DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2020

Comparing the Accuracy Between IEEE 754 and Posit

Using Matrix Multiplication

PONTUS KARLBERG

TONY LE

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Comparing the Accuracy Between IEEE 754 and Posit Using Matrix Multiplication

PONTUS KARLBERG, TONY LE

Bachelor in Computer Science
Date: June 8, 2020
Supervisor: Stefano Markidis
Examiner: Pawel Herman
School of Electrical Engineering and Computer Science
Swedish title: En jämförelse av noggrannheten mellan IEEE 754 och Posit med användning av matrismultiplikation

Abstract

The aim of this study was to compare the accuracy of the 32-bit IEEE 754 floating point numbers and 32-bit Posit numbers. In order to achieve this goal, two benchmarks were developed in C++. These benchmarks ran a series of matrix multiplications with IEEE 754 floats, posits and 64-bit doubles and computed an error for each by taking the average relative error of all elements in the result matrix. These relative errors were computed by comparing the element from each 32-bit float matrix and the corresponding element in the 64-bit double element matrix. By plotting the results it could be concluded that 32-bit posits produced results with a higher accuracy than 32-bit IEEE 754 floats for small values as elements. It was also clear from the results that 32-bit posits were greatly outperformed by 32-bit IEEE 754 floats for larger values.

Sammanfattning

Syftet med den här undersökningen var att jämföra noggrannheten hos 32-bitars IEEE 754 flyttal med 32-bitars Posit flyttal. För att uppnå det målet utvecklades två benchmarks i C++. Dessa benchmarks körde flera matrismultiplikationer med IEEE 754 flyttal, posits samt doubles och beräknade ett fel för varje genom att ta det genomsnittliga relativfelet av alla element i resultatmatrisen. Dessa relativfel beräknades genom att jämföra varje element från varje 32-bitars flyttalsmatris och det motsvarande elementet i 64-bitarsmatrisen. Genom att sammanställa resultaten i en graf kunde vi dra slutsatsen att 32-bitars posits producerade resultat med högre noggrannhet än 32-bitars IEEE 754 flyttal för små tal som element. Det var också tydligt från resultaten att 32-bitars posits var mycket sämre än 32-bitars IEEE 754 flyttal för större tal.

Contents

Chapter 1

Introduction

Computers are used for a large number of different purposes. Many of these uses involve calculations with real numbers. It isn't possible for a computer to represent all real numbers since some have infinitely many digits after the decimal point. Instead, floating-point numbers are used to represent real numbers as an approximation to support a trade-off between accuracy, range and speed. The contemporary technical standard floating-point format is IEEE 754 which was founded by The Institute of Electrical and Electronics Engineers (IEEE) in 1985 [1]. In the recent years however, some new promising standards have emerged, such as Posit, Bfloat and Flexpoint. Should a single standard be used for all purposes or might there be newer standards that are superior in some aspects worth using in specific cases?

1.1 Significance of this study

IEEE 754 is a relatively old standard even though it was revised in 2008 [2] and 2019 [3], and there could be room for improvement. In view of new formats emerging it's worth studying their strengths and weaknesses and contrast them with respect to the standard to determine whether or not the former's usage could be beneficial in some scenarios, and when as well as how their utilization could be justified.

In this study IEEE 754 will be compared to the Posit standard. Numbers from this standard are claimed to be able to exceed the dynamic range and accuracy of IEEE 754 utilizing the same number of bits [4]. This study is limited to mainly a comparison with respect to accuracy, which in this context specifically refers to how close an approximation is to an exact value.

Floating-point values are important in many areas of computer science and

in software as their existence makes computation faster with some trade-offs in accuracy and range. IEEE 754 might work fine for most implementations, with its balance of accuracy, range and speed. However what if there's a need to adjust the balance for a specific implementation and the accuracy requirement is higher? Another format could be found to be more accurate in certain scenarios, with little or no sacrifice of range and speed. If another format turns out to be more accurate, and faster, with greater range, then there should be even less reason for IEEE 754 to be the technical standard. There are cases where even a small deviation in accuracy is a matter of life or death. One case where a small lack of accuracy had extremely deadly consequences occurred on the 25th of February, 1991 when a small deviation in the calculation of time caused by the limited 24-bit precision floating points used prevented the interception of an incoming missile which resulted in the death of 28 U.S. soldiers in Dharan, Saudi Arabia [5].

1.2 Research Question

IEEE 754 is still being maintained even today, with the latest revision being from 2019, however determining possible areas of improvement and finding justification for a change which can move technology forward is important. There exist applications with higher need for accuracy, and where range as well as speed can be sacrificed to different extents to meet those requirements. To that end, the research question this study aims to answer is:

- *What differences in accuracy exist between the floating-point format IEEE 754 and Posit?*

1.3 Scope

The scope of this study is limited to mainly analyzing and comparing the accuracy of the 32-bit IEEE 754 and Posit representations.

