Bit Corruption with Floating Point Numbers and Posits

1st Griffin Bailey
Computer Engineering
Clemson University
Clemson, SC
gjbaile@clemson.edu

2nd Palmer Cone
Computer Science
Clemson University
Clemson, SC
pcone@clemson.edu

3rd Noah Case
Computer Engineering
Clemson University
Clemson, SC
ncase@clemson.edu

4th Joshua Apter
Computer Engineering
Clemson University
Clemson, SC
japter@clemson.edu

***Abstract*—Floating point arithmetic is common place in high performance computing (HPC) systems. Faults such as bit corruption can detrimental to computation results on these HPC data sets. The majority of floating point numbers in HPC systems are stored using the IEEE-754 standard which can be very susceptible to large skews from the expected data value.**

This paper studies the resiliency of a different method of storing floating point values – Posits. Posits have been studied to be an effective way to save on storage cost at a similar performance, but not much is known about their resiliency as compared to the IEEE-754 standard. We use bitwise operations to inject bit flip faults into five different EXAALT datasets retrieved from SDRBench. We then calculate a variety of statistical data to describe the sets before and after the bit flip to determine if Posits are more resilient to these types of faults.

I. Introduction

In HPC systems, floating point arithmetic presents a certain amount of issues. One of these is rounding error. In the IEEE standard, there are a fixed amount of bits for the exponent and fraction portion so in situations where the majority of information is contained in the fraction portion, there may not be enough available bits to store the entire number presented without rounding. This is a source of loss that is addressed with Posits. Posits do not have a fixed number of fraction or exponent bits, if any so they are more versatile to a larger range of floating point data. Although this has been examined to be true, not much is known about the data types resiliency to bit faults. The focus of this work is to provide insight to the data types resiliency through fault injection and statistical analysis to evaluate if Posits should be considered for use with large scientific data sets.

For our analysis, we used data collected from the Los Angeles National Laboratory involving molecular dynamics simulations. We used five specific data sets from the overall data: vy, yy, vx, xx, and vz. Each of the five data sets are approximately 11 kilobytes in size.

This paper is organized into several sections: Section 2 describes the Motivations behind the research. Section 3 details the specifics of the contributions each member had to the research. Section 4 includes graphs of the data collected highlighting the difference in data after the fault injection. Section 5 goes into detail in describing the contrast before and after bit corruption is presented to the data sets. Section 6 concludes and section 7 discusses future work to be done regarding the research.

II. Motivation

As HPC systems reach exascale, I/O bandwidth is a considerable bottleneck. Any way that the amount of storage required can be reduced is very important for the future of computing. This has led to the use of Posits becoming more widespread for I/O benefits. This research is motivated by this fact and provides some insight on whether or not this data type is resilient enough to handle the faults presented in an HPC environment, like bit corruption.

III. Contributions

A. Noah Case

My focus in this project involved the input of EXAALT data and programming of the baseline statistical calculations. For the inputted data, it is simply downloaded from SDRBench page as that is publicly available. For analysis, the data file is scanned through using C and each value is entered into an array for statistical analysis, after that Joshua and I created a C program that calculates statistics on the datasets before and after fault injection.

B. Palmer Cone

My work in this project was primarily focused on Posits and implementing them into our code. I spent a lot of time simplifying the code and finding better methods to implement the SoftPosits. I also added a Makefile that we could use to build SoftPosit for each of our trials, compile, and run the programs. I also spent a great deal of time looking through the SoftPosit repository to learn how the data was structured and what functions we could use.

C. Griffin Bailey

The work I completed revolved around fault injection and automation of testing. After values were initially read in, the code would generate a random number that would act as the index of the bit to be flipped. Those values were then analyzed for later use. In regards to automation, I worked to edit a python script that would start jobs on the Palmetto cluster to parallelize various test cases as they are independent of one

another. I also created a Matlab program that would read in the values from the test cases, create matrices from a CSV formatted file, and create various graphs to represent the data. This was the primary way we analyzed results.

