# Evaluating the Resiliency of Posits for Scientific Computing

Benjamin Schlueter
bschlue@clemson.edu
Holcombe Department of Electrical and Computer Engineering - Clemson University
Clemson, SC, USA
bschlue@clemson.edu

Alexandra Poulos
alpoulo@clemson.edu
Holcombe Department of Electrical and Computer Engineering - Clemson University
Clemson, SC, USA
alpoulo@clemson.edu

Jon Calhoun
jonccal@clemson.edu
Holcombe Department of Electrical and Computer Engineering - Clemson University
Clemson, SC, USA
jonccal@clemson.edu

## ABSTRACT

IEEE-754 is the de-facto standard for the implementation of floating-point number systems in hardware. With the rise of machine-learning and mixed-precision computation, applications tolerate small inaccuracies in the data by using lower bit-width floating-point values to improve computational speed and memory bandwidth, but still rely on some variant of IEEE-754. Posits have been proposed as a drop-in replacement to the IEEE-754 floating-point standard. Recent work has suggested that posits can offer greater numerical accuracy and reproducibility than IEEE-754-compliant floating point numbers at a comparable architectural cost. There have been several studies which consider the use of posits and other floating-point implementations in hardware and software, but there is limited work examining this new number system from a reliability perspective.

In this paper, we evaluate the resiliency of posits to inform hardware design for fault-tolerant systems. Using real-world scientific datasets, we conduct an extensive bit-flip fault injection campaign to explore the tradeoff between accuracy and sensitivity to silent data corruption (SDC) between posits and IEEE-754. Our analysis breaks down the impact of a bit flip on the various fields within both floating-point standards. Results show the presence of the regime reduces the number of bits that cause catastrophic error in posits compared to IEEE floats. This leads to overall less error in upper bit positions of posits, except in certain edge cases. We found flips in the sign bit usually cause more error in posits due to the effect sign has on posit magnitude, unlike in IEEE floats. We also found that unlike in IEEE floats, the posit exponent does not produce significant error due to its small, static two bit size. Finally, error in the posit fraction was found to be similar to that of IEEE floats.

## CCS CONCEPTS

• **Software and its engineering** → **Software fault tolerance**; • **Mathematics of computing**; • **Hardware** → *Fault tolerance*;

## KEYWORDS

Posit, IEEE-754, floating-point, reliability, soft error, accuracy



*SC-W 2023, November 12–17, 2023, Denver, CO, USA*

ACM ISBN 979-8-4007-0785-8/23/11.
https://doi.org/10.1145/3624062.3624116

**ACM Reference Format:**
Benjamin Schlueter, Alexandra Poulos, and Jon Calhoun. 2023. Evaluating the Resiliency of Posits for Scientific Computing. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3624062.3624116

## 1 INTRODUCTION

Floating-point arithmetic is indispensable to many areas of computing, such as computational science and engineering, as well as machine learning. Central to the utility of floating-point arithmetic on computing systems is being able to represent a floating-point number. In computing systems, continuous real-numbers must be represented using a finite number of bits. Thus, all floating-point representations are discrete approximations of the numbers they represent. Numbers not represented exactly in the floating-point format are rounded to a representable number. The discrepancy between the true real-number and the representable one yields an error. Controlling this error in the representation and during arithmetic is central to the utility of a floating-point number system.

IEEE-754 is the first standard for floating-point arithmetic, and since its adoption in 1985 has become the standard across all computing systems [3]. An IEEE-754 number is composed of three bit fields (sign, exponent, and fraction) whose size is dependent on the bit-width of the data-type. Prior to its adoption, multiple diverse methods existed to format, compute, and round the floating-point numbers. This discrepancy led to numerical issues and correctness issues when codes were ported across systems [1, 4]. Wide adoption of this standard leads to reproducible performance, which enables compilers to conduct higher levels of optimization [9, 40].

Training deep learning (DL) models is very computationally intense. To lower the computation cost, researchers discovered that DL training can be done with lower-precision arithmetic [36]. Using lower-precision arithmetic has two key performance advantages. The first is that lower precision floating-point computation is faster than higher precision. On an NVIDIA H100 SXM GPU, the theoretical peak performance for the FP64 tensor cores is 67 TFLOPS. FP32, FP16 and F8 tensor cores yield 14.8×, 29.5×, and 59.1× more TFLOPS, respectively [2]. The second is that effective memory bandwidth (array elements moved per unit time) improves proportionally as the bit-width decreases. To improve the accuracy at lower precision, modifying the number of bits allocated to the exponent and fraction has lead to new IEEE-754 data-types such as `bfloat` [6, 39]. The structure of IEEE-754 floating-point numbers often fail to offer the precision and reproducibility needed for some computations.

Despite the continuous improvements and adaption of the IEEE-754 standard, it still has limitations. One well-known issues with IEEE-754 is that it is vulnerable to data corruption due to transient soft errors that corrupt bits at random in registers, data paths, and memory [10, 21, 42]. A single bit-flip in the exponent of an IEEE-754 number often yields an error that is orders-of-magnitude larger than the original number prior to corruption. Accuracy issues in the presence of soft errors have led to the development of fault-tolerant techniques at both the hardware and software level to ensure the correctness of applications [23, 28, 29, 31].

Posits [27] have recently been proposed as a drop-in replacement to the IEEE-754 floating-point standard. Posits possess a bit field called the regime, whose size scales based on the magnitude. This allows for greater fractional accuracy for numbers close to one. Recent work has suggested that posits can offer greater numerical accuracy and reproducibility than IEEE-754-compliant floating point numbers at a comparable architectural cost [41]. In addition, there have been several studies which consider the use of posits and see application runtime and accuracy improvements [16, 17]. As researchers are currently exploring the advantages and trade-offs of posits, they must be evaluated based on their reliability in the presence of soft errors. Prior work [8] conducts an initial study on posit resilience, but primarily focuses on the impact posit bit flips have in machine learning applications instead of analyzing error caused by flipping individual bits.

In particular, this paper makes the following contributions:

- Studies the resiliency and performance of the posit datatype in fault prone environments to inform hardware design for future fault prone systems
- Observes the behavior of posits when injecting faults and how it compares to standardized data types
- Provides insights on specific posit bit positions that are vulnerable to significant error when flipped
- Shows posits have increased resilience compared to IEEE floats in most cases

The rest of this paper is outlined as follows. In Section 2, we present related work. In Section 3 we provide a background on the internal structure of IEEE-754 floating-point and posit numbers. We describe our fault model and fault injection methodology in Section 4. We analyze in detail our fault injection campaign's results in Section 5. Finally, in Section 6, we conclude this paper.

## 2 RELATED WORK

In order to do fault injection studies on HPC applications, several fault injection frameworks have been developed to inject bit-flip errors in hardware [22] and software [11, 33].

Elliot et al. [21] presents a detailed study of the impact of bit position on the deviation in floating-point accuracy of IEEE-754. This work derives formulas to compute the deviation in a floating-point value based on the bit corrupt. Moreover, it explores how vector length combined with a corrupted element changes the accuracy of HPC computations. Other works [12, 32] explore how bit-flips in floating-point values impact the accuracy of HPC applications.

To protect applications from the negative impacts of bit-flip errors, prior works have leveraged checkpoint-restart [37], error correction codes [24, 35], redundancy [23], and forward recovery [7] as part of their protection scheme.

