UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

PhD Thesis Summary

Number representation systems in computer engineering

Ștefan-Dan Ciocîrlan

Thesis advisor:

Prof. PhD. Eng. Răzvan-Victor Rughiniș

BUCHAREST
2023

CONTENTS

ABSTRACT

Mathematical computations are one of the fundamental blocks of humanity. Computers use number representation systems to emulate mathematical sets and operations. Currently, digital computers can only represent finite subsets of the infinite sets presented in mathematics, and this limitation produces errors in computations. Every computer software implementation uses a particular set of numbers. The obvious thing is to have a specific number representation system which can produce the best accuracy and efficiency for the application. The current solution is to use a system which is good enough. This lasted for decades until new number representation systems were proposed, and significant research was done to see the advantages and disadvantages compared to the current solution. The discussion might go again into a "dark age" for a few decades until a new opponent arises. The problem approached in this thesis is how to keep the discussion about number representation systems alive and engaging. The proposed solution reduces the discussion's entry-level and effort by creating a better research infrastructure. The research infrastructure contains a software library and a hardware generator library. The proposed software library reduces the time and effort of proposing a new number representation system to implementing the binary representation and the number constraints. The hardware generator library improves the path from an idea to multiple hardware prototype units. The infrastructure is used for proposing three new tapered floating point number representation systems: MorrisHEB, MorrisBiasHEB, and MorrisUnaryHEB. MorrisBiasHEB combines the benefits of the classic floating points and the tapered floating points, and has the first or second best results on the literature benchmarks. Its hardware units are one of the fastest, most resource efficient, and most energy efficient. The Kulisch accumulator unit for 16-bit MorrisBiasHEB has 1.81 times the maximum frequency and uses 23.98% fewer LUTs than the second-best in the respective category. MorrisUnaryHEB is only defined by its $size$ and has the highest percentage of exact results for addition (37.6%), the most crowded 'golden zone', better decimal accuracy on unary operations, and higher dynamic range. In addition to these two libraries, the thesis offers a statistical methods library (14 statistical methods), four scientific computing benchmarks, two low-precision machine learning frameworks and a complete hardware-software system specific to a number representation system. The statistical methods show better performance under tapered floating point systems. The 16-bit Posit offers similar results to its 32-bit counterparts. On the other hand, 16-bit IEEE754 does not offer valid results in 6 methods. The replacement of IEEE754 in scientific computing is unnecessary, even if Posit has at least one more correct decimal on N-Body simulation. The low-precision machine learning frameworks reduce the size of a deep neural network with 66.78%, 75%, and 76.78% with an accuracy degradation lower than 2% by using different number representation systems. The complete hardware-software 16-bit Posit ($es = 1$) offers 18% speedup for convolutional neural network, similar accuracy, and uses fewer hardware resources than 32-bit IEEE754.

Keywords: Numerical Analysis, Number Representation Systems, Computer Architecture, IEEE754, Posit, Scientific Computing, Artificial Intelligence, Statistical Methods

1 INTRODUCTION

$1 + 1 = 2$ might seem a simple, funny and innocent equation, but in some ways is one of the fundamental blocks of humanity. As a first argument, the entire human mathematical education is based on it. For most humans, it is the first assumption/operation/equation they learn. Next, all of the following mathematical concepts are explained from it. For example, the following simple addition $1 + 2 = 3$ is explained as $1 + 1 + 1 = 3$. The count of ones gives the answer. With time, these operations became unconscious, and the number value is separated from the count of ones. Another example is the multiplication presented as multiple additions. From transitivity, all of the future mathematical concepts based on these operations are also based on $1 + 1 = 2$. If the mathematical concepts are gathered in a tree structure based on their dependence, then $1 + 1 = 2$ is a root node or at least one of the closest nodes to the root. The second argument is that most of the current human technologies are based on or created with the help of mathematical concepts. In schools, there is a need for proficiency in mathematics to learn more complex science subjects. It is possible to explain physics without mathematical concepts. Neither less, It can not be argued that for some complex aspects of physics, the mathematical way is the easiest. A mind game for the reader is proposed in explaining the electricity (AC/DC [1]) without the mathematical concepts (without $1 + 1 = 2$). At the time of the writing of this thesis, artificial intelligence is starting to become common and familiar to people. Most artificial intelligence models are, in fact, mathematical constructions. The other important aspect (some consider even more important than the model) of artificial intelligence is the data used for training. This data is usually transformed and stored in a numerical form. Similar to artificial intelligence, there are concepts like digital signal processing, scientific computing, statistics, and economic models, which were and are a part of the current human technologies based on mathematical concepts. They also had similar development to artificial intelligence. Even if artificial intelligence will be replaced as the dominant emerging/used technology, history trains us to predict with a high probability that the replacing technology will be based on $2 = 1 + 1$.

A number representation system is a way how the concept of a number value is represented and how the basic operations work on the numbers. Simply, it can be how the equation $1 + 1 = 2$ is written and understood. Currently, the human number representation system is decimal. There are ten digits (0 to 9) used, and the result of the addition of two digits that has a result higher than 9 adds an extra digit to the left. The left-most digit is also considered the most significant digit. Another system used in the past is the roman number representation system in which the operation $1 + 1 = 2$ was $I + I = II$. For both, the operation inside the human mind can be similar or different. The readers can think about the way multiplication

[1] not the rock band

operation has evolved for them in their minds. In the beginning, multiple additions were used, but in time, the operation improved. It can also have different paths for improvement. The readers are invited to look at the difference between the Japanese multiplication method and the Arabic multiplication method. Another difference might be the way the human mind project the number values. These are the basic concepts of a number representation system:

- the way symbols represent the numbers has two processes: how the symbols are transformed in number values (decoding) and how the numbers are transformed in symbols (encoding).
- the way the operations for the numbers are done.

Computers are human creations to help with mathematical computation. In the development of computers, different number representation systems were selected. These systems are binary and finite. Different from the human mind, their limits are known. These limits create errors. Somehow the myth that computers are perfect and exact in computations was created. This myth is false. Number representation systems are the illusions which keep the myth alive. Neither less, with the increase in the number of computations and the trust of humans in computer mathematical computations, the illusion starts to break. To overcome this, new number representation systems were proposed [10, 11]. This thesis started as a research to find the domains where the Posit number representation system [11] can improve the illusion. Statistic methods, scientific computing algorithms, digital signal processing, artificial intelligence and hardware implementation were evaluated. On this journey, the author proposed three new number representation systems and tested them on the basic operations against the existing number representation systems. At that moment in time, the problem was still the errors that might break the illusion, and the research questions were:

- Which is the best number representation system for general computation?
- Are the number representation systems dependable on the specific application?
- Which is the most energy-efficient number representation system?
- What is the trade-off of speed, accuracy, and energy consumption for a number representation system?

These questions are answered through the chapters of this thesis, but this was not the end of the journey or the problem approached by this thesis. The problem is that the discussion about number representation systems stopped, and humanity accepted the "good enough" solution. Given the implication of a number representation system for computer mathematical computations, this must be avoided in the future. The work in this thesis, in addition to evaluating the number representation systems implications in multiple domains like statistics, artificial intelligence, scientific computing, and computer architecture, creates a research infrastructure which makes it easy to keep alive the discussion about number representation systems. Two libraries are implemented:

- A software library whose scope is to reduce the effort of proposing a number representation system to define the encoding/decoding and the limits of the numbers it can represent. The second objective of the library is to make effortless the benchmarks on the proposed number representation systems. This way, researchers and engineers can test their idea of a number representation system without going through the implementation of tedious benchmarks. They also get their results in contrast with the existing number representation systems.
- A hardware generator library whose scope is to generate multiple functional units (FPUs, KAUs, Binary/Unary Units, accelerators) with efforts similar to the software library (encoding/decoding and limits). The benchmarks and SoC integration do not need any effort from the developer. This library is complementary to the software library proposed. It offers a fast pipeline for a hardware prototype of a new proposed number representation system.

The contributions of this thesis are:

- The proposed software library can reduce the effort of proposing a new number representation system and benchmark multiple number representation systems. It contains 14 generic number representation systems.
- Three new number representation systems by introducing the concept of the hidden exponent bit.
- Analysis of 14 statistical methods under multiple number representation systems regarding accuracy and storage size.
- Four benchmarks in scientific computing for testing different number representation systems.
- Two low-precision machine learning frameworks for reducing the deep neural network size but keeping the accuracy within a threshold.
- The proposed hardware generator library generates functional units (FPUs, KAUs, Binary/Unary operations modules) for number representation systems. It also reduces the effort of having a working hardware prototype to implementing only the encoding and decoding modules.
- The proposed General Floating-Point Unit (GFPU) work simultaneously with different number representation systems.
- Analysis of an entire hardware-software system for a number representation system on three-level benchmarks.
- All of the above libraries, frameworks and benchmarks work effortlessly for any new number representation system added to the proposed software and hardware generator libraries.

2 THE DEVELOPMENT OF THE NRSS SOFTWARE LIBRARY (NRS-SL)

2.1 Hidden exponent bit tapered floating-point

In this section, the three new representations based on Morris tapered floating-point with a hidden exponent bit are introduced.

2.1.1 MorrisHEB(size, g, r)

The tapered floating-point introduced by Morris in [16] seems a good concept. Its utilization was shown under the posit system proposed by Gustafson [11]. The major problem is the multiple ways of representing the same number. A solution for this is in borrowing the concept of hidden bit from mantissa. The g field not only represents the G value which dictates the exponent size but also the position of the most significant bit set in the exponent. If the value of the exponent size is kept as $G+1$, then the minimum absolute value of the exponent is 2 when $G=0$. The exponent value is *exponent* = *exponent sign*$\times((1 \ll es)+$*binaryExponent*$)$. There is a need for having zero as exponent value. A solution for this is to change the formula for exponent size to $es = G-1$. The exponent is now:

$$\text{exponent} = \begin{cases} (-1)^{\text{exponent sign}} \times (2^{es} + \text{binaryExponent}), & es \neq -1 \\ 0, & es = -1. \end{cases} \tag{1}$$

This NRS is called MorrisHEB(size, g, r).