D. Joshua Apter

The bulk of the work I completed for this project revolved around the statistics and correlation metrics. In order to see the effects of flipping a single bit, we needed both baseline statistics of our data set as well as statistics of the effect data. I, with the help of Noah, wrote a C program that would read in 32 bit floats into an array and compute basic statistical analysis on them. After we had calculated these statistics, I expanded our C program to calculate correlation metrics such as RMSE on the two data sets as a whole. In addition to this, I ran the fault injection campaign and analysis of our data using posits.

IV. Results

In our experiments, a single bit flip in a data set proved to be very influential on the summary statistics used to describe the data set. In Figure 1, the mean for the floating point data sets vary in an uncorrelated way. In Figure 4, the mean square error for floating points is shown and the result is much like the mean for floating point data – the results were seemingly uncorrelated. The peak signal-to-noise ratio, seen in Figure 2, is different however, there were some interesting artifacts. At the larger values, there appeared to be no pattern and the values were widely arranged but at 0 and around -60 values for PSNR there appears to be a flat line where it commonly occurs. The results for Posits were fairly different than the IEEE-754 floating point standard in the general form of the graph. In Figure 3 it can be seen that the PSNR for Posits all fall within a much closer range of values than the IEEE data.

V. Discussion

A. Baseline Statistics

In order to get an idea of how flipping a single bit in our data set would effect the data set as a whole, we first ran baseline statistics on our data that we would use for comparison after our fault injection. These statistics included: minimum, maximum, mean, standard deviation, interquartile range, overall range, median, variance, and skew.

- Min/Max
  - Calculation
    We used a for loop to go through our entire data set and find both the minimum and maximum value and store those in an array for comparison and evaluation.
  - Insights
    From our testing we were able to see how flipping a single bit effects our data set on a large scale with this statistic. We saw that because we are only flipping a single bit, our minimum and maximum values where not changed too often. The only times these values would change dramatically was when an exponent bit was flipped, causing drastic changes in our value.
- Mean
  - Calculation
    In order to calculate the average of our data set we looped through the entire array, finding the sum, and then divided our sum by the number of entries in our array.
  - Insight
    The mean gave us an idea of how flipping a single bit will effect the data set as a whole. Our mean value changed on each iteration through our fault injection campaign and this is due to the fact that changing any number would change the average. This statistic was useful in seeing how big of a change a single bit flip can make by comparing entire data sets.
- Standard Deviation
  - Calculation
    In order to calculate the standard deviation, we used a for loop to find the difference between each value and the mean, squared that value and summed all iterations. Once that process was complete, we divided by the total number of values and took the square root. The equation can be seen below:

    $$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

  - Insight
    The standard deviation of our data sets shows us how spread out our numbers are. A small standard deviation would show us that our numbers are relatively similar in value, and large would mean that there is a wide range of values. From our testing, we were able to determine that a single bit flip would change the standard deviation drastically, showing that by changing a single number, the range of our data set is changed greatly.
- Interquartile Range (IQR)
  - Calculation
    In order to calculate the IQR, we ordered our data, found the median value, determined Q1, values below the median, and Q3, values above the median, and subtracted these two numbers. This gives an interquartile range of our entire data set.
  - Insight
    The IQR is a measure of where the bulk of our data lies. This gives us where the middle 50% of our data is which we can use to see how our data is distributed. From our analysis we found that the IQR was relatively stable when comparing it to our original data set.
- Range
  - Calculation
    In order to calculate the range our values, we ordered