1.4 Approach

A code for benchmarking the accuracy improvement or loss of Posit with respect to IEEE 754 was developed. The benchmark code generates and multiplies two matrices of varying sizes containing float values. The accuracy of the floating-point format used is to be assessed by measuring the relative error

between the resulting values and more exact values. The greater the error is, the less accurate the floating-point format is when used in calculations. The results of this study is evaluated based on how much the relative errors differ between matrix-matrix multiplications with both floating-point formats. The reason for using matrix multiplication as a benchmark is due to its common usage in many scientific fields, such as network theory, 3D computer graphics, and signal processing.

Chapter 2

Background

2.1 Floating point numbers

2.1.1 IEEE 754

As of 2019, IEEE summarizes the 754 standard with the following description [3]:

> "This standard specifies interchange and arithmetic formats and methods for binary and decimal floating-point arithmetic in computer programming environments. This standard specifies exception conditions and their default handling. An implementation of a floating-point system conforming to this standard may be realized entirely in software, entirely in hardware, or in any combination of software and hardware. For operations specified in the normative part of this standard, numerical results and exceptions are uniquely determined by the values of the input data, sequence of operations, and destination formats, all under user control."

The set of finite floating-point numbers is represented by the following integer parameters according to the IEEE 754 format [3]:

- b = the radix, 2 (binary) or 10 (decimal)
- p = the number of digits in the significand (precision)
- emin = the minimum exponent e, which shall be $1-emax$ for all formats
- emax = the maximum exponent e

32-bit floating point numbers following this standard will in this report be referred to as *float32* while *IEEE 754 float* and *float* will be used when the number of bits isn't specified.

2.1.2 Posit

Unums (Universal numbers) is a way to represent floating-point numbers created by John Gustafson. Posits are a more hardware friendly version of these. The bits of a posit is divided into the sign bit, regime bits, exponent bits and fraction bits. The regime and exponent bits serve the same purpose as the exponent bits in a IEEE 754 float. The number of bits used for the exponent and fraction is not static as is the case for IEEE 754 floats. Depending on the number of bits allocated for the exponent and fraction a posit can either match or exceed the dynamic range or the precision of a standard IEEE 754 float. Gustafson also claims that the hardware to support posits is simpler than that of IEEE 754 floats and that the reduced power and silicon used means that the number of posit operations per second can be significantly higher than the number of floating point operations per second in classical hardware [4].

In this report, the 32-bit variant of these numbers will be referred to as *posit32* while *posit* will be used to refer to these numbers when the number of bits isn't specified.

2.2 Related work

The accuracy improvement granted by using posits has previously been studied by Chien, Peng, and Markidis [6]. They assessed the precision optimization achieved by posits in high performance computing scientific applications by using the NAS Parallel Benchmark (NPB) suite developed by NASA. The results indicated that all tested kernels in their Posit NPB suite achieved higher precision, ranging from 0.6 to 1.4 decimal digit, compared to the IEEE 754 float baseline, but with an overhead of the software implementation of the posit encoding being 4x-19x that of the IEEE 754 hardware implementation. In our study, we're not assessing the overhead of either format, however their results in regards to precision are interesting.

Gustafson and Yonemoto [4] have also studied the benefits of using posits in comparison to IEEE 754 floats. They conclude that posits have higher accuracy, larger dynamic range and better closure. They claim that posits can produce better answers than IEEE 754 floats with the same number of bits,

and also that they can produce an equally good answer with fewer bits than an IEEE 754 float.

Dinechin et al. [7] has published an article in which the properties of Posit are studied. In their conclusions they mention that there are some clear use cases for posits, but there are also clear situations where they're worse than floating points. They also add that when posits are better than floats of the same size, they provide one or two extra digits of accuracy, but when they are worse, the degradation in accuracy can be arbitrarily large.

These papers may be used as supplements to this study for comparison of results and discussion.

Chapter 3

Methodology

This work aims to assess the differences in accuracy between float32 and posit32. The metric for accuracy is defined as the relative difference between the approximated value and the exact value. 64-bit precision doubles are chosen to correspond to exact values, due to possessing a higher precision and accuracy compared to the 32-bit precision formats.
The relative error is thus: $E_R = |\frac{v_a - v_e}{v_e} \cdot 100\%|$, where v_a is the value in the 32-bit precision formats and v_e is the corresponding value in the 64-bit precision format.

The two main parameters that were identified and tested for their effect on the error were the size of the elements and the size of the matrices. Thus, two different benchmarks were written in C++ to achieve the goal. Each benchmark involves repeated square matrix multiplication with the different floating point formats. A square matrix with 64-bit precision doubles elements is first generated using a uniform distribution and a given bound. The two other matrices are then created by copying and converting the double precision elements to the corresponding floating point formats. These steps are performed twice to generate a pair of matrices of each type. Each matrix pair is then multiplied together, using a naive multiplication algorithm, which results in three new result matrices.