The next formula is used for computing the value of all the three new NRSs binary representations presented in this section:

$$\text{value} = \begin{cases} 0, & \text{all bits 0} \\ \text{NR}, & \text{first bit 1 and the rest 0s} \\ (-1)^{\text{sign}} \times 2^{\text{exponent}} \times (1 + \frac{\text{f}}{2^{\text{fs}}}), & \text{otherwise} \end{cases} \tag{2}$$

The differences are in the way es and $exponent$ are computed. MorrisHEB(size, g, r) underflows to 0, overflows to NR, and uses TaperedFloatingPoint(size) for implementing the operations.

The binary representation starts with the sign bit. The next g bits represent the G value in natural base 2 format. The exponent sign bit follows the g field. The next e bits ($es = G-1$)

or the next remaining bits (whichever is smaller) represent the binary exponent value in natural base 2 format. If es is greater than the count of remaining bits, the remaining bits represent the most significant bits of the binary exponent value. The remaining least significant bits of the binary exponent value will be considered 0. After taking the exponent bits, the remaining bits are fraction bits and their count represents the fraction size. In summary, the binary format is:

$$s_f G_{\text{g-1}} G_{\text{g-2}} \ldots G_0 s_e e_{\text{es-1}} e_{\text{es-2}} \ldots e_0 f_{\text{fs-1}} f_{\text{fs-2}} \ldots f_0 \tag{3}$$

2.1.2 MorrisBiasHEB(size, g, r)

One might argue that the problem of multiple representations is still not solved because even the exponent may have multiple values (for $es = -1$ the exponent sign does not matter). The problem stems from having a bit dedicated to the exponent sign. This is already solved in IEEE754 by using a bias value. A bias value g is proposed. The exponent sign is the sign of G and the exponent size is $es = |G| - 1$. Another issue with Morris and MorrisHEB representations is that they do not have an order in binary form. A solution for this is to have the bits of the exponent negated when G is negative. This makes it easy to implement a hardware compare unit. The NRS with these features is called MorrisBiasHEB(size, g, r), where the exponent is:

$$\text{exponent} = \begin{cases} \text{signum(G)} \times (2^{es} + \text{binaryExponent}), & es \neq -1 \\ 0, & es = -1. \end{cases} \tag{4}$$

MorrisBiasHEB(size, g, r) underflows to 0, overflows to NR, and uses TaperedFloating-Point(size) for implementing the operations. The binary representation starts with the sign bit. The next g bits represent the G value in bias format with $\textit{bias} = 2^{g-1} - 1$. This means that $G = \textit{binary G} - \textit{bias}$. The next es bits ($es = |G| - 1$) or the next remaining bits (whichever is smaller) represent the exponent in natural base 2 format, if the signum(G) is 1. Otherwise, they need to be negated an the results is the binary exponent value. If es is grater than the number of the remaining bits, the remaining bits represent the most significant bits of the binary exponent value. The remaining least significant bits of the binary exponent value are considered 0. After taking the exponent bits, the remaining bits are fraction bits and their count is the fraction size. In summary, the binary format is:

$$s G_{\text{g-1}} G_{\text{g-2}} \ldots G_0 e_{\text{es-1}} e_{\text{es-2}} \ldots e_0 f_{\text{fs-1}} f_{\text{fs-2}} \ldots f_0 \tag{5}$$

2.1.3 MorrisUnaryHEB(size, r)

Can MorrisBiasHEB(size, g, r) be further improved? From the last standard of posit [11], the choice for fixing the exponent size to make it depends only on the size and making the conversion between different sizes easier was taken into account. This can be adapted using

a unary representation for the g value (similar to the *regime* in posit). There is also a need for the exponent size value of -1, so the formula for the exponent size is:

$$\text{exponent size} = \begin{cases} -k-1, & k < 0 \\ k-1, & k \geq 0. \end{cases} \tag{6}$$

where k is the regime.

MorrisUnaryHEB(size, r) underflows to 0, overflows to NR, and uses TaperedFloatingPoint(size) for implementing its operations. The binary representation starts with the sign bit. The next bit represents the first regime bit r_0. The next consecutive bits with the same value as r_0 are considered regime bits. The next bit after them, if it exists, has the negated value of r_0 and it is also considered as part of the regime. The regime k is computed as:

$$k = \begin{cases} -\text{NoC0}, & r_0 = 0 \\ \text{NoC1} - 1, & r_0 = 1. \end{cases} \tag{7}$$

The next es bits or the next remaining bits (whichever is smaller) represent the exponent value in natural base 2 format, if the signum(k) is 1. Otherwise, they need to be negated and the result is the binary exponent value. If es is grater than the remaining bits, the remaining bits represent the most significant bits of the binary exponent value. The remaining least significant bits of the binary exponent value are considered 0. The exponent is computed as:

$$\text{exponent} = \begin{cases} \text{signum(k)} \times (2^{es} + \text{binaryExponent}), & es \neq -1 \\ 0, & es = -1. \end{cases} \tag{8}$$

After taking the exponent bits, the remaining bits are fraction bits and their count is the fraction size. In summary, the binary format of MorrisUnaryHEB(size, r) is:

$$s r_0 r_1 \ldots r_{\text{rs-2}} \overline{r_{\text{rs-1}}} e_{\text{es-1}} e_{\text{es-2}} \ldots e_0 f_{\text{fs-1}} f_{\text{fs-2}} \ldots f_0 \tag{9}$$

2.2 Evaluation on software NRSs

In this section, the three new proposed NRSs in addition to well-know NRSs from the literature were evaluated. In the first subsection, we present the NRSs under evaluation and their characteristics such as minimum absolute value, maximum absolute value, dynamic range, and density of numbers in logarithmic scale. The second subsection presents the decimal accuracy of the unary operations for the tested NRSs with CDF graphs. In the third subsection, the color maps of binary operations are presented. The last subsection goes through some famous literature benchmarks. The next notation are used for rounding in this section: *RZ* for rounding towards zero and *RE* for rounding to the nearest tie to even. The values presented in this section are usually truncated to three decimals after the decimal point.

Table 1: 16-bit NRSs Dynamic Range

NRS	*Min(abs(X))*	*Max(abs(X))*/1^{st}	Dynamic Range	2^{nd}	3^{rd}
FixedFloatingPoint(5, 10, RE)	3.054×10^{-4}	130944	9.6322	130880	130816
FixedPoint(8, 8, RE)	0.003	127.996	4.515	127.992	127.988
half-IEEE754/IEEE754(5, 10, RE)	5.960×10^{-8}	65504	12.040	65472	65440
Posit(16, 2, RE)	1.387×10^{-17}	72.057×10^{15}	33.715	45.035×10^{14}	11.258×10^{14}
Morris(16, 4, RZ)	9.207×10^{-19710}	1.086×10^{19709}	39418.071	5.887×10^{19689}	3.191×10^{19670}
MorrisHEB(16, 4, RZ)	4.630×10^{-9860}	2.159×10^{9859}	19718.668	3.295×10^{9854}	5.028×10^{9849}
MorrisBiasHEB(16, 4, RE)	6.061×10^{-39}	1.121×10^{77}	115.267	1.085×10^{77}	1.049×10^{77}
MorrisUnaryHEB(16, RE)	9.168×10^{-2467}	1.090×10^{2466}	4932.075	1.044×10^{1233}	5.809×10^{924}

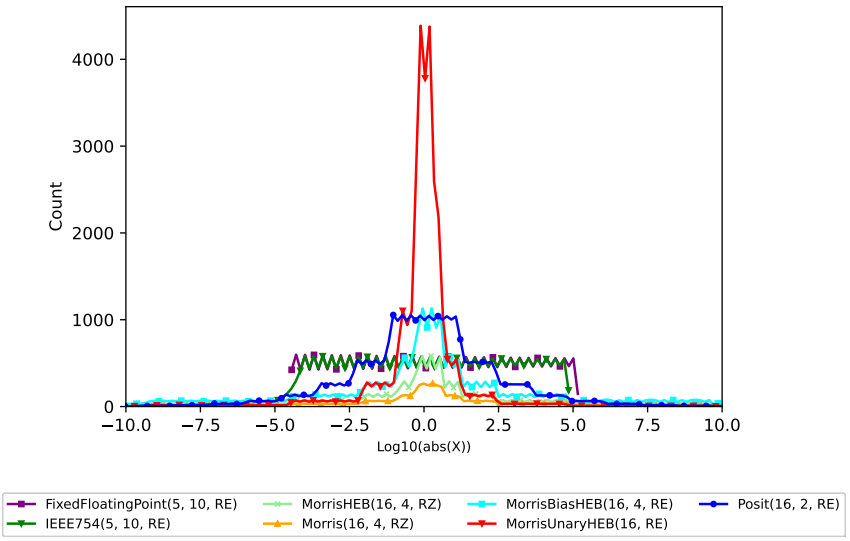


Figure 1: Distribution of unique absolute values

2.2.1 NRSs Characteristics

Table 1 presents the NRSs under evaluation with their minimum absolute value, maximum absolute value, and dynamic range when the total size is 16 bits. The three new NRSs based on Morris tapered format with hidden exponent bit are compared with the default Morris representation, fixed point, fixed floating point, IEEE754, and posit.

Tapered floating-point NRSs have a higher dynamic range and can represent higher and lower absolute values compared to IEEE754 and fixed point. On the other hand, the difference between consecutive values may be one order of magnitude. Figure 1 presents the count of unique absolute values for 16-bit NRSs on a logarithmic scale. The added value of the hidden exponent bit can be seen in the increased count of numbers for Morris-derived NRSs. An interesting result is the MorrisUnaryHEB(16, RE) "golden zone": it has 30,201 unique absolute values in the interval $(10^{-3}, 10^{3})$ versus 26,587 for Posit(16, 2, RE). This, together with the higher dynamic range, makes it a good competitor for posit in deep neural networks. Chapter 5 evaluates this.