Recently, Alouani et al. explores posit resilience in machine learning related applications [8]. This work conducts a bit flip fault injection campaign on posits and measures the mean relative error distance (MRED), which provides a general overview of error caused by bit flips. This work does not go in depth regarding posit error in individual bit positions caused by flips. Instead, fault injections are applied in different machine learning applications to analyze their performance.

## 3 BACKGROUND

### 3.1 IEEE-754 Floating-point Standard

An IEEE-754 floating-point number, typically containing 32 or 64 bits, has three fields: a sign bit $s$, an unsigned integer exponent $e$, and a fraction field also known as the *mantissa* or *significand*. It also has an implicit *bias*, which is determined by the number of exponent bits $E$ as $2^{E-1}$. A floating-point number is numerically interpreted as

$$(-1)^s \times 2^{(exponent-bias)} \times \left(1 + \sum_{i=1}^{M} b_{M-i} 2^{-i}\right),$$

where $b \in \{0, 1\}$ and $M$ is the number of bits in the fraction. Figure 1 shows an example of a 32-bit float.

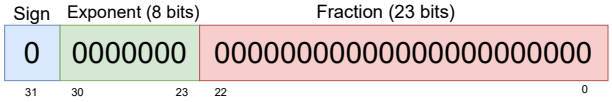


**Figure 1: 32 bit IEEE-754 Standard Structure**

IEEE-754 has specific representations for special values such as infinity, and NaN. For infinity, the exponent takes a value of 255, and the fraction is 0. The sign bit still applies for infinity, and the float will be positive infinity if the sign is 0 and negative infinity if the sign is 1. An example of the infinity representation can be seen below in Figure 2. The NaN representation also has an exponent of all 1's, but the fraction will be non-zero number [5].

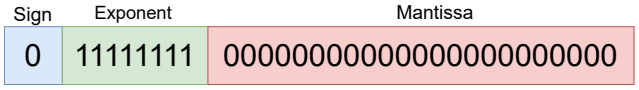


**Figure 2: IEEE-754 Representation of Positive Infinity**

Previous works show the IEEE-754 standard is vulnerable to bit flips [10, 21]. The impact of the corruption depends on the location where the bit-flip occurs. Bit-flips in the fraction have the smallest impact on the floating-point value. However, bit-flips in the most significant bits could have noticeable impact on the magnitude. A bit-flip in the exponent has the most significant impact on the induced error, where each bit-flip either multiplies or divides the value by a power of 2. This implies the impact of a bit-flip grows dramatically as the bit position where the bit-flip occurs increases.

The magnitude of the error caused by flipping the sign bit is always twice the original value, since it is equivalent to multiplying the value by -1 as shown below.

$$err_{abs} = |orig - faulty| = |orig - (-orig)| = 2orig$$

To better understand the impact of a bit-flip in each bit-position, Figure 3 shows the relative error for bit-flips in a 32-bit IEEE representation of 186.25.

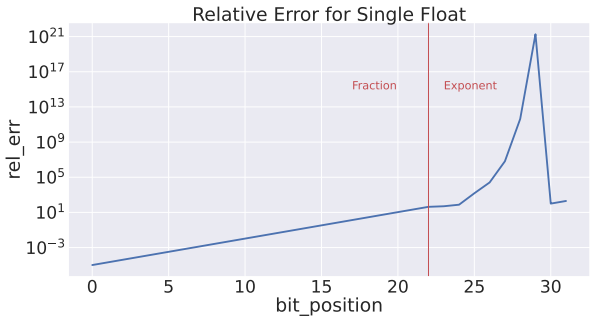


**Figure 3: Relative error with bit-flips in the representation of 186.25 in 32-bit IEEE-754.**

## 3.2 Posits

To improve the accuracy of floating-point numbers, posits, unlike IEEE-754, allow the field bit-widths inside the posit to vary in size even when the total number of bits for the posit remains the same. This dynamic nature enables the posit to more accurately represent the floating-point number in certain cases. Figure 4 shows the structure of the fields within the posit type. A posit of size $n$ consists of a sign bit $s$, one or more (up to $n-1$) regime bits, an optional unsigned integer exponent $e$, and an optional *significand*.

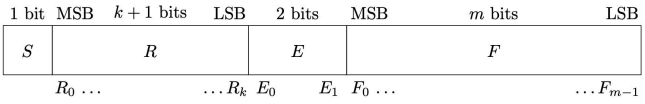


**Figure 4: A generic posit number [26].**

In Figure 4 [26], $S$ represents the sign bit which takes on a value of 0 or 1 and determines the sign of the posit, similar to IEEE floats. However, it is important to note that flipping the sign bit alone will not negate the posit. Negation of a posit requires the two's complement to be taken. $R$ is the regime field, which consists of $k$ identical bits starting at $R_0$, followed by a bit of opposite sign ($R_k$) that terminates the regime field. The value of $k$ depends on the magnitude of the posit using the following relation in Equation 1. In cases 1 and 3, the value of bits $R_0$ through $R_{k-1}$ will be 1 and $R_k$ will be 0. For cases 2 and 4, the regime bits will have the opposite states.

$$k = \begin{cases} p > 1 & \lfloor \log_{16} p \rfloor + 1 \\ 0 < p < 1 & \lceil \log_{16} p^{-1} \rceil \\ -1 < p < 0 & \lfloor \log_{16} p^{-1} \rfloor + 1 \\ p < -1 & \lceil \log_{16} -p \rceil \end{cases} \quad (1)$$

$E$ represents the exponent field, which is statically sized at two bits. Due to the impact the regime bits have on the magnitude, fewer exponent bits are needed than in IEEE-754 floats. $F$ represents the fraction, which has a maximum length of $n-5$ bits, but due to the dynamic size of the regime, some of these bits may be truncated. The fraction always begins with $F_0$, but any bits that extend beyond the LSB are truncated and do not affect the magnitude of the posit [26]. It is possible to have a posit with no fraction if the regime is large enough.

Equation 2 from the posit standard [26] shows the numerical interpretation of a posit (p).

$$p = ((1-3s)+f) \times 2^{(1-2s)\times(4r+e+s)} \quad (2)$$

The variable $s$ represents the sign, which takes on a value of 0 or 1. The value $r$ representing the regime is $r = -k$ if the value of $R_k$ is 1 and $r = k-1$ if $R_k$ is 0. The value $e$ is a 2 bit unsigned integer representation of the exponent bits. The fraction representation $f$ is an unsigned integer representation of the $m$ bits in the fraction field. The value of $f$ is determined by the following relation where $i$ represents a bit position and $F$ is the length of the fraction [26].

$$f = 2^{-m} \sum_{i=0}^{m-1} f_i 2^i \quad (3)$$

Figure 5 shows the numerical value of a posit being assembled from its bits.

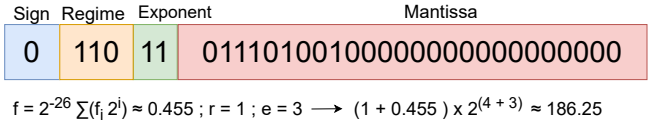


**Figure 5: Posit Numerical Value Assembly from Binary**

To improve accuracy, posits dedicate more bits to the fraction when magnitudes approach one, meaning more precision. As shown in the equation for regime size, Equation 1, a number with an absolute value close to one has a smaller regime and therefore more fraction bits, which increases precision. This is shown in figures 6 and 7.