To determine the accuracy of each format, the relative errors E_R of each element in the float32 and posit32 result matrices are measured. The average of all the elements' relative error is the relative error of the matrix, and reflects the accuracy of the floating point format. The errors of the resulting matrices were stored and used to generate graphs with Gnuplot.

The benchmarks were run on a Macbook Pro with a Intel Core i5-7267U CPU with 16GB RAM and a 250GB SSD, running macOS Mojave Version

10.14.6. The compiler used was G++ 4.2.1, with the C++ version of the latest SoftPosit library from the main development branch.

3.1 Benchmark 1

In the first benchmark, the dimensions of the multiplied matrices changes for each iteration. The benchmark takes four parameters: *startingDimension*, *fixedBound*, i*terations* and *growth*. The parameter *fixedBound* determines the upper bound for the size of the numbers inside each matrix while 0 is the lower bound. *startingDimension* determines the size of the matrix in the first iteration of the benchmark while *growth* and *iterations* determines how much each matrix grows per iteration and how many iterations that will be computed.

3.2 Benchmark 2

In the second benchmark, the upper bound of the generated numbers increases for each iteration. The benchmark takes four required parameters *fixedDimension*, *startingUpperBound*, *iterations* and *growth*, but also an optional parameter *rangeSize*. The parameter *fixedDimension* determines the size of each matrix. The parameters *startingUpperBound*, *growth* and *iterations* determines the upper bound for the size of the numbers inside each matrix, how much the bound grows by for each iteration and how many iterations are computed. If the optional parameter *rangeSize* is supplied, then the lower bound for the numbers in each matrix is given by $upperBound - rangeSize$ where *upperBound* is the current upper bound for that iteration. Otherwise the lower bound is 0.

Chapter 4

Results

4.1 Benchmark 1

The result from running Benchmark 1 with numbers in the range (0,1) shows that float32 lose relatively more accuracy than posit32 as the size of the matrices increases (see Figure 4.1).

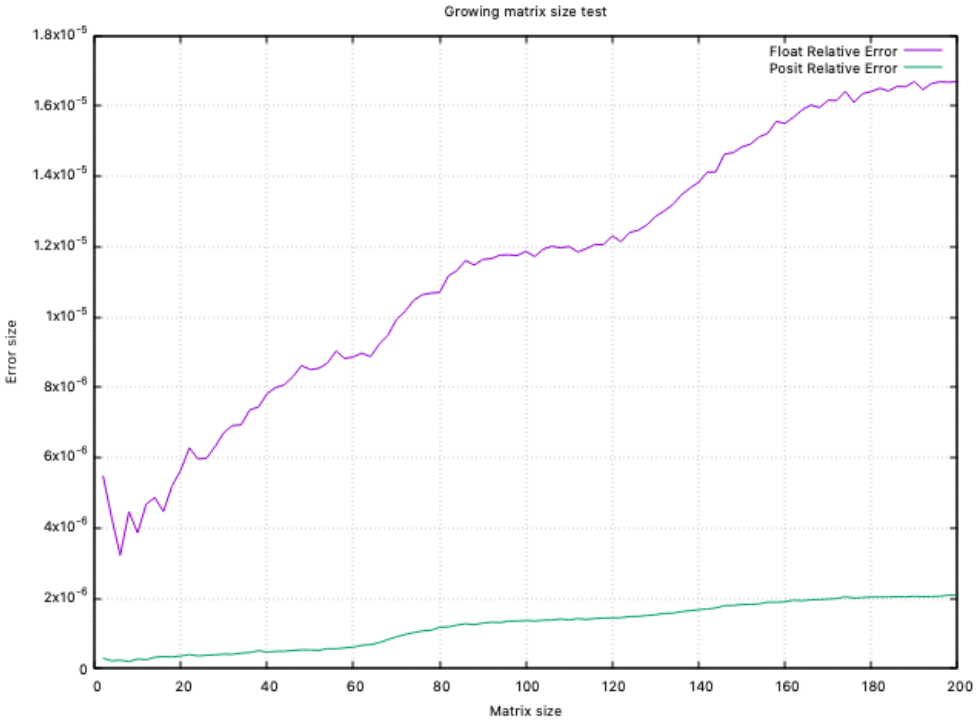

Figure 4.1: The result from Benchmark 1 with matrix dimensions starting from 2 and growing by 2 for 100 iterations. Element numbers are bounded between 0 and 1

Figure 4.2 illustrates the result when the benchmark is run with elements in the range (0,100). When the size of the matrices is small there's a rather small difference in accuracy between the two formats, but as the matrix sizes increase the gap widens, with posit32 achieving better accuracy until the gap closes after a certain matrix size. After the gap closes both relative errors are approximately the same.