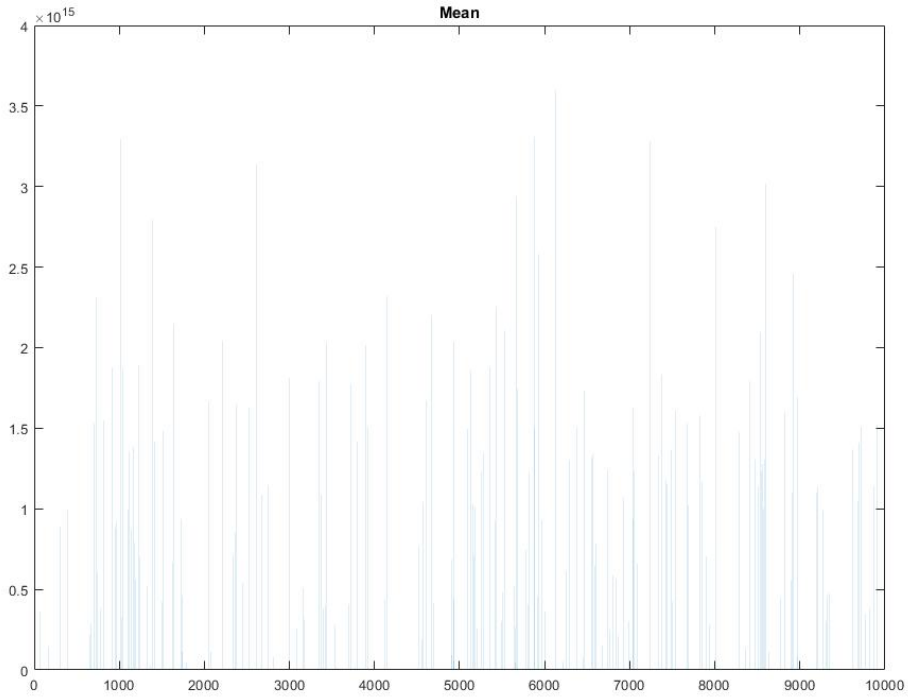

Fig. 1. Difference in Mean for Floating Point Numbers

our data set and simply subtracted the largest value from the smallest.

- Insight
  The range of our data is useful in seeing how much of a spread we have in relation to data value size. This is helpful in seeing how uniform our data is and assists in making our other calculations useful.

- Median
  - Calculation
    In order to calculate the median, we simply ran a for loop to order our data and then selected the middle value in our data set array.
  - Insight
    Similar to range, this value does not give a ton of insight on its own, but allows us to have perspective on our other statistics, making them much more meaningful.
- Variance
  - Calculation
    In order to calculate the variance, we used a process exactly like that of the standard deviation, but did not take the square root of our result in the end. The equation is below:

    $$\sigma^2 = \frac{\sum(x_i - \mu)^2}{N}$$

  - Insight
    While the standard deviation gives us a good idea of how spread out our numbers are from the mean, the variance shows us the average degree to which each point differs from the mean - all data points. As expected, the variance differs rather significantly between fault injections.
- Skew
  - Calculation
    In order to calculate the skew of our data, we ordered our data and found the sum of the difference between each value and the mean. We cubed that value and divided by the number of variables minus one times the standard deviation cubed. The equation can be seen below:

    $$\mu_3 = \frac{\sum(x_i - x)^3}{(N-1) * sigma^3}$$

  - Insight
    The skew gave us an insight into how much the data is spread out relative to the mean. A positive skew means the majority of our data lies above the mean and a negative skew show us that the majority of our data lies below the mean.

B. Quality / Correlation Metrics

We ran simple baseline calculations to determine how our data was distributed and the effect of our fault injection