The difference between the underflow and overflow rules of IEEE754(es, fs, r) and FixedFloatingPoint(es, fs, r) can be seen in the gradual underflow for IEEE754(es, fs, r) and the

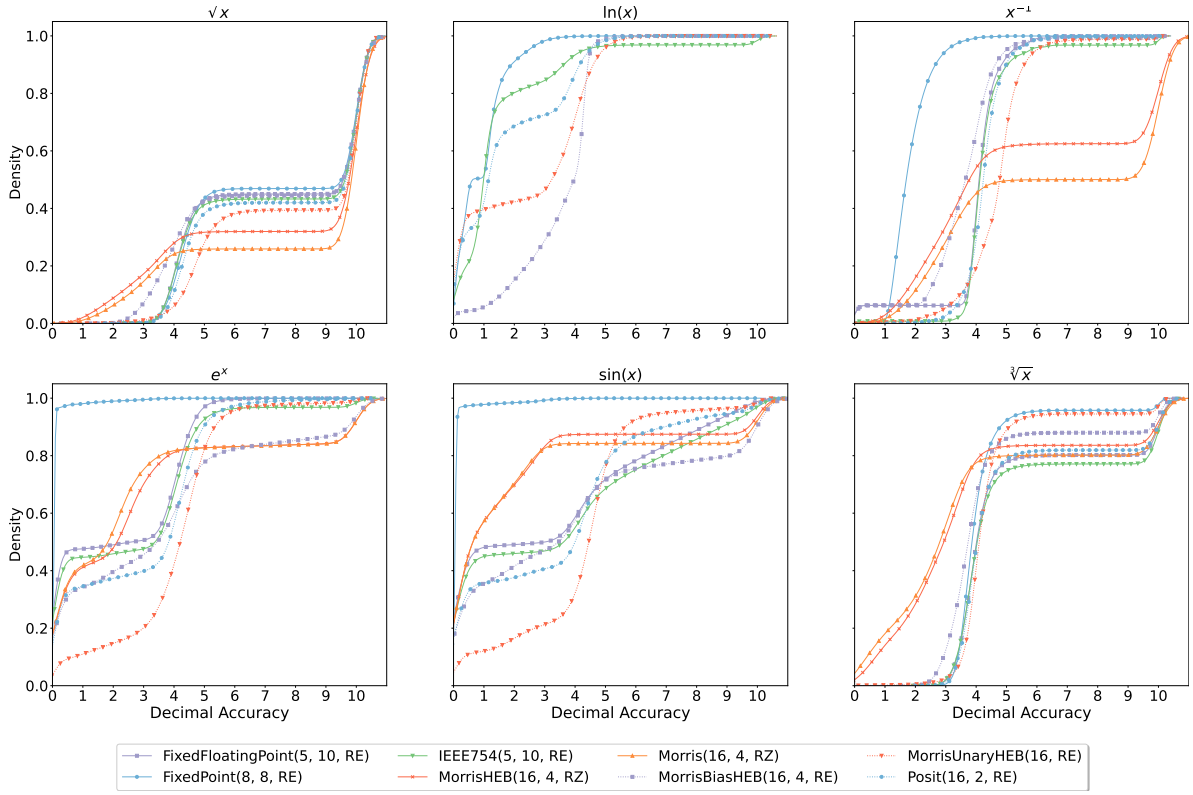


Figure 2: CDF of Unary Operations

additional higher values for FixedFloatingPoint(es, fs, r). The usage of a positive regime value for zero can be seen in the unequal distribution of the values of MorrisUnaryHEB(16, RE) and Posit(16, 2, RE) in Figure 1.

2.2.2 Unary Operations

Figure 2 presents the CDF of decimal accuracy for the square root, natural logarithm, inverse, exponential, sinus, and cube root operations. The x-axis represents how many accurate digits are there in the result. For all the Taylor series functions $(\ln(x), \sin(x), e^x)$, the decimal accuracy reference is the RationalNumber result after 30 iterations. For a decimal accuracy of at least three digits, MorrisUnaryHEB(16, RE) is the best NRS. This is because of its unique absolute values. Note that the exponential is the only function that increases the magnitude of the result.

2.2.3 Binary Operations

For binary operations, 12 bits NRSs were chosen because 8 bits hold too little information and 16 bits take too much storage space to keep all the values. The results are presented as color maps, where black represents an accuracy of 10 or more digits, while white represents zero or less.