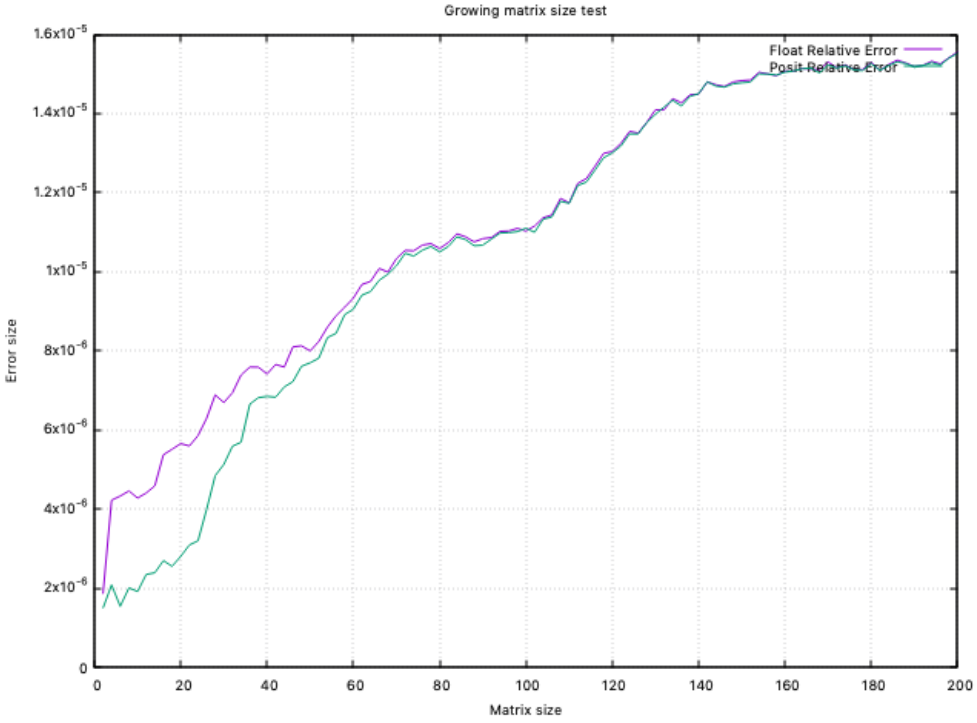

Figure 4.2: The result from Benchmark 1 with matrix dimensions starting from 2 and growing by 2 for 100 iterations. Element numbers are bounded between 0 and 100

When the benchmark is run with matrix elements ranging from 0 to 1000, as the matrices increase in size the difference in accuracy also grows. But this time it is the relative error for posit32 that grows at a higher rate. These results, as shown in Figure 4.3, demonstrates that posit32 has worse accuracy than float32 when numbers from a sufficiently large range are used.

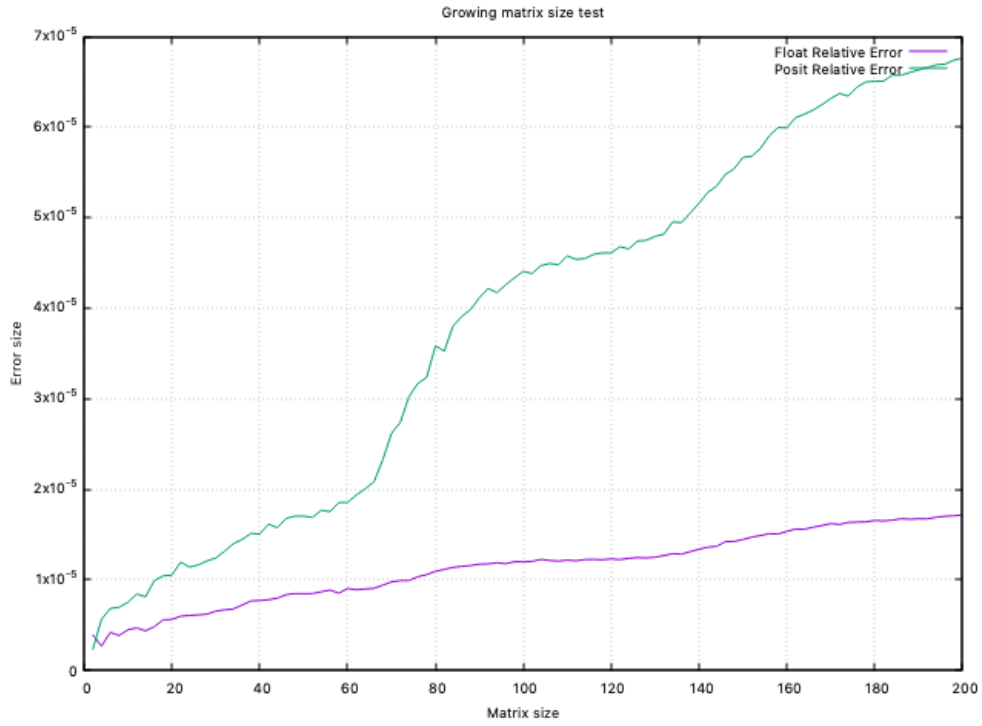

Figure 4.3: The result from Benchmark 1 with matrix dimensions starting from 2 and growing by 2 for 100 iterations. Element numbers are bounded between 0 and 1000

4.2 Benchmark 2

As seen in Figure 4.4, when 4x4 matrices are multiplied using values bounded between (0,10) and (0,2000) the resulting graph has a very spiky characteristic. Posit32 starts out being more accurate compared to float32 at the lower bounds, until they're roughly even between 400-1600, and then posit32 becomes increasingly more inaccurate in comparison.

