

[12] A. Geist, *What is the monster in the closet?* Invited Talk at Workshop on Architectures I: Exascale and Beyond: Gaps in Research, Gaps in our Thinking, Aug. 2011. 2, 4

[13] M. Hoemmen and M. A. Heroux, *Fault-tolerant iterative methods via selective reliability*. http://www.sandia.gov/ maherou/docs/FTGMRES.pdf. 3, 4

[14] V. Howle and P. Hough, *The effects of soft errors on krylov methods*. Invited Talk. SIAM Parallel Processing., Feb. 2012. 3

[15] V. Howle, P. Hough, M. Heroux, and E. Durant, *Soft errors in linear solvers as integrated components of a simulation*. Invited Talk, Apr. 2010. 3

[16] K.-H. Huang and J. A. Abraham, *Algorithm-based fault tolerance for matrix operations*, IEEE Transactions on Computers, C-33 (1984), pp. 518–528. 3

[17] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, *Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design*, in Architectural Support for Programming Languages and Operating Systems, 2012, pp. 111–122. 2, 4

[18] T. Z. Islam, K. Mohror, S. Bagchi, A. Moody, B. R. de Supinski, and R. Eigenmann, *Mcrengine - a scalable checkpointing system using data-aware aggregation and compression*, in Supercomputing, Nov. 2012. 2

[19] Y. Kim, J. S. Plank, and J. J. Dongarra, *Fault tolerant matrix operations using checksum and reverse computation*, in Symposium on the Frontiers of Massively Parallel Computing, Oct. 1996, pp. 70–77. 3

[20] D. Li, J. Vetter, and W. Yu, *Classifying soft error vulnerabilities in extreme-scale scientific applications using a binary instrumentation tool*, in Supercomputing, Nov. 2012. 3

[21] K. Malkowski, P. Raghavan, and M. Kandemir, *Analyzing the soft error resilience of linear solvers on multicore multiprocessors*, in International Symposium on Parallel Distributed Processing, Apr. 2010, pp. 1 –12. 3

[22] A. Moody, G. Bronevetsky, K. Mohror, and B. de Supinski, *Design, modeling, and evaluation of a scalable multi-level checkpointing system*, in Supercomputing, Nov. 2010. 2

[23] I. Philp, *Software failures and the road to a petaflop machine*, in Workshop on High Performance Computing Reliability Issues, IEEE Computer Society, 2005. 2

[24] E. Pinheiro, W.-D. Weber, and L. A. Barroso, *Failure trends in a large disk drive population*, in USENIX Conference on File and Storage Technologies, 2007. 2

[25] R. Riesen, K. Ferreira, D. D. Silva, P. Lemarinier, D. Arnold, and P. G. Bridges, *Alleviating scalability issues of checkpointing protocols*, in Supercomputing, Nov. 2012. 2

[26] W. Rudin, *Principles of mathematical analysis*, vol. 3, McGraw-Hill New York, 1964. 12

[27] K. Sato, A. Moody, K. Mohror, T. Gamblin, B. R. de Supinski, N. Maruyama, and S. Matsuoka, *Design and modeling of a non-blocking checkpointing system*, in Supercomputing, Nov. 2012. 2

[28] B. Schroeder, E. Pinheiro, and W.-D. Weber, *Dram errors in the wild: a large-scale field study*, in SIGMETRICS Conference on Measurement and Modeling ofComputer Systems, 2009, pp. 193–204. 2

[29] M. Shantharam, S. Srinivasmurthy, and P. Raghavan, *Characterizing the impact of soft errors on iterative methods in scientific computing*, in International Conference on Supercomputing, 2011, pp. 152–161. 3

[30] J. Sloan, R. Kumar, G. Bronevetsky, and T. Kolev, *Algorithmic approaches to low overhead fault detection for sparse linear algebra*, Dependable Systems and Networks, (2012). 3

[31] V. Sridharan and D. Liberty, *A study of dram failures in the field*, in Supercomputing, Nov. 2012. 2

[32] C. Wang, F. Mueller, C. Engelmann, and S. Scott, *A job pause service under LAM/MPI+BLCR for transparent fault tolerance*, in International Parallel and Distributed Processing Symposium, Apr. 2007. 2

[33] ——, *Proactive process-level live migration in hpc environments*, in Supercomputing, 2008. 2