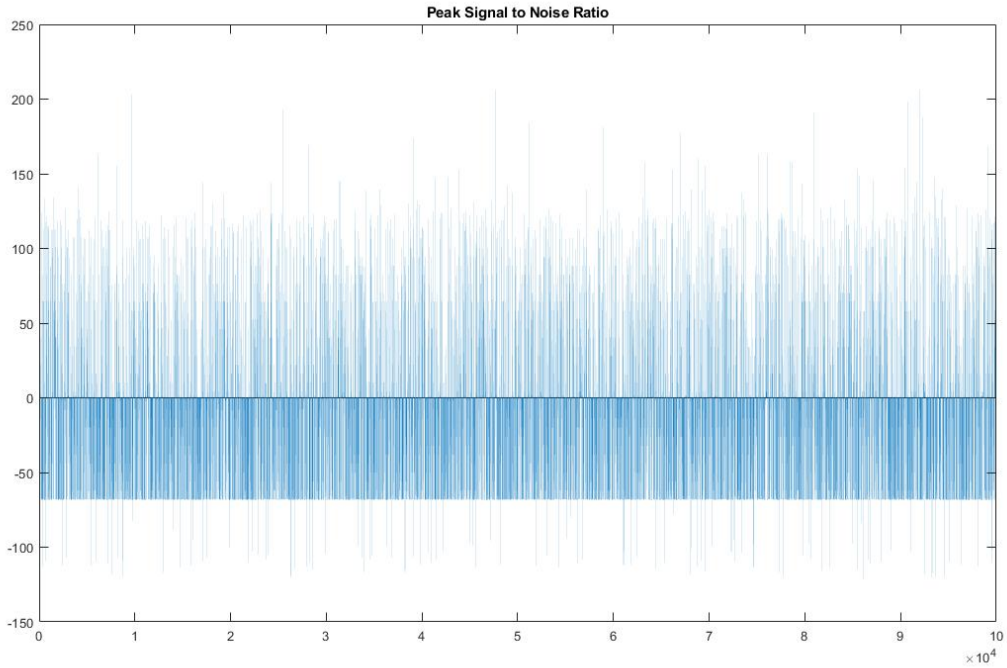

Fig. 2. PSNR for Floating Point Numbers

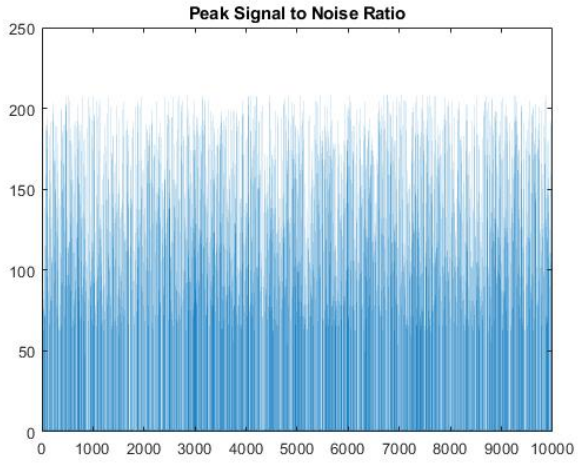

Fig. 3. Difference in PSNR for Posits

campaign on this distribution. In order to further analyze our data, we also wrote a C program to calculate several quality/correlation metrics that we could use to compare our entire data sets and gain insight on how posits and floats are effected differently by our fault injection campaign.

RMSE

- Calculation
  In order to calculate the RMSE, we used a predicted value from our original data set,P, and an observed value from our data set after our fault injection campaign, O. The equation can be seen below:
  $RMSE = \sqrt{\frac{\sum(P_i - O_i)^2}{N}}$
- Insight:
  The RMSE is a good way of representing the effect of our fault injection campaign across our data set as a whole. We are taking the difference between each value individually and summing them, giving us a big picture look at our overall differences. The smaller the value of our RMSE, the closer our data sets are in value. From our results, we can see that despite only changing a single value, our RMSE changes drastically in some

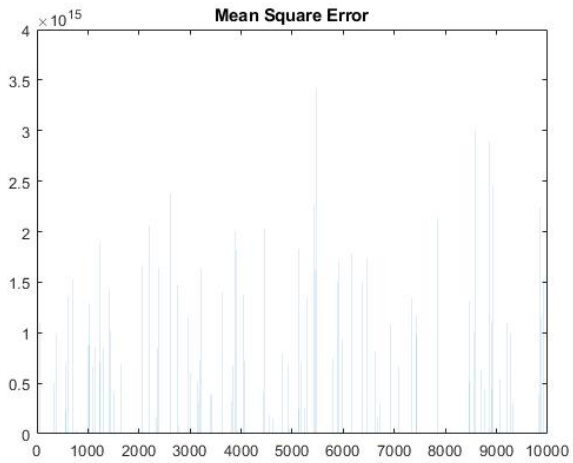

Fig. 4. Difference in Mean Square Error for Floating Point Numbers

fault injection campaigns, showing us that despite only changing a single value, our data can be changed a lot.

PSNR

- Calculation In order to calculate PSNR, we first needed to find our MSE, mean square error, and our maximum value. From there we were able to calculate our PSNR as below:

  $PSNR = 20log_{10}\sqrt{\frac{MAX}{MSE}}$

- Insight The PSNR or Peak signal to noise ratio is typically used to see the ratio between a signal and the noise of the signal. In our case, we are using it to determine the difference between our original data and our corrupted data. This gives us a look at how much our data is affected by each fault injection. From our testing we can see that the PSNR changes quite a lot between injections, leading us to the conclusion that flipping a single bit can still have a drastic effect on the data set as a whole.