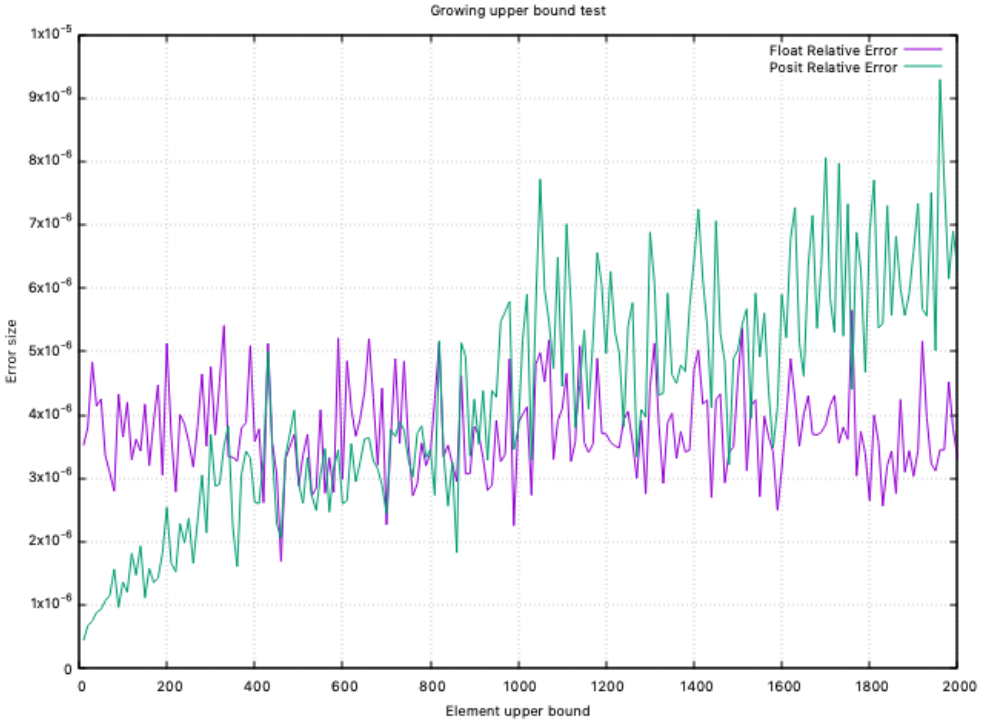

Figure 4.4: The result from Benchmark 2 with matrix dimension 4. Element numbers start being bounded between 0 and 10, and the upper bound increases by 10 for 200 iterations

The results from when the same benchmark is run with matrices of size 16 can be seen in Figure 4.5. The results are very similar to when 4x4 matrices were used, with the exception that the graph's spiky characteristic is less prominent. It also seems that the posit32 error surpasses the float32 error earlier.