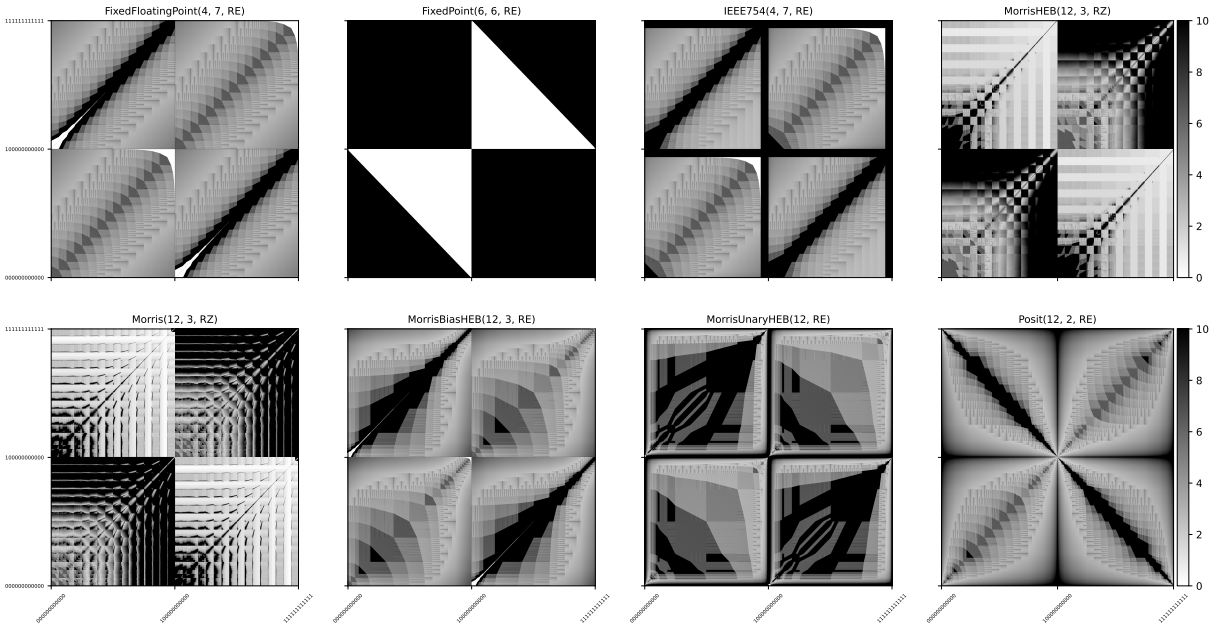


Figure 3: Color Maps for Addition

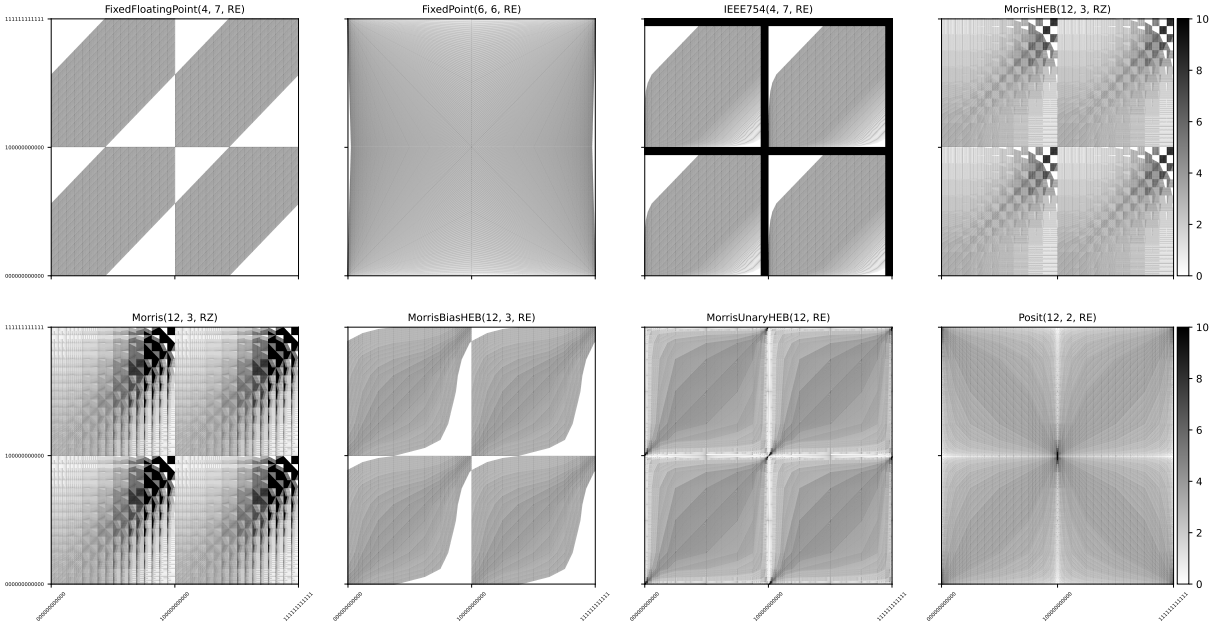


Figure 4: Color Maps for Division

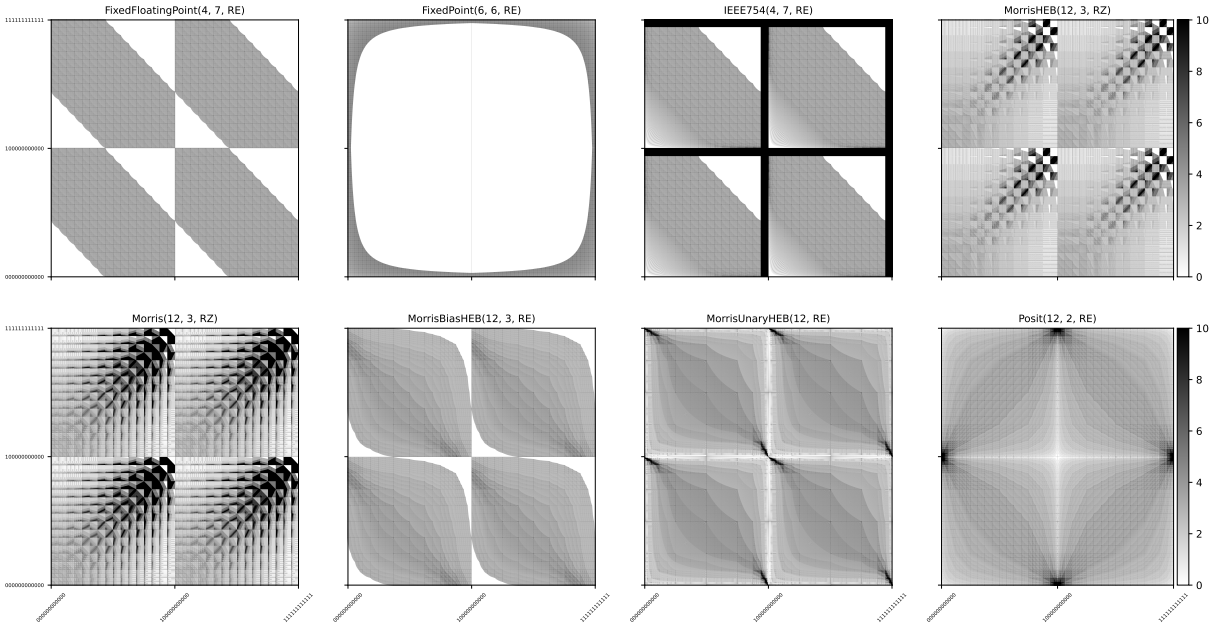


Figure 5: Color Maps for Multiplication

The color map of addition is presented in Figure 3. The subtraction is similar to addition. This plot beautifully shows why FixedPoint(is, es, r) is the perfect NRS for accumulators if the range of the results is known. The white color space represents the overflow area for positive and negative values. The similarities between FixedFloatingPoint(es, fs, r) and IEEE754(es, fs, r) are obvious, but one can also observe the effect of the gradual underflow in IEEE754(es, fs, r). The black border and the plus lines for IEEE754(es, fs, r) represent NaNs (NaN plus anything else results in a NaN).

The maps for Morris(size, g, r) and MorrisHEB(size, g, r) are different from the other maps because the binary representations do not represent ordered values. MorrisBiasHEB(size, g, r) looks like a mixed between FixedFloatingPoint(es, fs, r) and Posit(size, es, r): it exhibits tapered floating-point features by having an inverse proportional relationship between accuracy and absolute values. That is, when the absolute values of the operands increase, the decimal accuracy decreases. Posit(size, es, r) has a more uniform distribution of the accuracy. Note that Posit(size, es, r) does not use sign magnitude but uses 2's complement for negative numbers, so its map symmetry is different from the other maps.

The results of decimal accuracy for multiplication are presented in Figure 5. The overflow problem of FixedPoint(is, es, r) is obvious, while gradual underflow helps IEEE754(es, fs, r). The black borders of IEEE754(es, fs, r) are from the NaN values. MorrisUnaryHEB(size, r), Posit(size, es, r), and MorrisBiasHEB(size, g, r) exhibit their tapered floating-point proprieties in waves (or bands) of accuracy. Comparing MorrisUnaryHEB(size, r) and Posit(size, es, r), the rule for underflow can be observed as the white band in the color map of MorrisUnaryHEB(size, r). These results suggest that the Posit(size, es, r) rule for underflow might be the best one to be implemented in an NRS.

The accuracy for division is presented in Figure 4. The effect of gradual underflow in

Table 2: Binary Operations (ADD,DIV,MUL) Results

NRS	Exact			Average Accuracy			Kops		
	ADD	DIV	MUL	ADD	DIV	MUL	ADD	DIV	MUL
FixedFloatingPoint(4, 7, RE)	16.4%	2.4%	2.2%	3.3	2.4	2.4	191	374	278
FixedPoint(6, 6, RE)	75.0%	0.9%	0.9%	0.0	2.8	0.5	6535	2551	4117
IEEE754(4, 7, RE) (12.1% NaNs)	28.6%	14.3%	14.4%	3.2	2.7	2.7	362	370	326
Posit(12, 2, RE)	12.4%	4.2%	4.2%	2.8	4.0	2.8	150	206	253
Morris(12, 3, RZ)	20.9%	22.1%	26.4%	4.9	1.5	1.5	145	256	347
MorrisHEB(12, 3, RZ)	14.2%	8.9%	8.8%	5.4	1.9	1.8	185	301	385
MorrisBiasHEB(12, 3, RE)	20.2%	2.2%	2.2%	3.4	2.7	2.9	148	221	261
MorrisUnaryHEB(12, RE)	37.6%	1.9%	1.9%	4.2	3.0	3.0	142	219	263

IEEE754(es, fs, r) can be seen in the bottom-right of the quadrants, in contrast to FixedFloatingPoint(es, fs, r). The black borders of IEEE754(es, fs, r) are from the NaN values. Posit(size, es, r) and FixedPoint(is, es, r) exhibit the most uniform patters. MorrisUnaryHEB(size, r),Posit(size, es, r), and MorrisBiasHEB(size, g, r) exhibit their tapered floating-point proprieties in waves (or bands) of accuracy. Comparing MorrisUnaryHEB(size, r) and Posit(size, es, r), the rule for underflow can be observed as the white band in the color map of MorrisUnaryHEB(size, r). These results suggest that the Posit(size, es, r) rule for underflow might be the best one to be implemented in an NRS. The results of decimal accuracy for multiplication are similar to the ones for division and are omitted due to space constraints.

Table 2 presents the percentage of exact results, the average decimal accuracy for inexact results, and the number of (thousands) operations per second (Kops). From IEEE754(4, 7, RE), one should remove 12.1% of the results because they represent NaNs. The interesting results in Table 2 are: (i) the high number of exact results for MorrisUnaryHEB(12, RE), (ii) the relatively good average decimal accuracy on inexact results for MorrisUnaryHEB(12, RE) and MorrisBiasHEB(12, 3, RE) on all operations, and (iii) the relatively low percentage of exact results for Posit(12, 2, RE) (this is because of the increased exponent size).

2.2.4 Literature Benchmarks

In Table 3, the results of the evaluations proposed by Gustafson in [10] are summarised. The proposed evaluations are:

- John Wallis Product: $2 \times \prod_{i=1}^{n} \frac{(2\times i)^2}{(2\times i-1)\times(2\times i+1)}$ for $n = 30$,
- Kahan series: $u_{i+2} = 111 - \frac{1130}{u_{i+1}} + \frac{3000}{u_i \times u_{i+1}}$ for u_{30},
- Jean Micheal Muller: $E(0) = 1, E(z) = \frac{e^z-1}{z}, Q(x) = |x-\sqrt{x^2+1}| - \frac{1}{x+\sqrt{x^2+1}}, H(x) = E((Q(x))^2)$ for $H(15), H(16), H(17), H(9999)$,
- Siegfried Rump: $333.74 \times y^6 + x^2 \times (11 \times x^2 \times y^2 - y^6 - 121 \times y^4 - 2) + 5.5 \times y^8 + \frac{x}{2\times y}$ for $x = 77517$ and $y = 33096$,
- Decimal accuracy for r_1 from Quadratic formula for $a = 3, b = 100, c = 2$,

Table 3: Benchmarks in [10]

NRS	John Wallis	Kahan u_{30}	Jean Micheal Muller	Siegfried Rump	r_1 DA	David Bailey
32-bit NRSs						
FixedFloatingPoint(8, 23, RE)	3.091	100	$(0, 0, 0, 0)$	-63.382×10^{28}	5.612	(NR, NR)
FixedPoint(16, 16, RE)	3.091	100	$(1, 1, 1, NR)$	NR	3.787	(NR, NR)
IEEE754(8, 23, RE)	3.091	100	$(0, 0, 0, 0)$	-1.901×10^{30}	5.612	(NR, NR)
Posit(32, 2, RE)	3.091	100	$(0, 0, 0, 0)$	1.172	5.996	$(-4, 2)$
Morris(32, 4, RZ)	3.091	99.999	$(0, 0, 0, 0.995)$	20.282×10^{30}	4.599	$(0, 1)$
MorrisHEB(32, 4, RZ)	3.091	99.999	$(0, 0, 0, 0.989)$	15.211×10^{30}	4.945	$(2, 1)$
MorrisBiasHEB(32, 4, RE)	3.091	100	$(0, 0, 0, 0)$	-25.353×10^{29}	5.612	$(1, 0.5)$
MorrisUnaryHEB(32, RE)	3.091	100	$(0, 0, 0, 0.999)$	1.172	5.612	$(2, 0)$
64-bit NRSs						
FixedFloatingPoint(11, 52, RE)	3.091	99.999	$(0, 0, 0, 0)$	11.805×10^{20}	14.258	$(0, 1.333)$
FixedPoint(32, 32, RE)	3.091	100	$(1, 1, 1, 1)$	NR	8.570	(NR, NR)
IEEE754(11, 52, RE)	3.091	99.999	$(0, 0, 0, 0)$	11.805×10^{20}	14.258	$(0, 1.333)$
Morris(64, 5, RZ)	3.091	99.999	$(0, 0, 0, 0)$	47.223×10^{20}	14.133	$(-1.142, 2.071)$
MorrisHEB(64, 5, RZ)	3.091	99.999	$(0, 0, 0, 0)$	1.172	15.032	$(-1, 2)$
MorrisBiasHEB(64, 5, RE)	3.091	99.999	$(0, 0, 0, 0)$	11.802×10^{20}	15.030	$(-1.017, 2.035)$
MorrisUnaryHEB(64, RE)	3.091	99.999	$(0, 0, 0, 0)$	37.778×10^{21}	15.031	$(-1.004, 1.997)$
Posit(64, 2, RE)	3.091	100	$(0, 0, 51.669 \times 10^{14}, 84.750 \times 10^{8})$	1.172	15.891	$(-1.013, 2)$
RationalNumber	3.091	6.004	$(1, 1, 1, 1)$	-0.827	1	$(-1, 2)$

- David Bailey's system of equations: $0.25510582 \times x + 0.52746197 \times y = 0.79981812, 0.80143857 \times x + 1.65707065 \times y = 2.51270273$ solved with Cramer's rule.

Note that none of the NRSs passes all the evaluations. The problem is with the limitations of finite representations.