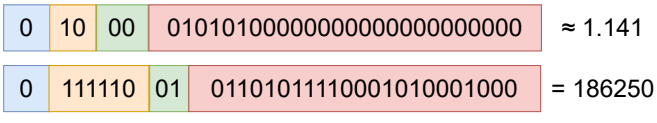


**Figure 6: Binary Comparison of Different Magnitude Posits Showing Regime/Fraction Size Differences**

Because of this structure, posits are less accurate when the magnitude is large because the size of the regime is larger, causing more fraction bits to be truncated, meaning less fractional accuracy. We refer the reader to prior works for additional details [25, 26].

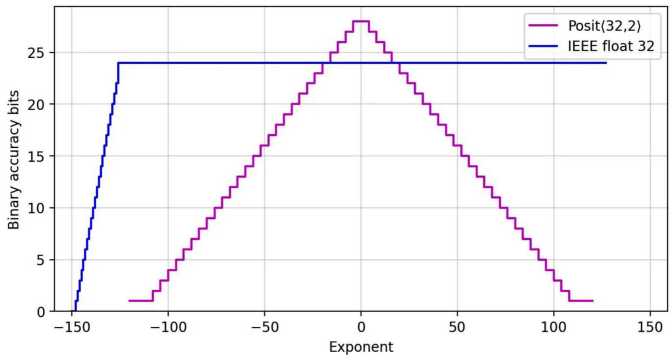


**Figure 7: Posit Fractional Accuracy per Exponent Value [25]**

## 3.3 Soft Errors

Soft Errors are transient upset events in computing systems where charged particles — e.g., cosmic radiation — interact with computing hardware and manifest as bit-corruption in registers and data paths [34, 44, 45]. This corruption often occurs without the user's or system's knowledge and is referred to as silent data corruption (SDC). Prior works show that a single bit corruption inside a running application leads to crashes and significant corruption of the output [12, 13, 20]. To guard against this worst case scenario, researchers have devised solutions at the hardware [18, 38] and software level [15, 19, 23, 30].

# 4 METHODOLOGY

## 4.1 Fault Injection Campaign

In order to study the resiliency of posits and compare them to IEEE-754 floats, we need to define and conduct a fault injection campaign that executes the experiment. This allows us to systematically conduct bit-flips in different positions a certain number of times across different datasets. The fault injection campaign launches a series of fault injection trials for each bit position. Each trial injects a single bit-flip into a random floating-point value in the data, and then computes metrics to quantify the error. We execute 313 trials for each of the 32 bits in the posit, for a grand total of approximately 10,000 trials per field within each dataset. This number allows for diverse data selection, while not being computationally prohibitive. For efficiency, we execute the fault injection campaigns for the individual fields in parallel across different compute nodes in a cluster.

Initially, a running instance of the fault injection campaign reads a binary file containing a field from a scientific data set (see Table 1), and loads it into an array. Then, we calculate basic statistics for the original data using functions we wrote. This provides a baseline that the error metrics from the faulty data can be compared to. After

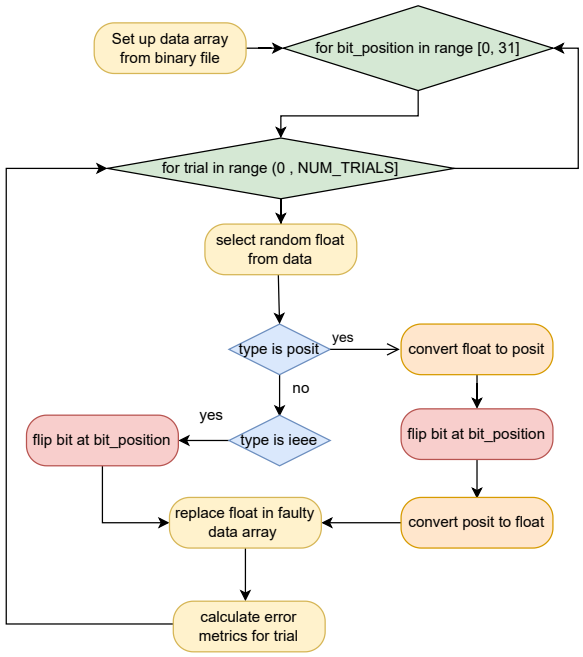


**Figure 8: Flowchart for Fault Injection Campaign**

that, we seed the random number generator for reproducibility, and allocate memory for the faulty data to be written to.

After the initial set up, the bit flipping trials begin. Each of these trials takes a single float and flips a single bit. For each of the 32 bit positions in the posit type, the following steps are executed. First, the trial is set up by generating a random index to select the float, and copying the original data to the faulty array to clean up the faulty value from the previous trial. Then, a bit mask is created with a one in the bit position to be flipped, and zeros elsewhere. Because the process for flipping a bit in IEEE floats and posits is different, the program then detects whether the type is an IEEE or posit.

*4.1.1 IEEE.* To allow for bitwise operations on the float, we interpret its bit representation as an unsigned integer. Then, we XOR the unsigned integer form and the bit mask created earlier in the trial to flip the bit, as we see in Figure 9. We use XOR because it provides the behavior we desire of flipping the bit in the float at the position of the 1 in the mask. All bits with a corresponding mask value of 0 are unaffected due to the nature of XOR. Now that the bit is flipped, the unsigned integer is reinterpreted as a float and added to the faulty array at the index of the original datum.

*4.1.2 Posit.* To perform the necessary conversions from IEEE-754 floats to posits, we leverage a posit arithmetic library called SoftPosit [14], which is a software implementation of the current posit standard. We first convert the randomly selected float into a 32-bit posit using the SoftPosit library function `convertFloatToP32(float)`.

**Table 1: Evaluation Dataset Summary.**

| Dataset | Field | Dimensions | Mean | Median | Max | Min | Std. Dev. |
|---|---|---|---|---|---|---|---|
| CESM | OMEGA-1-26-1800-3600 | 26 x 1800 x 3600 | -3.88E-06 | 3.41E-06 | 4.18E-03 | -5.01E-03 | 3.11E-04 |
| CESM | CLOUD-1-26-1800-3600 | 26 x 1800 x 3600 | 6.37E-02 | 2.89E-02 | 9.64E-01 | -1.14E-17 | 7.42E-02 |
| CESM | RELHUM-1-26-1800-3600 | 26 x 1800 x 3600 | 4.07E+01 | 4.56E+01 | 9.96E+01 | 1.12E-03 | 2.02E+01 |
| EXAFEL | smd-cxif5315-r129-dark | 50 x 32 x 185 x 388 | 2.18E-35 | 2.02E-35 | 9.53E-01 | 6.81E-43 | 1.94E-03 |
| HACC | vy | 280953867 | 4.08E+00 | -4.98E-01 | 3.74E+03 | -3.50E+03 | 2.41E+02 |
| HACC | vx | 280953867 | 1.79E+01 | 2.34E+01 | 3.39E+03 | -3.52E+03 | 2.27E+02 |
| HACC | vz | 280953867 | 2.45E+00 | -1.17E+00 | 3.18E+03 | -4.08E+03 | 2.63E+02 |
| Hurricane | PRECIPf48 | 100 x 500 x 500 | 1.24E-05 | 7.09E-09 | 7.51E-03 | 0.00E+00 | 7.77E-05 |
| Hurricane | Wf30 | 100 x 500 x 500 | 6.91E-03 | -7.78E-05 | 1.55E+01 | -4.57E+00 | 1.72E-01 |
| Hurricane | Uf30 | 100 x 500 x 500 | -5.54E-01 | -6.93E-01 | 6.89E+01 | -7.95E+01 | 9.36E+00 |
| Hurricane | Pf48 | 100 x 500 x 500 | 3.76E+02 | 2.25E+02 | 3.22E+03 | -3.41E+03 | 4.55E+02 |
| Hurricane | CLOUDf48 | 100 x 500 x 500 | 8.60E-06 | 0.00E+00 | 2.05E-03 | 0.00E+00 | 5.18E-05 |
| Hurricane | Vf30 | 100 x 500 x 500 | 3.63E+00 | 3.48E+00 | 6.98E+01 | -6.86E+01 | 9.76E+00 |
| Nyx | velocity-x | 512 x 512 x 512 | 3.54E+02 | 4.68E+05 | 3.19E+07 | -5.04E+07 | 4.97E+06 |
| Nyx | dark-matter-density | 512 x 512 x 512 | 1.00E+00 | 3.93E-01 | 1.38E+04 | 0.00E+00 | 8.37E+00 |
| Nyx | temperature | 512 x 512 x 512 | 8.45E+03 | 7.09E+03 | 4.78E+06 | 2.28E+03 | 1.54E+04 |