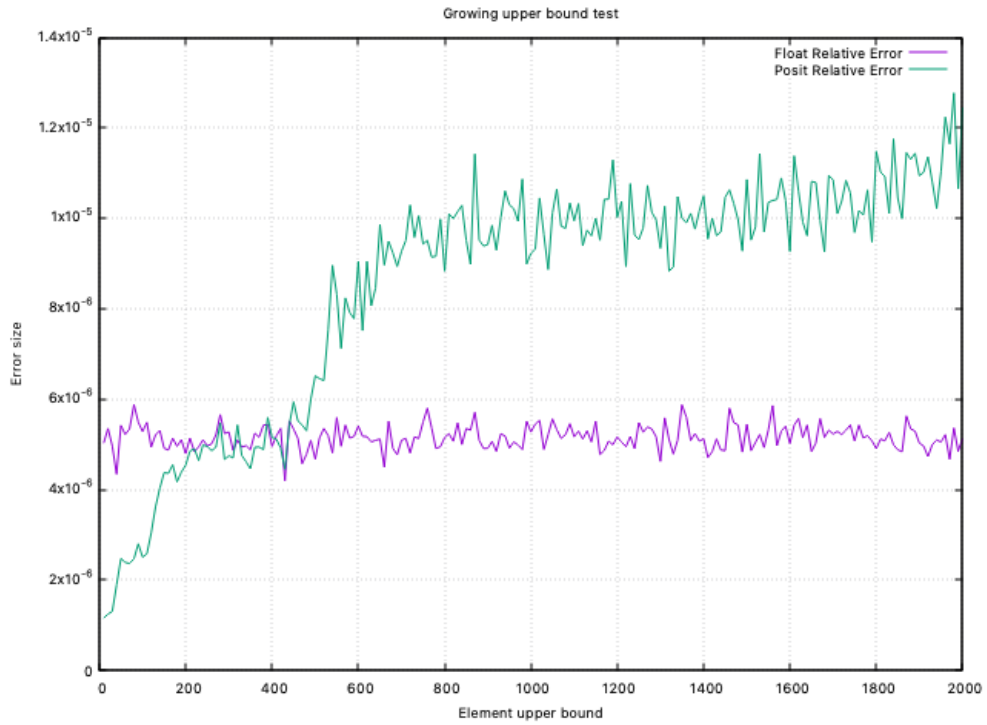

Figure 4.5: The result from Benchmark 2 with matrix dimension 16. Element numbers start being bounded between 0 and 10, and the upper bound increases by 10 for 200 iterations

When the matrix sizes increase to 64, the spiky characteristic is almost gone as illustrated in Figure 4.6. It is also apparent that the error for posit32 seems to rapidly increase in short bursts when the upper bound is around 50, 200 and 1000.

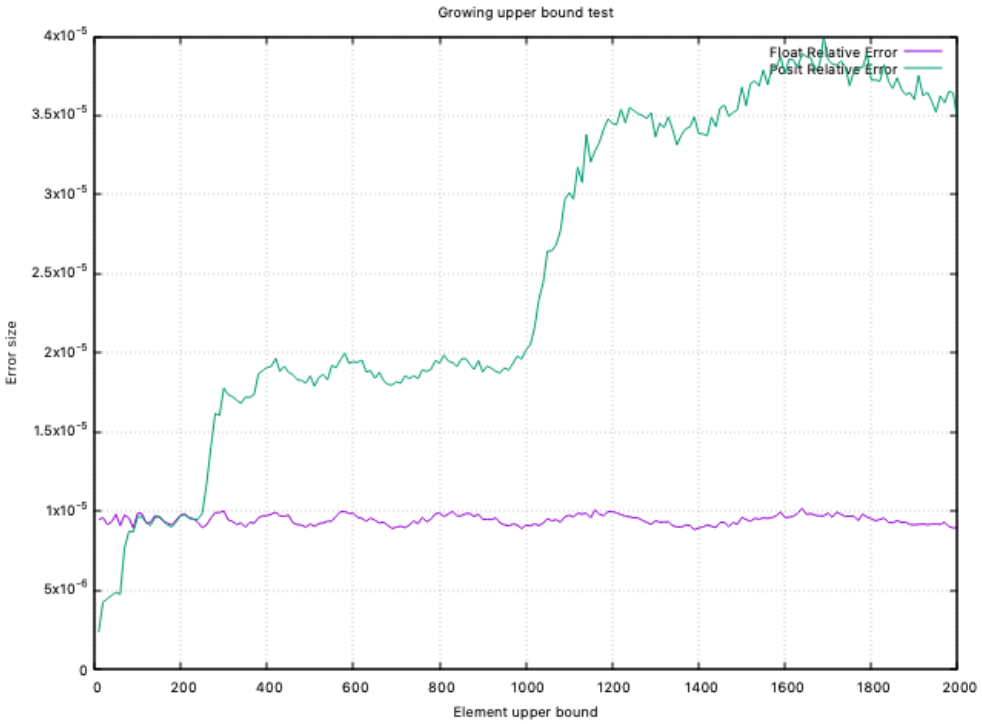

Figure 4.6: The result from Benchmark 2 with matrix dimension 64. Element numbers start being bounded between 0 and 10, and the upper bound increases by 10 for 200 iterations

Running the benchmark for 4x4 matrices and letting the elements' upper bound grow to 10000 generates results were the accuracy of the float32 matrices is superior to that of the posit32 matrices (see Figure 4.7).