In Table 4, the results of multiple benchmarks from the literature [6, 9, 11] are presented. These benchmarks are:

- thin triangle area for $a = 7, c = b = \frac{7+2^{-25}}{2}$,
- the formula $x = (\frac{27/10-e}{\pi-(\sqrt{2}+\sqrt{3})})^{67/16}$,
- the fraction $\frac{x^n}{n!}$ for $x = 7, n = 20$ and $x = 25, n = 30$,
- Planck constant $h = 6.626070150 \times 10^{-34}$,
- Avogadro number $L = 6.02214076 \times 10^{23}$,
- speed of light $c = 299792458$,
- charge of $\bar{e} 1.602176634 \times 10^{-19}$,
- Boltzmann constant $k = 1.380649 \times 10^{-23}$.

The values in Table 4 represent the decimal accuracy of the results compared to the correct result. The first two benchmarks are favorable to Posit(size, es, r) while the last six are favorable to IEEE754(es, fs, r). Morris and its derived NRSs exhibit results that are close to the best NRS for each benchmark. MorrisBiasHEB(size, g, r) has good results for the entire spectrum of benchmarks.

Table 4: Other Literature Benchmarks [6, 9, 11]

NRS	Thin Triangle	x	$\frac{x^n}{n!}$	Planck	L	c	$\overline{e}$	k
32-bit NRSs								
FixedPoint(16, 16, RE)	0	2.289	(0, 0)	0	0	0	0	0
IEEE754(8, 23, RE)	0	4.370	(7.135, 0)	8.727	8.075	7.839	8.004	7.782
Posit(32, 2, RE)	1.204	5.684	(4.339, 0)	0.627	4.091	6.969	4.213	4.037
Morris(32, 4, RZ)	0	5.101	(6.016, 5.604)	6.347	6.429	6.969	7.347	6.480
MorrisHEB(32, 4, RZ)	0	5.098	(6.245, 6.188)	6.680	6.784	7.839	7.347	6.480
MorrisBiasHEB(32, 4, RE)	0	5.682	(6.911, 8.619)	7.053	7.219	7.839	7.347	6.878
MorrisUnaryHEB(32, RE)	0	6.875	(6.017, 5.289)	6.031	5.919	6.969	7.347	5.566
64-bit NRSs								
FixedPoint(32, 32, RE)	8.343	7.976	(0, 0)	0	0	∞	0	0
IEEE754(11, 52, RE)	16.710	13.030	(16.644, 15.905)	16.952	17.028	∞	16.540	16.537
Morris(64, 5, RZ)	16.710	14.221	(16.644, 14.875)	16.172	15.810	∞	16.026	15.467
MorrisHEB(64, 5, RZ)	16.710	14.710	(16.950, 15.794)	16.172	16.238	∞	16.540	15.807
MorrisBiasHEB(64, 5, RE)	17.341	14.831	(17.263, 16.012)	16.952	17.028	∞	16.540	16.197
MorrisUnaryHEB(64, RE)	17.341	15.323	(15.743, 14.942)	16.172	15.458	∞	16.026	15.467
Posit(64, 2, RE)	17.835	15.672	(14.018, 8.372)	11.094	13.200	∞	14.263	12.974

2.3 Insights on number representation systems

In this chapter, (i) a Scala library that simplifies the adding, testing, and fine-tuning number representation systems (NRSs) is presented, (ii) three new NRSs, based on Morris tapered floating-point, are introduced, and (iii) these three proposed NRSs are analysed together with well-known NRSs such as IEEE754 floating-point and posit.

By adding the hidden exponent bit to Morris tapered floating-point in three different forms, the resulting NRSs became competitors for IEEE754 and posit. MorrisBiasHEB(size, g, r) exhibits the best results on literature benchmarks on 32 and 64 bits when compared to the other NRSs. On the other hand, MorrisUnaryHEB(size, r) is a great candidate for machine learning computations due to its "golden zone" population, dynamic range, percentile of exact results on addition and average decimal accuracy for inexact results on multiplication.

The library exhibits a performance of 200 Kops, which is good enough for testing and evaluating NRSs, but not enough for real-world applications. In future works, the library will be integrated with the Aparapi library[1] and tested on GPU and used for machine learning models with Spark.

[1] https://aparapi.com/

3 NRS IN SCIENTIFIC COMPUTING

3.1 Benchmarks Results

For the below benchmarks, the next eleven NRSs were used: 32-bit IEEE754 (8 bits exponent and 23 bits mantissa) called IEEE754, 32-bit Posit (2 bits exponent) called Posit, 16-bit IEEE754 (5 bits exponent and 10 bits mantissa) called half-IEEE754, 16-bit IEEE754 (8 bits exponent and 7 bits mantissa) called bfloat16, 19-bit IEEE754 (8 bits exponent and 10 bits mantissa) called TF32, 24-bit IEEE754 (7 bits exponent and 16 bits mantissa) called FP24, 24-bit IEEE754 (8 bits exponent and 15 bits mantissa) called PXR24, 32-bit floating-point (8 bits exponent and 23 bits mantissa) called FloatP, 32-bit fixed-point (16 bits integer and 16 bits fractional) called FixedP, infinite precision fixed-point called IP_NR_FixedP (FixedPoint(∞)), infinite precision floating-point called IP_NR_FloatP (FloatingPoint(∞)). The rounding method used was rounding to the nearest tie to even.

3.1.1 Conjugate gradient method benchmark

For conjugate gradient method benchmark, the reference NRS is 1024-bit FractionalNumber(512, 512, RE) (512 bits integer nominator and 512 bits unsigned integer denominator), $N = 5$, MAX POWER $= 3$. Because several matrix multiplications are executed on every iteration of the algorithm, the magnitude order of the values used grows fast during the execution. This is why high-magnitude order numbers were not generated as input. Again, the advantage of Posit on small numbers can be noticed, but its disadvantage with large num-

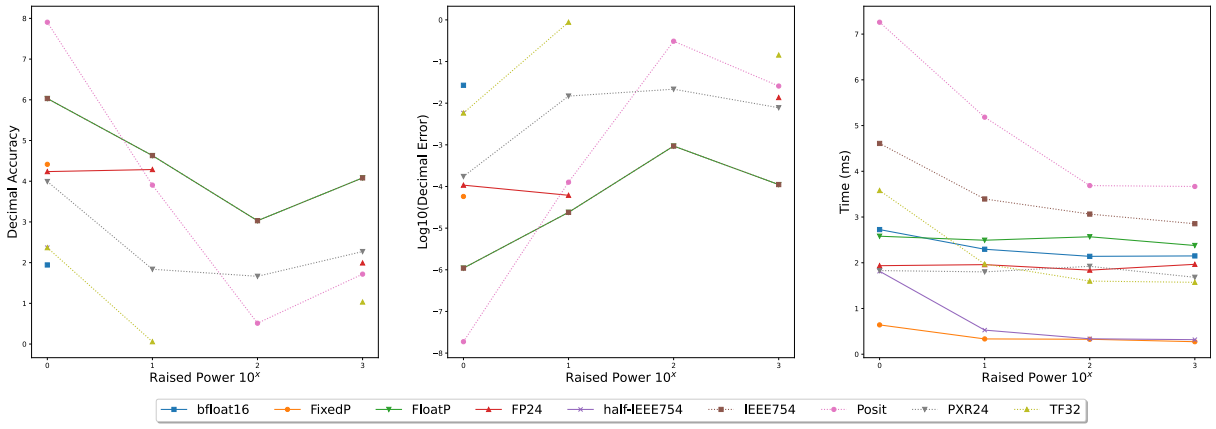


Figure 6: Decimal accuracy, decimal error and computation time for conjugate gradient method benchmark

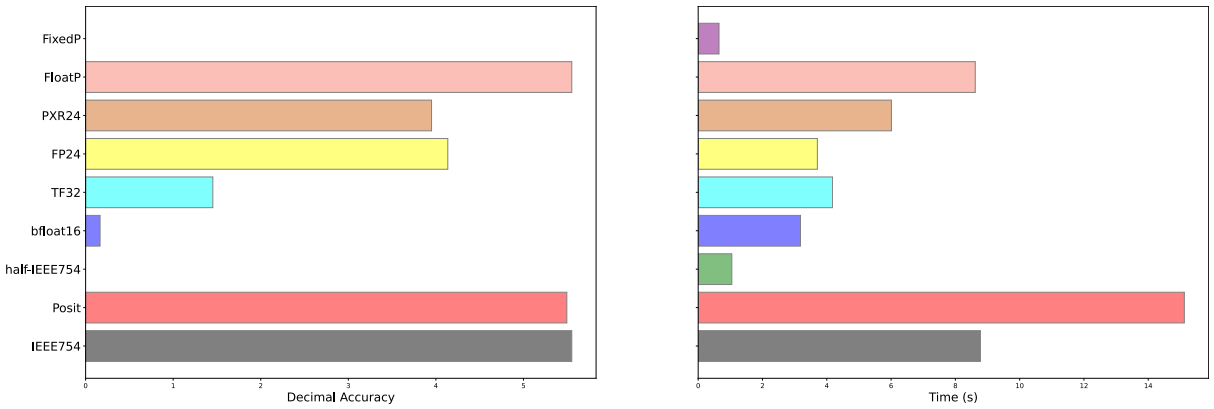


Figure 7: Decimal accuracy and computation time for Simpson's integration benchmark

bers stands out even more during this test. It can be seen that it loses half of its accuracy during the test, which makes it even less accurate than PXR24, which uses 8 bits less than Posit. TF32 shows a behaviour that is harder to predetermine. After multiple iterations of the algorithms as can be seen in Figure 6, its accuracy does not seem to take into account the order of the input values. It follows an irregular pattern that increases and decreases with many exact decimals in general, although for a few runs it kept a constant difference of one decimal. Regarding the time, the results were pretty predictable. IEEE754 and Posit need the most time, but they also produce the most accurate results. Regardless of the number of bits used and the accuracy obtained, the other NRSs had about the same running time range.