$L_2 - norm$

Calculation In order to calculate our L_2-norm, we needed to find the square root of the square of each value in our original data, O, set compared to that of the fault injected data, P. The equation can be seen below:
$L_2 - norm = \sqrt{\sum(O_i - P_i)^2}$

Insight The L_2-norm shows us the difference between each value in our original data set and our fault injected data set. This one to one comparison gives us the distance between each element and from this we are able to see how much flipping one bit will effect each number in the data set.

$L - norm$

Calculation In order to calculate the L-norm we simply needed to find the max absolute value in each of out data sets.

Insight

This metric shows the absolute value of the largest value in our data set. This will show us the largest overall distance our values can be from zero.

MAE

- Calculation in order to calculate the Mean Absolute Error, we simply find the absolute difference between our two data sets. Original data set, O , and fault injected data set, P. This can be seen from the equation below:
  $MAE = \sqrt{\frac{\sum(O_i - P_i)}{N}}$
- Insight
  The MAE is a representation of the the average of the absolute value of the error. From this we can see how different our data sets are from an absolute perspective, rather than signed representation.

MSE

- Calculation
  In order to calculate the Mean Squared Error (MSE) we needed to find the sum of the square of the difference between our original data set and our fault injected data set. This can be seen in the equation below:
  $MSE = \frac{1}{n}\sum(O_i - P_i)^2$

- Insight
  By finding the MSE of our data sets, we were able to see the quality of our estimator. This shows us how different our data is when comparing the average across the data set as a whole.

SEM

- Calculation In order to calculate the Standard Error of the Mean (SEM), we needed to calculate the standard deviation, and divide it by the square root