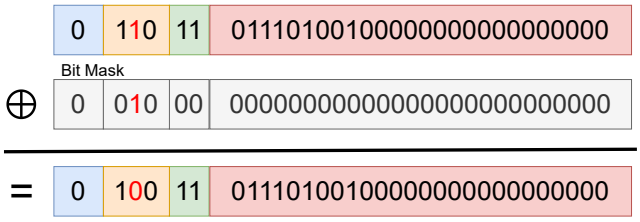


**Figure 9: Injecting a Bit-Flip Example**

When we perform bit-flips on posits, we first extract the unsigned integer member of the `posit32_t` struct returned by the conversion function, which contains the bit field representing the posit value. Next, we perform XOR with the mask to flip the bit. Our conversion accuracy test shows that calling `p32_to_ui32(posit_32t)` and `ui32_to_p32(uint32_t)` performs rounding, and introduces a relative error of $10^{-5}$ to the experimental results. We use the unsigned integer struct member instead of the conversion function to evade this.

To convert back to posit, we assign the unsigned integer with the bit-flip back to the `posit32_t` struct. Finally, we convert the posit back to an IEEE-754 float using another SoftPosit library call. We then add the faulty float to the array for faulty data at the index of the original float, such that this float is the only faulty one in this array.

## 4.2 Error Evaluation Metrics

After the bit-flip occurs, we use the faulty data to calculate error metrics that quantify the error. The first metrics we compute for the faulty data are summary statistics: mean, max, min, and standard deviation. This is done with the same functions we use for the initial baseline. For bit-flips that cause a small change in the magnitude of the value, we do not expect these to deviate significantly from the original. However, in extreme cases with a large magnitude shift, certain summary statics may shift drastically.

Additionally, we compute a variety of error metrics between the original and faulty data. We apply Quick Compression Analysis Toolkit (QCAT) to calculate the absolute error, relative error, mean squared error, and norm error, since they most accurately quantify the error caused by our bit-flip. Once we compute the metrics, we write them to a log file in CSV form for offline analysis and visualization.

# 5 EXPERIMENTAL RESULTS

## 5.1 Testing Environment

All the experiments in this study are run on Clemson University's Palmetto Cluster, a 3.0+ petaflop heterogeneous system with 1,786 compute nodes and 34,916 CPU cores. The cluster has a variety of processors throughout its nodes. For the fault injection campaign, we use nodes containing two Intel Xeon E5-2665 2.40GHz processors and 64 GB of memory. Our software environment includes gcc version 8.5.0 for compilation, SoftPosit version 0.4.1 for posit conversion [14], and QCAT version 1.3 for calculation of error metrics.

## 5.2 Datasets

The data we use in these experiments is from the Scientific Data Reduction Benchmarks Site [43]. Specifically, the datasets used are Hurricane Isabel (weather simulation), Nyx (cosmology), CESM (climate), EXAFEL (Images from the LCLS instrument) and HACC (Cosmology). Table 1 presents summary information on our testing datasets, including the name, size, basic statistics.

## 5.3 Posit vs IEEE-754 Summary

Our experiments show a large improvement in posit resiliency over the IEEE-754 standard, confirming prior work[8]. As a general representation of the experimental results, Figure 10 shows a comparison of the mean relative error across all trials for each bit position,

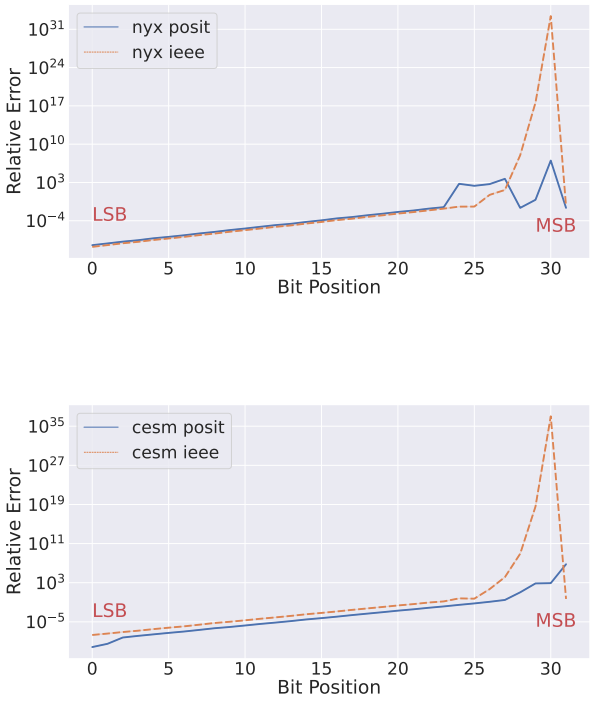


**Figure 10: Posit vs IEEE-754 Mean Relative Error in Nyx and CESM**

plotted for both IEEE-754 floats and posits for NYX and CESM, respectively. While Figure 10 only shows NYX and CESM data, we observe a similar trend for other datasets. Across all datasets, the IEEE-754 floats have a sharp and consistent exponential spike in error towards the most significant bit positions (exponent and sign). However, posits have lower, but more distributed and erratic error in the upper bit positions (sign and regime). In some situations, posits yield larger errors than IEEE-754 floats. Although they are slightly offset by different amounts in different datasets in the lower bit positions, the slope of the error increase is similar in both posits and IEEE-754 floats (fraction). In the following sections, we explore in depth why these trends occur.

## 5.4 Regime Bits

The regime bits are the primary factor that differentiates posits from IEEE-754 floats. This field has the most significant contribution to the magnitude, so bit-flips in the regime frequently cause substantial error. To understand the implications of bit-flips in different bit positions of the regime, we compute the average absolute error from flips in posits with different regime sizes. This method isolates error trends in different regime bits. It is also necessary because most datasets contain posits with a variety of regime sizes, leading to noisy results when focusing on regime error trends without separation. To execute this, the equation to calculate regime size 1 is implemented to sort results from different datasets collected during the fault injection campaign. The results reveal two distinct error trends in different posit magnitude ranges. In posits with an absolute value greater than one, there is an error spike associated with the terminating regime bit $R_k$. Posits with an absolute value less than one show large error spikes in the sign bit, but not elsewhere. The following explains this in more detail.