3.1.2 Simpson's Integration benchmark

For Simpson's integration benchmark the reference NRS is 1024-bitFractionalNumber(512, 512, RE) (512 bits integer nominator and 512 bits unsigned integer denominator) , $N = 1000$, $[a, b]$ is $[0, 10]$, and the functions are: x^2, x^3, $\sqrt{x}$, $\sqrt[3]{x}$, $\sqrt[3]{\frac{x^2+\sqrt{x}}{x+7}}$, $(\sqrt[5]{x^2+34} \times x^3)^{\frac{3}{4}}$. Being an algorithm that does not perform so many complex calculations, the accuracy does not differ too much between the representations with the same number of bits (Figure 7 e.g. Posit, IEEE754, FloatP). In this test, the computing time becomes relevant because it can be seen that even though those 3 NRSs using 32 bits got the same accuracy, the difference in time strongly disadvantages Posit which runs for twice as long as IEEE754.

3.1.3 N-body simulation benchmark

For the N-body simulation benchmark, the reference NRS is 1024-bit FractionalNumber(512, 512, RE) (512 bits integer nominator and 512 bits unsigned integer denominator), $N = 5$, $M = 10$, iteration time step $= 1$ (second), $G = 6.67498 \times 10^{-11}$. Positions, velocities and mass were separated in two magnitude order ranges (low and high). For high magnitude order range, positions x and y of every particle are generated in the interval $[-10^{11}, 10^{11}]$, velocities x and y of every particle are generated in the interval $[30, 5 \times 10^6]$ and mass of every particle is

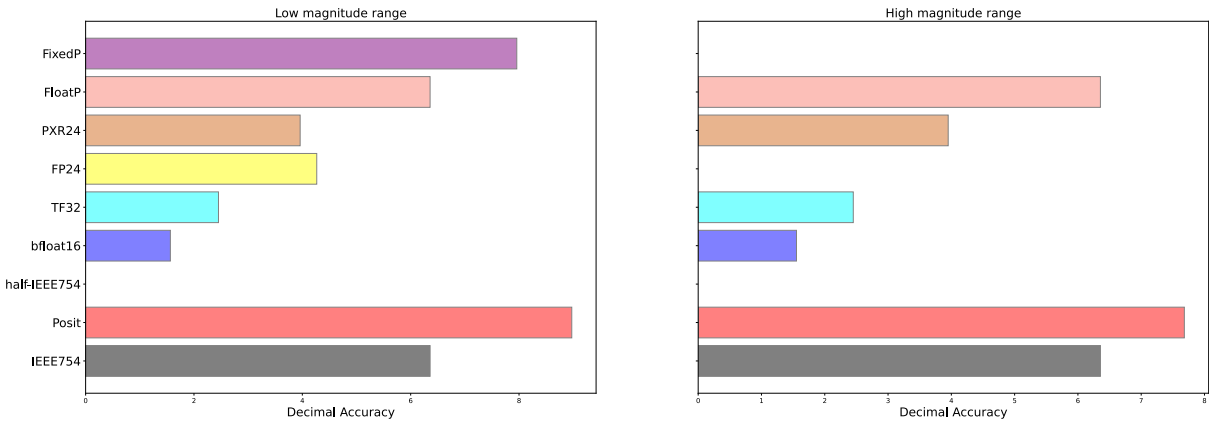


Figure 8: Decimal accuracy for N-body simulation benchmark

generated in the interval $[1, 6 \times 10^5]$. For low magnitude order range, positions x and y of every particle are generated in the interval $[0, 10^4]$, velocities x and y of every particle are generated in the interval $[30, 50]$ and mass of every particle is generated in the interval $[10, 60]$. Figure 8 suggests the superiority of Posit by obtaining almost three more exact decimals than the current standard IEEE754 and one more exact decimal than the second most accurate NRS FixedP. However, the remarkable performance of Posit and FixedP can be easily explained by the ranges from which the input data for each particle were chosen. Small numbers for particle coordinates were chosen to have them close enough to each other so that the attraction force between them is noticeable. Using high magnitude order values (even outside of the Golden Zone) made FixedP to return NR.

3.2 Insights on Scientific Computing Results

This current chapter proposed a benchmark for base scientific computing algorithms under different NRSs. The benchmark used four algorithms: matrix multiplication, solving a linear system of equations by gradient conjugate method, integration by Simpson's formula, and N-body simulation. The algorithm's input has been varied, so the performance of all eleven NRSs could be analysed under different circumstances. The results of the simulations offer a perspective of the trade-off between different NRSs.

The gold zone of Posit [6] has been validated through all experiments. The possibility of using IEEE754 half-precision in some specific cases, where the number range does not surpass its dynamic range, is an exciting result. It can be seen as a faster and more energy-efficient NRS solution for a scientific computing application. If the developers desire a faster solution, fixed-point NRS can be their option, given the requirements offered in this chapter.

Replacing the current standard with a new NRS may be necessary, given the results presented. However, the lack of hardware implementation and industry adoption will delay the process.

4 THE USAGE OF NRSS FOR STATISTICAL METHODS

4.1 Statistical Methods Evaluation

According to figure 9, it can be noticed that the accuracy obtained by Posit is better than that of the IEEE754 on the same number of bits. Also, getting valid results on fewer bits, as seen for Posit in some cases, is a win in terms of both memory and execution speed. According to the results obtained in 4.1, where the performance of the IEEE754 versus Posit on elementary mathematical operations was analysed, it can be considered that the difference in accuracy between Posit, IEEE754 and the actual result was expected. This is because Posit provides much better accuracy than IEEE754 on a smaller number of bits when discussing addition and subtraction. Still, its accuracy decreases when variance or radical calculations are also involved, especially on larger data sets. It can be observed that the precision obtained

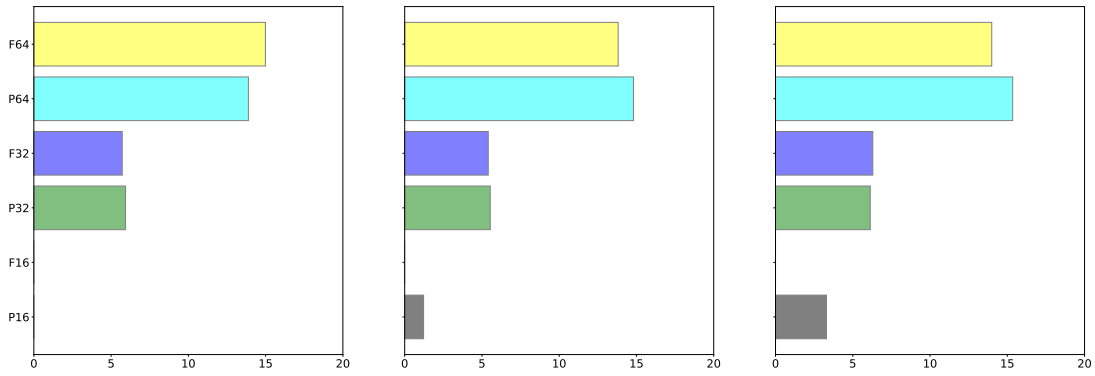


Figure 9: Decimal accuracy for Pearson coefficient

on the different number representation systems is close for both one data set and two data sets. Although Posit is closer to the actual value of the result at any number of bits, the differences are not extremely large. Nevertheless, it still provides more correct digits than its counterpart. The Kolmogorov-Smirnov test contains only basic mathematical operations (addition, subtraction, multiplication and division). So, on average, for data sets with values greater than 1, the accuracy of the two data types is often similar, and this is also the case in this example. It should be noted that on 16-bit, the bits used by IEEE754 were not enough to represent the result. According to figure 11, posit has more identical decimal digits with the actual result compared to IEEE754. So, the accuracy of the Posit number representation system still provides several significant decimals and overall a result closer to the actual value. The Shapiro-Wilk test contains, in addition to the basic mathematical operations (addition, subtraction, multiplication and division), operations of variance and raising to powers. Although Posit presents a problem of accuracy in calculating the variance

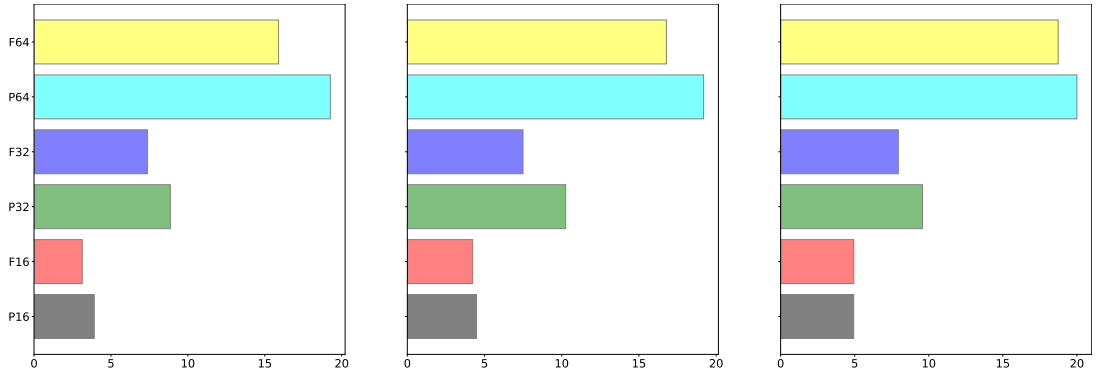


Figure 10: Decimal accuracy for Kolmogorov-Smirnov Test for one dataset

compared to IEEE754, the multitude of the operations in the formula, plus the number of bits on which the number is represented, made the Posit number representation system have better accuracy than its competitor still. As the data set grows, the operations' results become higher in absolute value, and the 16-bit (5 exponent bits and 10 fraction bits) are no longer enough to produce a valid result for IEEE754. At the same time, 16-bit Posit continues to provide relevant results with reasonable accuracy. The mathematical operations for determining the