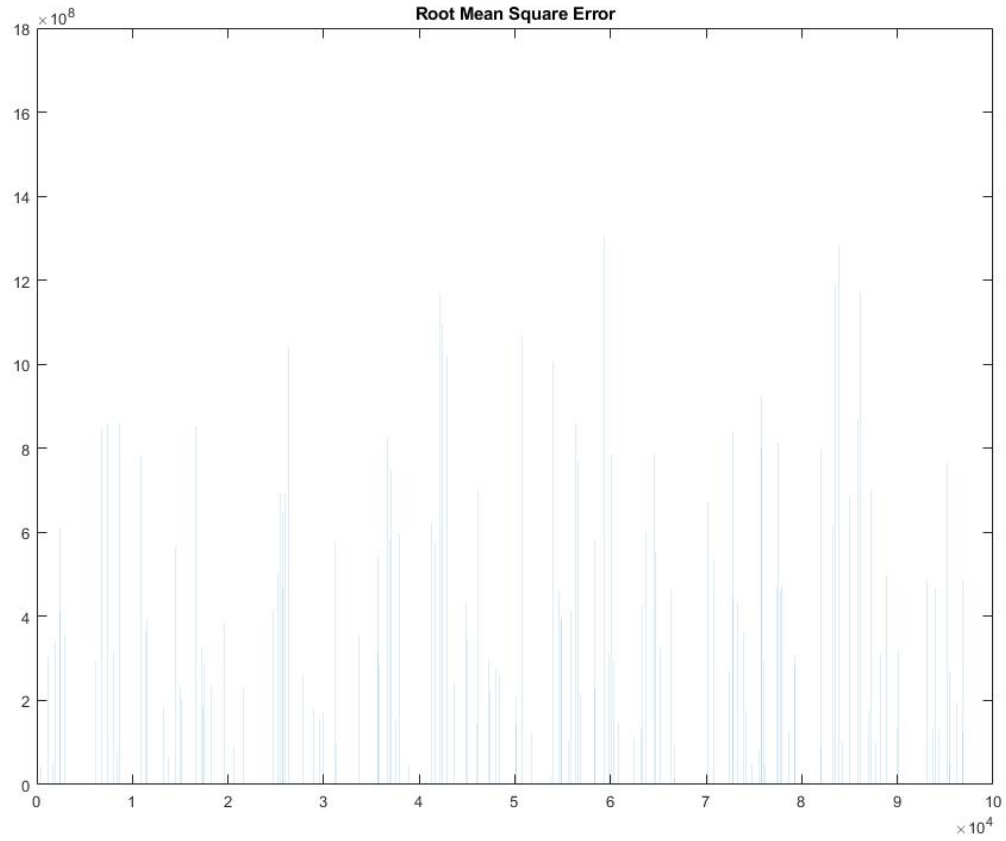

Fig. 5. RMSE for Floating Point Numbers

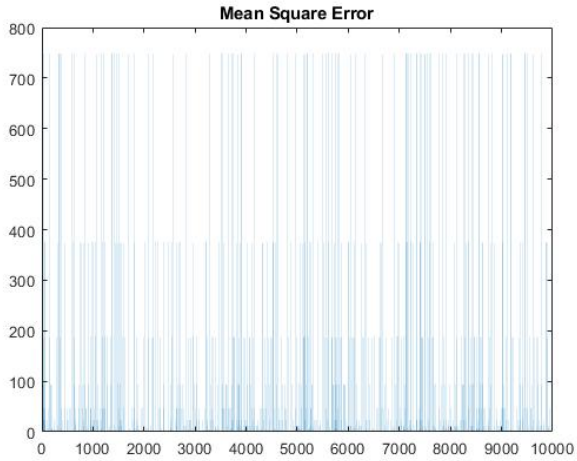

Fig. 6. Difference in Mean Square Error for Posits

of the sample size. This can be seen in the equation below.
$SEM = \frac{\sigma}{\sqrt{n}}$

- Insight The SEM helped to evaluate how far off some values were from the mean. This allowed for added robustness when analyzing how accurate our values were after the initial bit corruption.

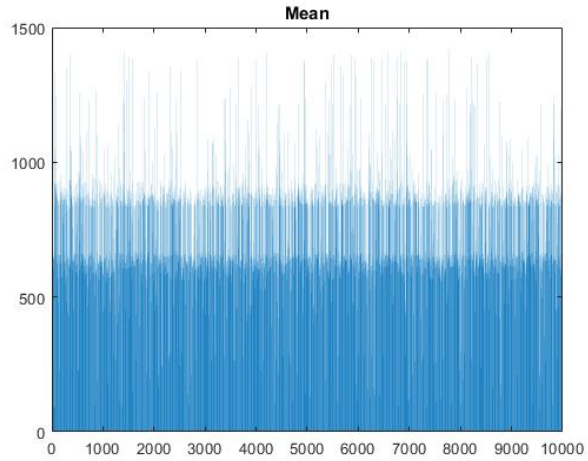

Fig. 7. Difference in Mean for Posits

C. Data Evaluation Methods

We went through many ideas for visualization and analysis of the data. We had considered using Paraview in order to create graphs that would show the spread of values for bit corruption. However, Paraview is very limited in its 2d graphing capabilities and was overall not the best fit. We decided Matlab would best fit our needs. Matlab is very efficient at Matrix math and manipulation allowing for a simpler implementation. The data files were formatted in a typical CSV format. CSV was chosen due to its simplicity of implementation and compatibility with other tools such as Matlab. Each test file had three lines representing calculations after fault injection, the differential pre and post injection, and the correlation metrics mentioned above. These values were then organized into their own matrices and graphed as seen in Figures 1-6.

In the Figures above, it is important to note that some of them only have 10,000 trials. This is due to the graphical representation of the data. It becomes unreadable with 100,000 points as the values become too close together to properly evaluate. Another reason was caused by extreme outliers in the data. For example, a value above 1e+32 was found when calculating the difference in mean. The next highest value was slightly above 1e+15 and left many other data points unseen. It was decided that this snippet was acceptable as it still provided an accurate depiction of the results while successfully visualizing the results.