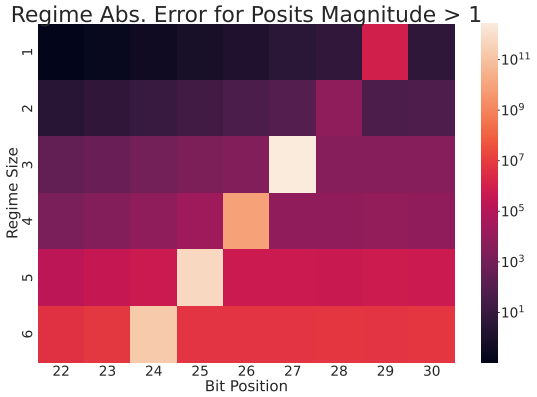


**Figure 11: Average Relative Error in Posits with Magnitude Greater than One**

*5.4.1 Error in Posits with Absolute Value Greater than One.* Figure 11 shows the average absolute error for different bit positions in posits with varying regime sizes. Figure 11 shows a spike in error is associated with the terminating bit of the regime ($R_k$). The reason for this trend is that when $R_k$ is flipped, it takes on the value of the regime bits $R_0$ through $R_{k-1}$. Since the regime field is terminated by a bit opposite the state of $R_0$, the regime expands into what was once the exponent and fraction until an opposite bit is detected. This is visualized in figure 12, where a numerical example shows the impact of this event on a posit using equation 2.

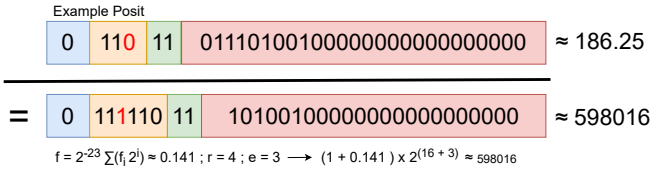


**Figure 12: Bit-flip induced regime expansion.**

If the exponent and most significant bits in the fraction are the same state as the faulty state of $R_k$, the size of the regime can increase significantly. This scales the magnitude of the posit by $2^{4n}$ where $n$ is the number of new regime bits. However, if a bit opposite the flipped state of $R_k$ occurs immediately after $R_k$, the error is much less significant since the size of the regime only increases by 1 bit.

We also see that there is a consistent error across regime bits $[R_0, R_{k-1}]$. The error in these regime bits increases with regime size because the magnitude of the posits increase. When a flip occurs in

one of these bits, the regime size shrinks, reducing the magnitude of the posit. The consistency of the error is due to the exponential contribution regime size has on the magnitude of a posit. Because of this, a bit-flip in one of the regime bits in $[R_0, R_{k-1}]$ causes a reduction in magnitude. Even a flip in $R_{k-1}$ does not typically produce an absolute error more significant than previous bits, which is shown in figure 13. This figure uses a numerical example to demonstrate how absolute error will not change significantly when bits $[R_0, R_{k-1}]$ are flipped, as shown in figure 11, despite exponentially increasing error caused by bit flips approaching $R_k$.

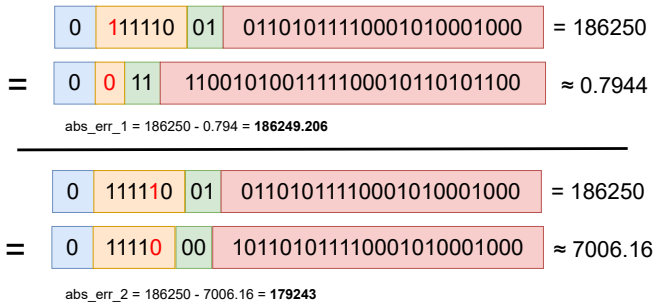


**Figure 13: Absolute Error Comparison in $R_0$ vs $R_{k-1}$**

*5.4.2 Posits with Absolute Value Less than 1.* Posits with an absolute value less than one show no relative error spike in the terminating regime bit, $R_k$. This is due to the small scale of the numbers. Despite the small quantifiable error, regime expansion occurs when $R_k$ is flipped. However, this flip decreases the magnitude of the posit. It is not possible for the posit to increase in magnitude, or decrease below 0 in this case. Thus, the overall change in magnitude is relatively small. Therefore, the relative error does not spike significantly even when the regime expands by several bits or the posit decreases by several orders of magnitude. This is shown in example 4. In most cases, the relative error is near one. This example takes a posit with original magnitude $3.395274 \times 10^5$ and faulty magnitude $8.644184 \times 10^8$, and demonstrates the low relative error in this case.

$$\frac{|orig - faulty|}{orig} = \frac{|3.395 \times 10^{-5} - 8.644 \times 10^{-8}|}{3.395 \times 10^{-5}}$$
$$\approx \frac{3.395 \times 10^{-5}}{3.395 \times 10^{-5}} \approx 1$$

For posits in this range with regime size 1, we notice an extreme spike (up to $10^{11}$) in absolute error in bit position 30, which is the sole regime bit. This error is not shown in figure 14 to make the general trend more readable. These spikes in error come from an edge case where the flipping of the sole regime bit will not only expand the regime, but also invert it. Figure 15 demonstrates this with a numeric example. The flip in $R_0$ not only expands the regime from 1 bit to 5 bits, but inverts the terminating bit $R_k$. Thus, the value of $r$ in equation 2 changes significantly. This causes a large error since the magnitude of the posit is scaled by $2^{4r}$.

Due to the way the regime is interpreted, when the regime bits are inverted completely, meaning the state of $R_0$ through $R_{k-1}$ and $R_k$ are inverted, the sign of the regime component will change. This

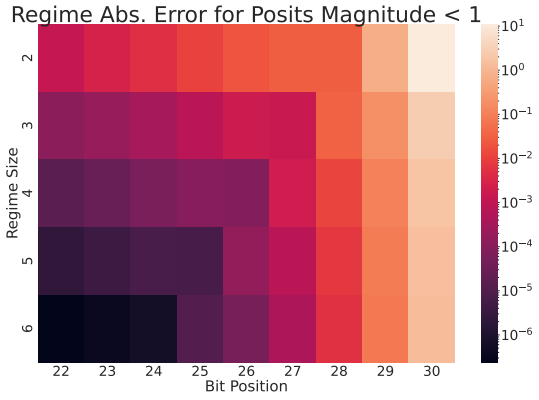


**Figure 14: Average Relative Error in Posits with Magnitude Less than One**

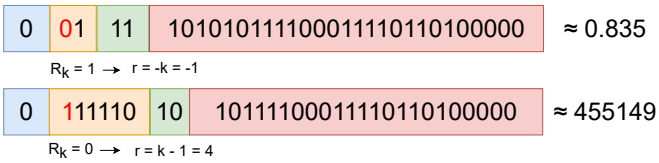


**Figure 15: Edge Case Where Regime Expands and Inverts**

can be seen in equation 2 and the description below it. If this edge case occurs, the magnitude of the posit will change drastically.

*5.4.3 Regime Discussion.* These results show that posits with magnitude greater than one are exceptionally susceptible to bit-flips in regime bit $R_k$, but bit-flips anywhere in the regime still causes significant changes in magnitude. Posits with magnitude less than one are not impacted significantly by bit-flips in $R_k$, but are still susceptible to flips in other regime bits.

In most cases, the impact of bit-flips in the regime of posits is still far less substantial than the upper exponent bits of IEEE-754 floats by many orders of magnitude, which is a major improvement. Figure 10 shows that unlike the uniform error spike in IEEE floats, posits have a more erratic distribution of error. This is due to the combination of the error spikes coming from flipping the final bit in the regime field. Because the size of the regime depends on the magnitude of the posit, the width of the error distribution depends on the variance and median of the data. Datasets with large variances and medians have a wider error distribution since there are more values with larger numbers of regime bits, as shown by the standard deviations of Nyx and CESM in 1. This causes error spikes in lower bit positions, since $R_k$ is in a lower position. Unlike in IEEE floats where the error increases as the bit position increases, error in the upper bit positions of posits comes in spikes with location depending on the regime bits.