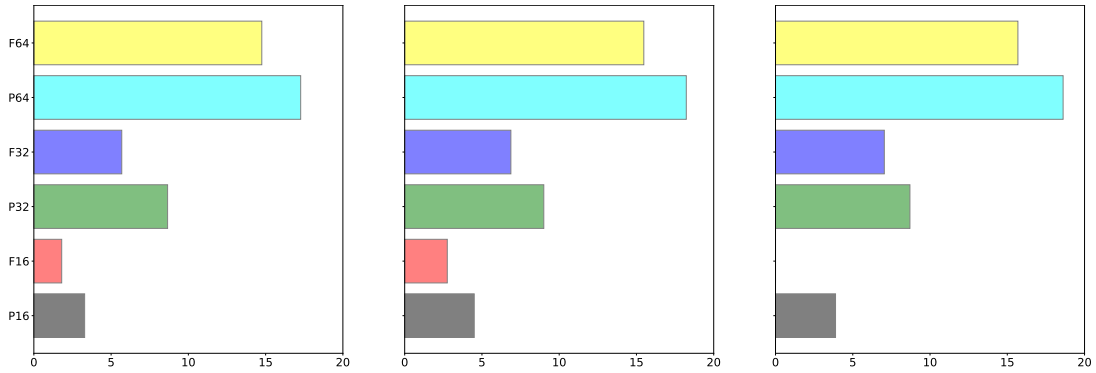


Figure 11: Decimal accuracy for Shapiro-Wilk test

ANOVA statistic involve, in addition to basic arithmetic, operations for variance. According to the results of 4.3, Posit gives poorer accuracy results on the variance calculation. However, all this accumulation of operations led to a similar accuracy between Posit and IEEE754, the difference being the number of decimals displayed, where Posit wins. It should be noted that Posit accuracy is still better than IEEE754 for both one-way ANOVA and two-way ANOVA. According to figure 13, the accuracy of Posit is better than that of IEEE754, it has a higher number of decimals, and the number of exact decimals is higher than IEEE754. The Kruskall-Wallis test is a rank test, which means that it has values from 1 to N - the number of values in the data set. As a result, for small and medium data sets, the representation problem does not appear even on 16-bit. In addition, the fact that the formula is composed of basic arithmetic operations contributes to the previous idea. Within Figure 14), it can be seen that Posit has slightly better accuracy on all tests and shows more correct digits, but the differences are not

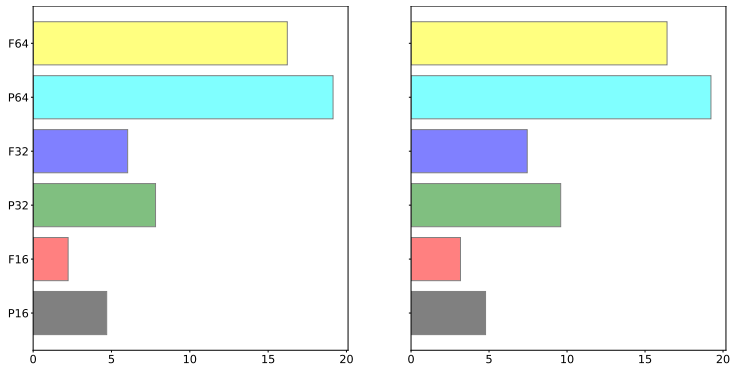


Figure 12: Decimal accuracy for two-way ANOVA

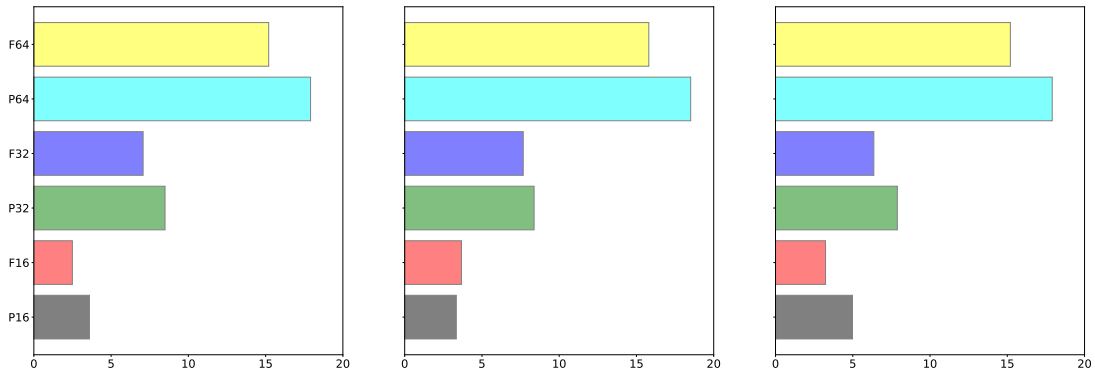


Figure 13: Decimal accuracy for Kruskall-Wallis test

extremely large between the two. Of course, as the number of inputs (including calculations) increases, the results obtained in F32 have less accuracy. The higher accuracy of P32 may be due to the maximum number of bits declared for the exponent, compared to F32, which has a fixed number of bits.

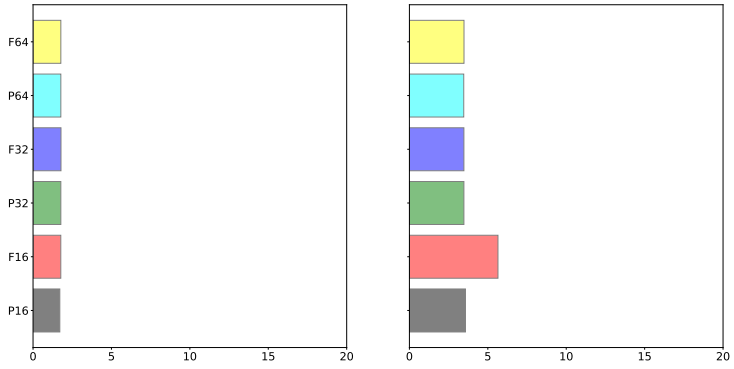


Figure 14: Decimal accuracy for T-test for pairs

4.2 Insights on Statistical Methods Results

This chapter examined the effect of changing the number representation system within statistical methods. Although at first glance, a difference of a few hundredths may not seem to have much impact on the final result, there are fields, such as biostatistics or astrostatistics, where any extra tenth is considered a piece of significant new information. The statistical functions implemented were chosen because they are relevant functions within the statistical domain. Elementary statistical methods were implemented in Python, and correlation, distribution detection, and statistical differences were written in Scala. The Scala language was used for complex functions because the Posit system in it is more powerful and provides support for more operations. The following results were observed in the fight between the Posit number representation system with the already established IEEE754 number representation system. Observed Results:

- Although the data set on which the test is also performed matters Posit always provides better accuracy than IEEE754 on the same number of bits, regardless of the statistical method used.
- Depending on the data set and the statistical method, IEEE754 can not give a valid result (in this case on 16-bit), while Posit provides a good result on the same number of bits; this is due to the flexibility of Posit in terms of exponent and fraction, compared to IEEE754 which has a fixed number (5 bits for exponent and 10 bits for fraction).
- In the case of tests that rely almost entirely on calculating the variance in the formula, such as the t-test, IEEE754 has better accuracy than Posit.
- In the case of tests that also contain calculus of variance plus other arithmetic operations, such as ANOVA, the Posit still offers better accuracy, a more significant number of decimal digits, and better results on fewer bits.
- In Mann-Whitney and Chi-Square, Posit and IEEE754 have a close accuracy, the formula being quite simple from a mathematical point of view, and therefore there are no significant differences in calculation; however, it can be considered that Posit is better than IEEE754 because it can provide meaningful results on fewer bits.

Following this chapter, the strengths and weaknesses of the Posit number representation system within the statistical methods were studied compared to the current IEEE754 number representation system. The implemented operations were: basic arithmetic, correlations, and statistical differences. The main reason why Posit is a strong competitor in the fight to replace the IEEE754 is the fluctuating character of the exponent and the mantissa. This helps to obtain significant results on a smaller number of bits, thus providing a variant that consumes less memory and is faster in statistical calculations. Posit can be considered a replacement for IEEE754 in statistical methods, often performing better. More thorough research is still needed to see the full power of this new number representation system.

5 NRSS IMPACT ON ARTIFICIAL INTELLIGENCE

5.1 Frameworks Results

For the first LPML evaluation, the MNIST data set is used for training a neural network formed out of two fully connected layers followed by a logSoftmax layer. The learning rate of 0.01, the batch size of 64, and the number of epochs 3 were used for training. The time results for forward and backward propagation are presented in Table 5. The total time for Posit and Fixed-Point was estimated. The software overhead is too high for doing feasible training using other number representation systems than IEEE754. From this point on Posit and Fixed-Point were used only for inference and precision optimisation. A consistent improvement of the software version of Posit or Fixed-Point or a hardware implementation can reopen the subject of training. The LPML framework is ready for this and can be used.

The IEEE754 resulting network has an accuracy of 93.47%. Uniform precision conversion was used for the fully connected layers and the results are presented in Table 6. The Fixed-Point number representation system can show the boundaries of neural networks. The decrease to 4 bits for the fraction part produces an accuracy degradation of 26.26%, incomparable with the decrease of the integer part to 4 bits (0.19% accuracy degradation). In the tests going to 2 bits for integer parts degrades the accuracy below 50%. This validates that Fixed-Point is more sensitive to fraction size than integer size in neural networks. Posit has better results than Fixed-Point even on this small network with the same precision. The interesting result is that Posit(32,2) increases the accuracy of the network by 0.15%. Posit(16,1) reduce the memory used for the fully connected layers by 50% with an accuracy degradation as small as 0.02%. Posit(8,0) offer a solution for a high reduction of memory (75%) with a cost on accuracy of 1.29%. Given the bad performance of Fixed-Point for LPML even on a small network, but also in the next experiments their result was omitted. The next LPML experiments present

Number Representation system	Precision	Forward Propagation (one)	Forward Propagation (total)	Backward Propagation (one)	Backward Propagation (total)
IEEE754	32	0.0006s	13.1s	0.0006s	12.6s
Posit	8	43.2s	33.75h	74.5s	58.20h
Posit	16	42.6s	29.25h	73.9s	57.73h
Fixed-Point	4-4	455s	355h	878s	685h
Fixed-Point	8-8	460s	359h	899s	702h

Table 5: Forward and Backward Propagation Time

Number representation system	Precision	Accuracy	Δ Acc	Memory reduction
IEEE754(8,23)	32	93.47%	+0.00%	0%
Posit(8,0)	8	92.18%	-1.29%	75%
Posit(16,1)	16	93.45%	-0.02%	50%
Posit(32,2)	32	93.62%	+0.15%	0%
Fixed-Point(4,4)	8	67.21%	-26.26%	75%
Fixed-Point(4,8)	12	93.28%	-0.19%	62.5%
Fixed-Point(8,8)	16	93.42%	-0.05%	50%