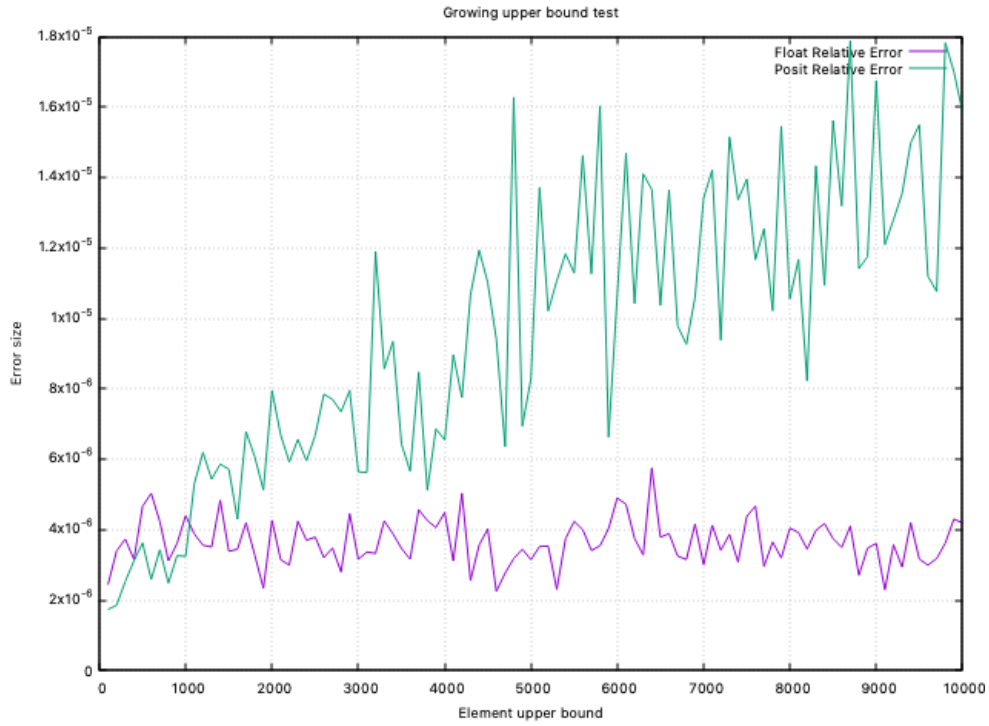

Figure 4.7: The result from Benchmark 2 with matrix dimension 4. Element numbers start being bounded between 0 and 10, and the upper bound increases by 100 for 200 iterations

The graph that can be seen in Figure 4.8 does show that posit32's accuracy still does degrade, although in arbitrary extents, as values get larger, even with a small interval between the lower and upper bound of the elements.

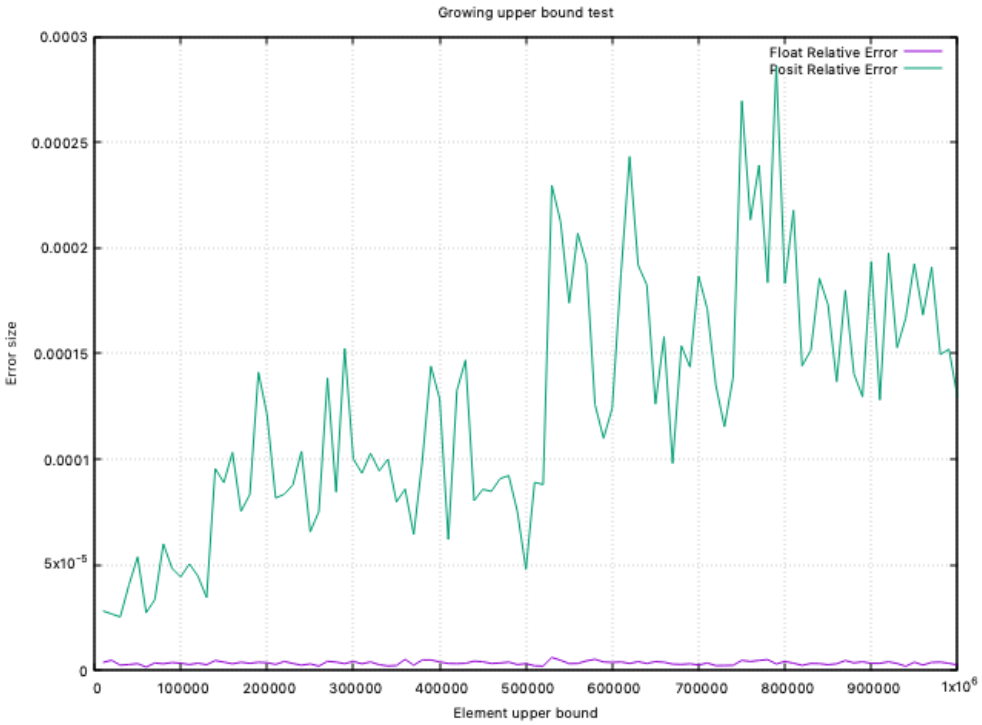

Figure 4.8: The result from Benchmark 2 with matrix dimension 4. Element numbers start being bounded between 9990 and 10000, and the lower AND upper bound increases by 100 for 200 iterations

Chapter 5

Discussion

5.1 Result analysis

The results of tests with the first benchmark indicate that posit32 does bring a visible improvement relative to float32 for smaller values ranging from (0,1) when used in matrix multiplications, as can be seen in Figure 4.1. As the upper bound increases so that the values range from (0,100), posit32 still shows some improvement, until the dimensions of the matrices grow to a certain point, where the difference in accuracy between the two formats starts becoming negligible. Once the upper bound of the values increase beyond that point, while float32 grows more errant, it's relatively less compared to posit32 which grows significantly in error. Tests with the second benchmark follows this up with similar characteristics, and it can clearly be seen that as the element upper bound increases, while float32's accuracy barely sees any significant change, posit32's degradation of accuracy gets noticeably larger. Comparing the two formats, posit32 seems to possess higher accuracy when dealing with smaller floating point values, but it's significantly inferior in dealing with larger values.