## 5.5 Fraction

The part of the posit that usually takes up the majority of the bits is the fraction. However, the fraction bits do not contribute as significantly to the magnitude of the posit compared to other fields. This is because the magnitude of the posit scales linearly as the value of the fraction bit field changes, denoted $f$ in equation 2. The main difference between IEEE and posit fractions is that the latter has a variable size, which depends on the number of regime bits.

The error in the fraction bit field is visualized in Figure 16. This plot was created with data from the HACC and Hurricane datasets with 1 regime bit (k = 1). A specific number of regime bits is selected to maintain an equal fraction size among posits. Specifically, 1 regime bit posits are chosen because they are the most plentiful in our data pool. Additionally, it is observed that fractional error is not correlated with regime size, indicated by similar trends in posits with regime sizes 1 - 6. Furthermore, this plot is in log scale to make the error trend more clear.

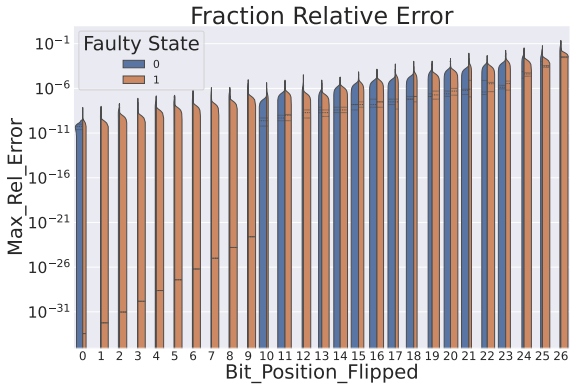


**Figure 16: Relative Error Fraction**

The results in figure 16 are expected because the significance of bits in the fraction doubles for each increment towards the MSB. This is indicated by equation 3, which shows that the state of each bit in the fraction is multiplied by $2^{index}$. Thus, the magnitude of the error associated with a specific fractional bit position will be approximately double the previous bit. Depending on whether the fraction has more or less bits, there is more or less error associated with flips in bits closer to the MSB.

## 5.6 Exponent

In IEEE floats, the exponent bits are associated with much larger error than the fraction due to the exponential effect their value has on the magnitude. This raises the question: is the massive error spike in the exponent field still an issue in posits? Our experiments find that unlike IEEE floats, there is no relatively significant error associated with the exponent field in posits.

In posits, the exponent field is in between the regime and fraction fields, and spans a constant 2 bits. Because the exponent in posits is much smaller than that of IEEE floats, flipping one of the exponent bits causes a smaller error. The error caused by this would be the same as flipping the corresponding bit in an IEEE float, since the magnitude of both IEEE floats and posits are scaled by $2^{exponent}$. Since posit exponents only have two bits, the max potential magnitude shift due to an exponent bit flip would be multiplying or dividing the original value of the posit by 4. This is because the value of the exponent in equation 2 is increased by 2. In IEEE floats, the exponent value will shift by 128 when the uppermost bit is flipped, causing a magnitude shift of $2^{128}$. This is because the exponent is 8 bits in IEEE floats. Figure 17 visualizes a case where the uppermost bit positions in an IEEE-754 float and posit are flipped.

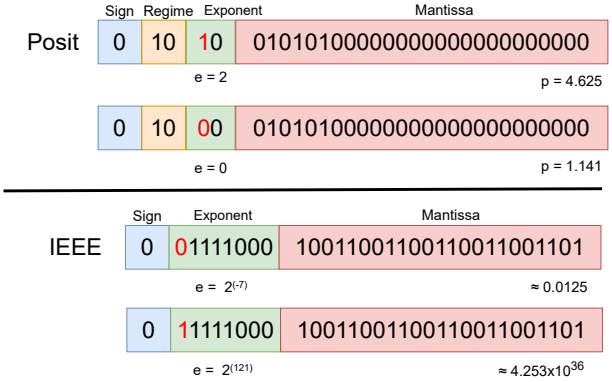


**Figure 17: Uppermost Exponent Bit Flip**

To support this, figure 18 shows how the smooth error increase trend in the fraction does not break when the exponent is reached. This indicates that the error associated with the exponent bits is similar to the trend in the fraction. It is not visible through error where the exponent is located in the posit, unlike in IEEE floats where the exponent is denoted by a large spike in error. The trend in the exponent bits is similar to the trend in the fraction.

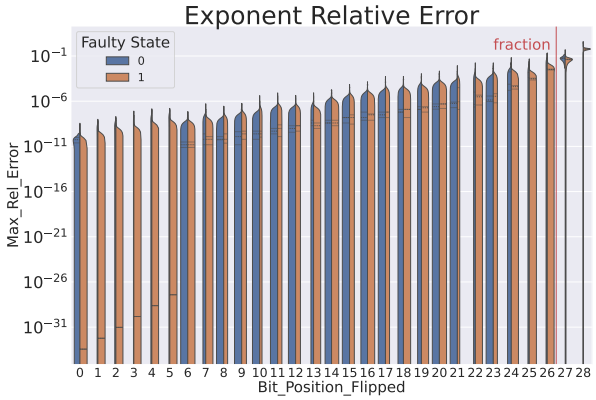


**Figure 18: Relative Error in Exponent compared to Fraction**

### 5.7 Sign Bit

The sign bit in posits has a greater impact on the magnitude than the IEEE-754 sign bit. Because of this, it is important to understand how flips in the sign bit affect the magnitude of posits. In IEEE floats, when the sign bit is flipped, the magnitude is unchanged, and just the sign of the number is affected. The result of the flip will be the negation of the float, but this is not the case in posits. To negate a posit, the two's complement of the bits must be taken and its magnitude evaluated (shown in figure 19). Therefore, when the sign bit alone is flipped in a posit, not only will the sign of the number change, the magnitude will change as well. The presence of the sign variable $s$ in positions that affect the magnitude in Equation 2 shows why this occurs.

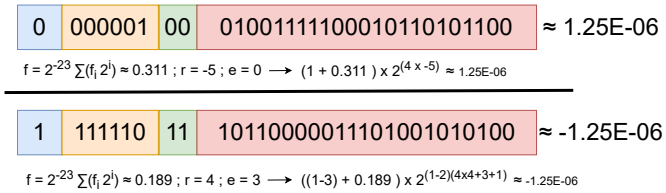


**Figure 19: Negation of a Posit**

This change in magnitude can sometimes be drastic. Figure 20 shows the absolute error caused by sign flips in posits with different regime sizes. We use a box plot because a violin plot would not fit this data well due to variation in the distributions and number of samples. Posits of all magnitude ranges are included, since this sign bit error was consistent across all posit magnitude ranges. This figure shows that the absolute error from flipping the sign bit increases exponentially as the regime size increases. This means posits with extremely large magnitudes are most effected by sign bit flips. Posits with magnitude close to 1 with small regimes are not affected as significantly.