D. Automation of Process

In order to thoroughly test the resiliency of floats and Posits, it is necessary to create a pipeline for efficiency. This started with editing an existing script utilized in a previous project. The script would run the main program by creating numerous jobs on the Palmetto cluster and outputting the results to individual test files. Each test runs independently of another so this was an ideal problem to solve with parallel jobs. After completion, the data was in CSV format in each test file. The level of automation created by our solution allowed for us to work out any issues we had left in this research.

E. Resiliency

Upon analysis of floats, some notable results were made. The most significant being the large difference that can occur upon bit corruption. Values for each statistic varied greatly and were very inconsistent. This can be seen by the graphs. This is likely due to little protection within a floating point number. Since floating point numbers are primarily fraction and exponent bits, one change can greatly impact the number. This is seen with the extreme outlier values as mentioned above.

Posits, in comparison, were much more consistent than floats. This is likely due to to their differing representation when compared to a floating point number and how each bit effects the dataset. The regime bits within a posit are the primary reasoning for this. These regime bits add an extra layer of security that allow the Posit to only vary by little. As seen in Figure 7, the mean only varies by up to 1500 compared to the 1e+31 as seen in floating point numbers. It is also interesting to note that Posits differ post corruption in tiers. These tiers are clearly seen in Figure 7. Most values seem to differ by a maximum of 600-700 from their pre-corrupted value. The next tier varies by 800 and the final tier varying by 1000-1500.

VI. Conclusions

In order to evaluate the resilience of Posits and floating point numbers, we conducted 100,000 trials on 5 different data sets. This created the Figures seen throughout. Based on these Figures, it can be concluded that posits are much more resilient than the IEEE-754 floating point standard. This seems to primarily be due to the regime

bits of a Posit. Floating point values have much room for variability due to only have exponent and fractional bits with no safe guard. Posits have this safeguard and it works very well as shown in the figures.

VII. Future Work

Even though numerous datasets were tested throughout our experiments, we would like to see more extreme examples used. This would primarily focus around data sets that have wide ranges of values and seem inconsistent.
In addition, we would like to see an expansion to SoftPosits similar to SoftPosit-Math to add more functionality with them so they can be easier utilized. As of yet, there are few math SoftPosit functions for the 32-bit posit. I believe adding more would increase the ability to use this software easier.

VIII. Feedback/Discussion

Throughout this project, we all had to escape our comfort zones to enter a realm we were unfamiliar with. A primary area we all enjoyed learning about was in automation by scripting. Parallel programming is extremely useful and our problem was a perfect one to solve with parallel computations. It taught us a more practical use of programming when compared to often theory heavy projects most of us completed in the past. Another area of learning was with fault injection as a whole. It is not a common thought to consider the repercussions of how results can changed based on a single bit changing from a 0 to a 1 or 1 to a 0. This line of thinking will be extremely helpful when considering software development in the future.
One of the many problems we had to solve was how best to analyze our data. This involved a process of weighing different options and evaluating them. Originally we had thought to use Matlab, then suggested and tried Paraview, then considered writing a C program, and eventually landed back with Matlab. The process really helped to teach us how to choose the best tool. There are so many different programming languages and programs all with pros and cons. Being able to do this for a project helped all to better our abilities.
Initially, when writing the version of our code that implemented SoftPosits, we attempted to do all calculations with posits and have minimal use of doubles and floats. However, because we were using the 32 bit posit, this proved difficult because not many math functions were implemented for the 32 bit SoftPosit. After multiple versions of our code, the final implementation only used SoftPosits for the fault injection.

IX. Related Work

References

[1] John L. Gustafson and Isaac Yonemoto. "Beating Floating Point at its Own Game: Posit Arithmetic." Web. 2017.

[2] Z. Li et al., "SpotSDC: Revealing the Silent Data Corruption Propagation in High-performance Computing Systems," in IEEE Transactions on Visualization and Computer Graphics.

[3] Volker Heydegger. M. Agosti et al. (Eds.): ECDL 2009, LNCS 5714, pp. 315–326, 2009. © Springer-Verlag Berlin Heidelberg 2009 "Just One Bit in a Million: On the Effects of Data Corruption in Files"

[4] Cerlane Leong. https://gitlab.com/cerlane/SoftPosit

[5] Cerlane Leong. https://gitlab.com/cerlane/softposit-math