To exclude the possibility that posit32 isn't bad at specifically operations between small och large values, and confirm that it's just bad at dealing with large values, this special case was tested for in Figure 4.8. The graph still shows that posit32 steadily increases in error as the values grow, although in arbitrary amounts at times.

It's interesting how, in Figure 4.6, posit32's relative error seems to drastically increase at certain points and at some points its accuracy is relatively stable.

Overall, the accuracy improvement achieved by using posits with smaller values, compared to using floats, is comparatively less than the immense ac-

curacy degradation that we see as the values get larger and larger. The huge loss of accuracy with larger values was somewhat surprising to us, as we had high expectations from Posit, which were partially based on what Chien, Peng, and Markidis [6] and Gustafson and Yonemoto [4] concluded in their papers. What we see in our results is instead more similar to the conclusion drawn by Dinechin et al. [7], which reinforces the idea that Posit can not fully replace IEEE 754.

5.2 Method analysis

The reasoning for using matrix-matrix multiplication was due to its common usage in many scientific applications. We identified that there were going to be two main factors that would have a big impact on our results, namely the size of the matrices that we're multiplying, and the range of values that we're going to be using as the elements of the matrices. The distribution of the pseudo-random float values could also affect and skew the results to some degree, but by using a uniform distribution and averaging the relative error of all the elements in the matrices, we minimized this factor. However, the latter measure is less effective the smaller the matrices are, which might explain why some of the graphs that were generated by the second benchmarks, such as Figure 4.4 exhibit spiky characteristics, seeing as they're done with a rather small, fixed matrix size. When tests are done with greater matrix sizes in the second benchmark we noticed that the spiky characteristics are less prominent, such as in Figure 4.5 and Figure 4.6.

We chose to measure the relative error of the two floating point formats relative to the 64-bit precision double format, which still isn't exact, and there are options with higher accuracy, such as the 128-bit precision quad. For our purposes the 64-bit format should be sufficiently more accurate compared to the 32-bit formats, that it shouldn't affect the results substantially.

5.3 Future work

In addition to measuring accuracy, there are other important factors that can be benchmarked to evaluate whether Posit is superior to IEEE 754 in some aspects. Although it's pretty apparent that Posit suffers from rather extreme accuracy degradation when dealing with large values, perhaps the format could have an advantage in speed, or range, and those improvements are significant

enough for it to be a better contender in some scenarios. This could be interesting to test in future works.

Chapter 6

Conclusions

This thesis aimed to answer *What differences in accuracy exist between the floating-point format IEEE 754 and Posit?* To achieve this we measured the relative errors that result from matrix-matrix multiplications using 32-bit posits and 32-bit IEEE 754 floats in comparison to 64-bit doubles, and how the results vary as the dimensions of the matrices, as well as the range of the elements change. The results of our study show that posit has a slight superiority in accuracy when dealing with very small values, but as they increase the gap between the two formats narrows. When dealing with larger values however, posit's accuracy degrades wildly, whereas float exhibits comparatively minimal change in accuracy. These observations lead us to conclude that when dealing with greater values, floats have far superior accuracy compared to posits. Posit should not replace IEEE 754 as the technical standard, and scenarios where they may be preferred over IEEE 754 for higher accuracy requirements are limited to only when dealing with very small floating point values.

Bibliography

[1] "IEEE Standard for Binary Floating-Point Arithmetic". In: *ANSI/IEEE Std 754-1985* (1985), pp. 1–20.

[2] "IEEE Standard for Floating-Point Arithmetic". In: *IEEE Std 754-2008* (2008), pp. 1–70.

[3] "IEEE Standard for Floating-Point Arithmetic". In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), pp. 1–84.

[4] John Gustafson and Isaac Yonemoto. "Beating Floating Point at its Own Game: Posit Arithmetic". In: *Supercomputing Frontiers and Innovations* 4 (Jan. 2017), pp. 71–86. DOI: 10.14529/jsfi170206. URL: http://www.johngustafson.net/pdfs/BeatingFloatingPoint.pdf.

[5] U.S. Government Accountability Office. *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia*. Feb. 1992. URL: https://www.gao.gov/products/imtec-92-26.

[6] Steven Wei Der Chien, Ivy Bo Peng, and Stefano Markidis. "Posit NPB: Assessing the Precision Improvement in HPC Scientific Applications". In: *CoRR* abs/1907.05917 (2019). arXiv: 1907.05917. URL: http://arxiv.org/abs/1907.05917.

[7] Florent de Dinechin et al. "Posits: the good, the bad and the ugly". In: *CoNGA 2019 - Conference on Next-Generation Arithmetic*. Singapore, Singapore: ACM Press, Mar. 2019, pp. 1–10. DOI: 10.1145/3316279.3316285. URL: https://hal.inria.fr/hal-01959581.

TRITA-EECS-EX-2020:350