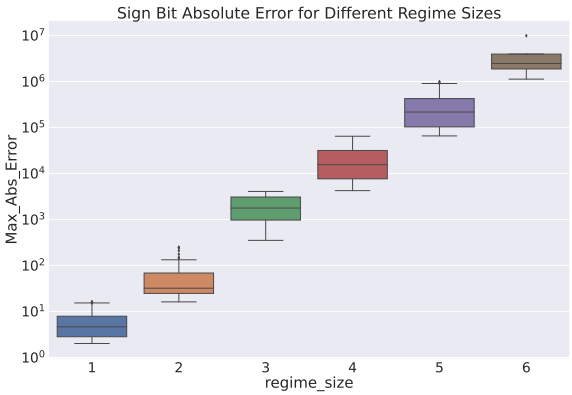


**Figure 20: Sign Bit Error In Small Magnitude Posits with Different Regime Sizes**

The reason massive error occurs when the sign bit is flipped can be seen in the formula for posit magnitude 2 and the numeric example in figure 21. This example demonstrates that when the sign bit is 1, it multiplies the exponent in equation 2 containing the regime, and exponent values by −1. Flipping the sign bit also flips the sign of the exponent in equation 2, which can drastically affect the magnitude.

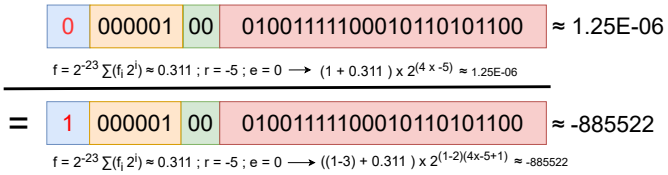


**Figure 21: Error Caused by Flipping Sign Bit**

This effect is not as significant in posits close to 1, because the regime size in the exponent is small, and therefore this exponential sign flip does not produce as large of a deviation from the original magnitude.

## 6 CONCLUSION

As the usage of posits continues to grow due to their higher precision representation of certain numbers, it is important to understand their resilience to bit flips. This study shows that posits possess more resiliency to bit flips than IEEE-754 floats in most cases. After examining the effect of bits flips in all major components of the posit, we have found and analyzed patterns and quirks regarding bit flip error in posits. The presence of the regime reduces the number of bits that cause catastrophic error in posits compared to IEEE floats. This leads to overall less error in upper bit positions except in specific cases. We found flips in the sign bit usually cause more error in posits due to the way sign affects the magnitude, unlike in IEEE floats. We also found that unlike in IEEE floats, the posit exponent does not produce significant error due to its small size. Lastly the fraction error was found to be similar to that of IEEE floats.

To expand knowledge on this topic further, more research is necessary. Potential future research topics include the following. Different size posits will react differently to bit flips, so fault injection campaigns on 8, 16 and 64 bit posits would be beneficial. Multi-bit flip analysis would also provide valuable insights. Mathematical analysis could be done to predict potential error in posits due to bit flips.

## ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197 and SHF-1943114.

## REFERENCES

[1] [n. d.]. ARIANE 5 Failure - Full Report. http://sunnyday.mit.edu/nasa-class/Ariane5-report.html. Accessed: 2023-04-20.
[2] [n. d.]. NVIDIA H100 Tensor Core GPU Architecture. https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper. Accessed: 2023-04-20.
[3] 1985. IEEE Standard for Binary Floating-Point Arithmetic. *ANSI/IEEE Std 754-1985* (1985), 1–20. https://doi.org/10.1109/IEEESTD.1985.82928

[4] 2000. What Every Computer Scientist Should Know About Floating-Point Arithmetic. https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html.

[5] 2023. https://en.wikipedia.org/wiki/IEEE_754

[6] Ankur Agrawal, Silvia M. Mueller, Bruce M. Fleischer, Xiao Sun, Naigang Wang, Jungwook Choi, and Kailash Gopalakrishnan. 2019. DLFloat: A 16-b Floating Point Format Designed for Deep Learning Training and Inference. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. 92–95. https://doi.org/10.1109/ARITH.2019.00023

[7] Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Jean Roman, and Mawussi Zounon. 2016. Numerical recovery strategies for parallel resilient Krylov linear solvers. *Numerical Linear Algebra with Applications* 23, 5 (2016), 888–905.

[8] Ihsen Alouani, Anouar Ben Khalifa, Farhad Merchant, and Rainer Leupers. 2021. An Investigation on Inherent Robustness of Posit Data Representation. In *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*. 276–281. https://doi.org/10.1109/VLSID51830.2021.00052

[9] David F. Bacon, Susan L. Graham, and Oliver J. Sharp. 1994. Compiler Transformations for High-Performance Computing. *ACM Comput. Surv.* 26, 4 (dec 1994), 345–420. https://doi.org/10.1145/197405.197406

[10] Leonardo Bautista-Gomez, Ferad Zyulkyarov, Osman Unsal, and Simon McIntosh-Smith. 2016. Unprotected Computing: A Large-scale Study of DRAM Raw Error Rate on a Supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, Utah) *(SC '16)*. IEEE Press, Piscataway, NJ, USA, Article 55, 11 pages. http://dl.acm.org/citation.cfm?id=3014904.3014978

[11] Jon Calhoun, Luke Olson, and Marc Snir. 2014. FlipIt: An LLVM Based Fault Injector for HPC. In *Proceedings of the 20th International Euro-Par Conference on Parallel Processing (Euro-Par '14)*.

[12] Jon Calhoun, Luke N. Olson, Marc Snir, and William D. Gropp. 2017. Towards a More Complete Understanding of SDC Propagation. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing* (Washington D.C., USA) *(HPDC '17)*. ACM, New York, NY, USA.

[13] Marc Casas, Bronis R. de Supinski, Greg Bronevetsky, and Martin Schulz. 2012. Fault resilience of the algebraic multi-grid solver. In *Proceedings of the 26th ACM international conference on Supercomputing* (San Servolo Island, Venice, Italy) *(ICS '12)*. ACM, New York, NY, USA, 91–100. https://doi.org/10.1145/2304576.2304590

[14] Cerlane Leong. [n. d.]. https://gitlab.com/cerlane/SoftPosit. Online.

[15] Zizhong Chen. 2011. Algorithm-based recovery for iterative methods without checkpointing. In *Proceedings of the 20th international symposium on High performance distributed computing* (San Jose, California, USA) *(HPDC '11)*. ACM, New York, NY, USA, 73–84. https://doi.org/10.1145/1996130.1996142

[16] Steven W. D. Chien, Ivy B. Peng, and Stefano Markidis. 2020. Posit NPB: Assessing the Precision Improvement in HPC Scientific Applications. In *Parallel Processing and Applied Mathematics*, Roman Wyrzykowski, Ewa Deelman, Jack Dongarra, and Konrad Karczewski (Eds.). Springer International Publishing, Cham, 301–310.

[17] Florent de Dinechin, Luc Forget, Jean-Michel Muller, and Yohann Uguen. 2019. Posits: The Good, the Bad and the Ugly. In *Proceedings of the Conference for Next Generation Arithmetic 2019* (Singapore, Singapore) *(CoNGA'19)*. Association for Computing Machinery, New York, NY, USA, Article 6, 10 pages. https://doi.org/10.1145/3316279.3316285

[18] Timothy J. Dell. 1997. *A White Paper on the Benefits of ChipkillCorrect ECC for PC Server Main Memory*. Technical Report. IBM Microelectronics Division.

[19] Sheng Di and Franck Cappello. 2016. Adaptive Impact-Driven Detection of Silent Data Corruption for HPC Applications. *IEEE Trans. Parallel Distrib. Syst.* 27, 10 (Oct. 2016), 2809–2823. https://doi.org/10.1109/TPDS.2016.2517639

[20] James Elliott, Mark Hoemmen, and Frank Mueller. 2014. Evaluating the Impact of SDC on the GMRES Iterative Solver. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS '14)*. IEEE Computer Society, Washington, DC, USA, 1193–1202. https://doi.org/10.1109/IPDPS.2014.123

[21] James Elliott, Frank Mueller, Frank Stoyanov, and Clayton Webster. 2013. *Quantifying the impact of single bit flips on floating point arithmetic*. Technical Report. North Carolina State University. Dept. of Computer Science.

[22] Mohammad Eslami, Behnam Ghavami, Mohsen Raji, and Ali Mahani. 2020. A survey on fault injection methods of digital integrated circuits. *Integration* 71 (2020), 154–163. https://doi.org/10.1016/j.vlsi.2019.11.006

[23] David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. 2012. Detection and Correction of Silent Data Corruption for Large-scale High-performance Computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, Utah) *(SC '12)*. IEEE Computer Society Press, Los Alamitos, CA, USA, Article 78, 12 pages. http://dl.acm.org/citation.cfm?id=2388996.2389102

[24] Dakota Fulp, Alexandra Poulos, Robert Underwood, and Jon C. Calhoun. 2021. ARC: An Automated Approach to Resiliency for Lossy Compressed Data via Error Correcting Codes. In *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing* (Virtual Event, Sweden) *(HPDC 21)*. Association for Computing Machinery, New York, NY, USA, 57–68. https://doi.org/10.1145/3431379.3460638

[25] Dina Genkina. 2023. Posits, a new kind of number, improves the math of ai. https://spectrum.ieee.org/floating-point-numbers-posits-processor

[26] John Gustafson, Gerd Bohlender, Shin Yee Chung, Vassil Dimitrov, Geoff Jones, Siew Hoon Leong (Cerlane), Peter Lindstrom, Theodore Omtzigt, Hauke Rehr, Andrew Shewmaker, and Isaac Yonemoto. 2021. *Posit Standard Documentation Release 4.12-draft*. Technical Report. National Supercomputing Centre (NSCC) Singapore.

[27] J. Gustafson and I. Yonemoto. 2017. Beating Floating Point at its Own Game: Posit Arithmetic. *Supercomputing Frontiers and Innovations* 4, 2 (Jun 2017). https://doi.org/10.14529/jsfi170206

[28] Siva Kumar Sastry Hari, Sarita V. Adve, and Helia Naeimi. 2012. Low-cost Program-level Detectors for Reducing Silent Data Corruptions. In *Proceedings of the 2012 42Nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (DSN '12)*. IEEE Computer Society, Washington, DC, USA, 1–12. http://dl.acm.org/citation.cfm?id=2354410.2355132

[29] Kuang-Hua Huang and Jacob A. Abraham. 1984. Algorithm-Based Fault Tolerance for Matrix Operations. *IEEE Trans. Comput.* C-33, 6 (1984), 518–528. https://doi.org/10.1109/TC.1984.1676475

[30] Kuang-Hua Huang and J. A. Abraham. 1984. Algorithm-Based Fault Tolerance for Matrix Operations. *IEEE Trans. Comput.* 33, 6 (June 1984), 518–528. https://doi.org/10.1109/TC.1984.1676475

[31] Ignacio Laguna, Martin Schulz, David F. Richards, Jon Calhoun, and Luke Olson. 2016. IPAS: Intelligent Protection Against Silent Output Corruption in Scientific Applications. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization* (Barcelona, Spain) *(CGO 2016)*. ACM, New York, NY, USA, 227–238. https://doi.org/10.1145/2854038.2854059

[32] Guanpeng Li, Karthik Pattabiraman, Siva Kumar Sastry Hari, Michael B. Sullivan, and Timothy Tsai. 2018. Modeling Soft-Error Propagation in Programs. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (DSN '18)*.

[33] Qining Lu, Mostafa Farahani, Jiesheng Wei, Anna Thomas, and Karthik Pattabiraman. 2015. LLFI: An Intermediate Code-Level Fault Injection Tool for Hardware Faults. In *Proceedings of the 2015 IEEE International Conference on Software Quality, Reliability and Security (QRS '15)*. IEEE Computer Society, Washington, DC, USA, 11–16. https://doi.org/10.1109/QRS.2015.13

[34] T. C. May and Murray H. Woods. 1979. Alpha-particle-induced soft errors in dynamic memories. *Electron Devices, IEEE Transactions on* 26, 1 (Jan. 1979), 2–9. https://doi.org/10.1109/T-ED.1979.19370

[35] Simon McIntosh–Smith, Rob Hunt, James Price, and Alex Warwick Vesztrocy. 2017. Application-based fault tolerance techniques for sparse matrix solvers. *The International Journal of High Performance Computing Applications* 0, 0 (2017), 1094342017694946. https://doi.org/10.1177/1094342017694946 arXiv:https://doi.org/10.1177/1094342017694946

[36] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

[37] Xiang Ni, Esteban Meneses, Nikhil Jain, and Laxmikant V. Kale. 2013. ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '13)*. IEEE Computer Society.

[38] Steven K. Reinhardt and Shubhendu S. Mukherjee. 2000. Transient Fault Detection via Simultaneous Multithreading. In *Proceedings of the 27th Annual International Symposium on Computer Architecture* (Vancouver, British Columbia, Canada) *(ISCA '00)*. ACM, New York, NY, USA, 25–36. https://doi.org/10.1145/339647.339652

[39] Aleksandr Yu. Romanov, Alexander L. Stempkovsky, Ilia V. Lariushkin, Georgy E. Novoselov, Roman A. Solovyev, Vladimir A. Starykh, Irina I. Romanova, Dmitry V. Telpukhov, and Ilya A. Mkrtchan. 2021. Analysis of Posit and Bfloat Arithmetic of Real Numbers for Machine Learning. *IEEE Access* 9 (2021), 82318–82324. https://doi.org/10.1109/ACCESS.2021.3086669

[40] Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen, David H. Bailey, Costin Iancu, and David Hough. 2013. Precimonious: Tuning Assistant for Floating-Point Precision. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '13)*. Association for Computing Machinery, New York, NY, USA, Article 27, 12 pages. https://doi.org/10.1145/2503210.2503296

[41] Yohann Uguen, Luc Forget, and Florent de Dinechin. 2019. Evaluating the Hardware Cost of the Posit Number System. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. 106–113. https://doi.org/10.1109/FPL.2019.00026

[42] Jiesheng Wei, Anna Thomas, Guanpeng Li, and Karthik Pattabiraman. 2014. Quantifying the Accuracy of High-Level Fault Injection Techniques for Hardware Faults. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 375–382. https://doi.org/10.1109/DSN.2014.2

[43] Kai Zhao, Sheng Di, Xin Lian, Sihuan Li, Dingwen Tao, Julie Bessac, Zizhong Chen, and Franck Cappello. 2020. SDRBench: Scientific Data Reduction Benchmark for Lossy Compressors. In *2020 IEEE International Conference on Big Data (Big Data)*. 2716–2724. https://doi.org/10.1109/BigData50022.2020.9378449

[44] James Ziegler and Helmut Puchner. 2004. *SER – History, Trends and Challenges: A guide for designing with Memory ICs*. Cypress.

[45] J. F. Ziegler and W. A. Lanford. 1979. Effect of Cosmic Rays on Computer Memories. In *Science*, Vol. 206. 776–788. http://www.sciencemag.org/cgi/content/abstract/206/4420/776