Table 6: Accuracy and memory reduction for different number representation systems on simple network

Method	Layer Precision	Accuracy	Memory reduction
Classic	32-32-32-32-32-32-32-32-32-32	94.11%	0%
KD	32-32-32-32-32-32-32-32-32-32	94.98%	0%
KD+PO	32-32-32-16-16-16-16-8-8-32	93.85%	43.47%

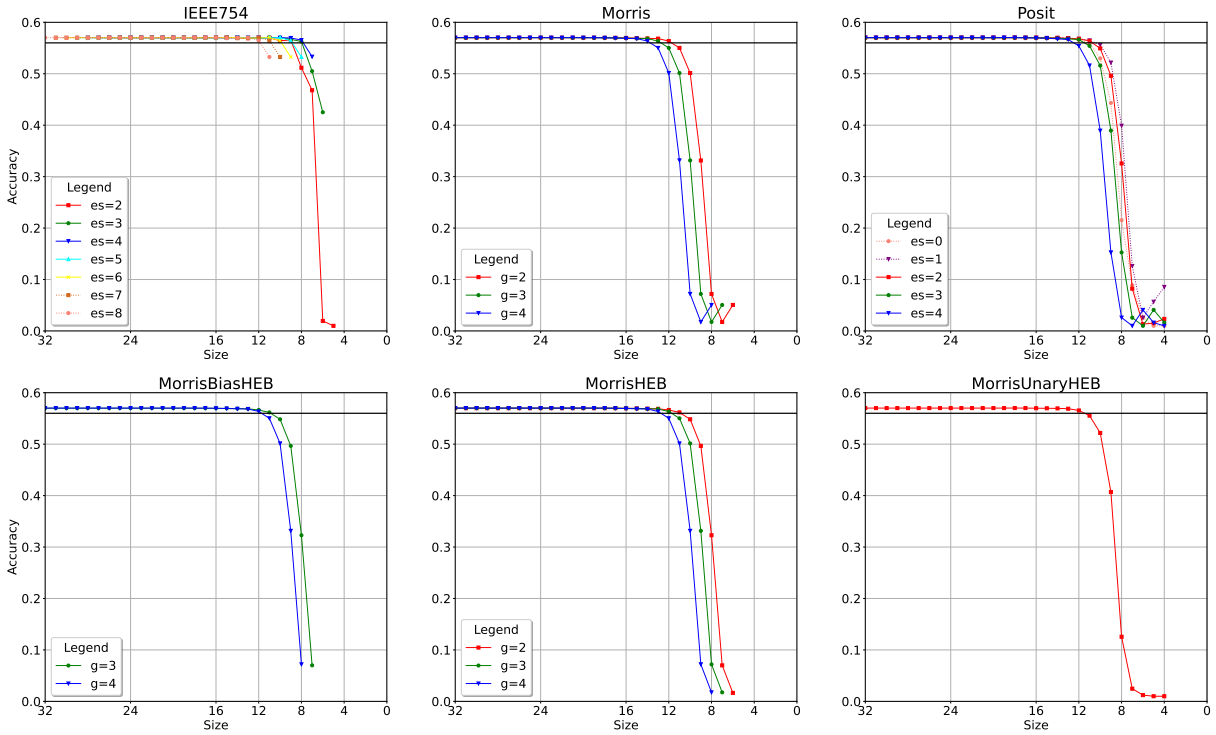
Table 7: Accuracy for ResNet18 on different training methods

the usage of Posit and IEEE754.

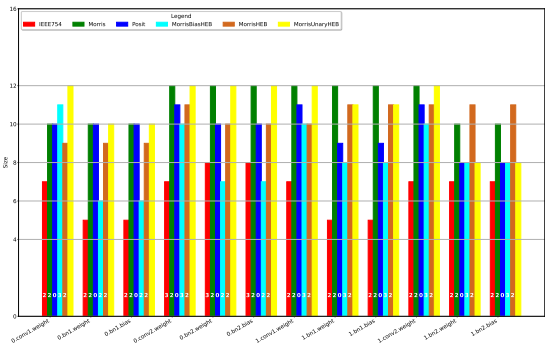
To test LPML knowledge distillation training and layer precision optimisation the CIFAR-10 data set, the ADAM optimiser, the learning rate of 0.005 and the accuracy threshold of 2% were used. The deep neural network family is Resnet, Resnet34 is the teacher network and Resnet18 is the student network. The increase of accuracy for knowledge distillation training versus classic training is validated in Table 7 with a value of 0.87%. The usage of layer precision optimisation on the knowledge distillation trained model has an accuracy degradation of 0.26% compared to the classic training a reduction of memory for fully connected layers and convolutional layers of 43.47%. Resnet34 has an accuracy of 95.23% classic trained. The Resnet18 trained with knowledge distillation and layer precision optimisation reduces the storage size for fully connected layers and convolutional layers with 66.75% for a degradation in accuracy of 1.38%. The Resnet18 trained with knowledge distillation without layer precision optimisation reduces the storage for the same layers by 41.19% with degradation in accuracy as small as 0.25%. The usage of knowledge distillation and layer precision optimisation can reduce the storage space of a deep neural network by half with a degradation close to 1%.

For the first CPML evaluation, the CIFAR-100 data set is used for training a ResNet18 neural network. The optimiser used is ADAM coupled with the StepLR learning rate scheduler. The learning rate of 0.001, the batch size of 128, and the number of epochs 10 were used for training. First, uniform precision optimisation was used, and the results are in Figure 15. The accuracy threshold chosen is 1% and is seen as the horizontal bold line. IEEE754 gives us

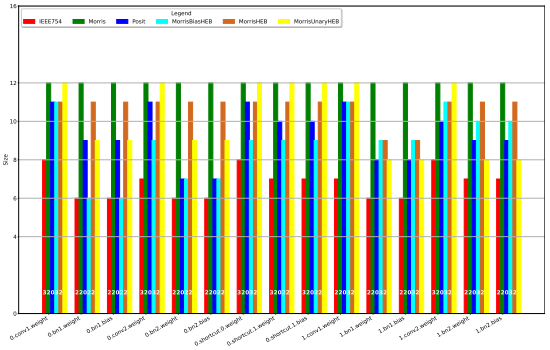
Figure 15: CPML Uniform Precision



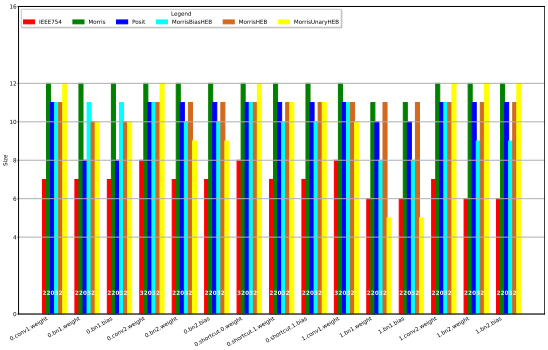
information about the minimum values. There is a need for at least 3 bits for mantissa. All the IEEE754 systems with a mantissa less than 3 are under the line. For exponent bits, the best value is 4. Higher values are unnecessary, and lower values degrade faster in accuracy. The exponent values are between $[-7, 8]$. IEEE754 has the smallest size with a value of 8 for IEEE754(4,3). MorrisBiasHEB, MorrisHEB, and Posit follow with a size of 11. What is interesting for Posit is that $es = 1$ has the best accuracy on all sizes, but the standard $es = 2$ is still close. Morris and MorrisUnaryHEB need at least 12 bits to stay within the accuracy threshold. MorrisHEB improves Morris by 1 bit. MorrisUnaryHEB is beneficial because it does need extra elements to know its custom form like the other number representation systems. The smallest size within the accuracy threshold for every number representation system is chosen to enter the layer precision optimisation process. Figure 16 contains the sizes of every number representation system used for the layers in the Resnet18. With white on every bar is written the exponent size for IEEE754 and Posit, and the g value for Morris, MorrisBiasHEB and MorrisUnaryHEB. The only system which has all the sizes under 8 is IEEE754. The next best system is MorrisBiasHEB, and the worst system is Morris. IEEE754 shows the exponent value range for every layer. The only systems to have better results than IEEE754 on some of the layers are MrrosiBiasHEB and MorrisUnaryHEB. The reader needs to be reminded that MorrisUnaryHEB does need extra information represented by the white digits. In Table 8 the result of uniform and layer precision optimisation are presented. The initial accuracy is 57.01%. For IEEE754 with an accuracy degradation less than 2%, the storage space is reduced by 76.68%. The MR (memory reduction) between the uniform precision optimised network



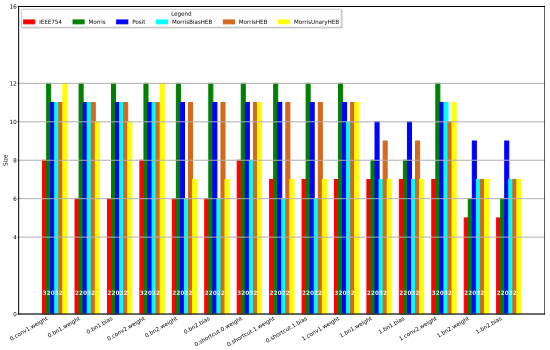
(a) Block 1



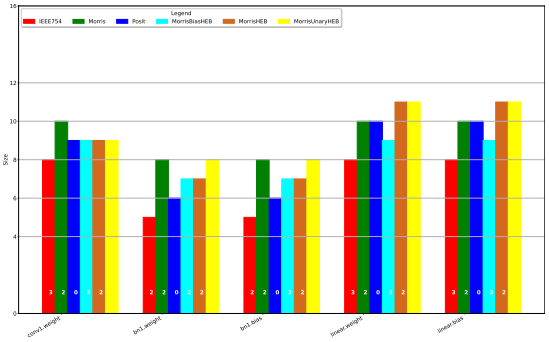
(b) Block 2



(c) Block 3



(d) Block 4



(e) First and last Block

Figure 16: Resnet18 Layer Precision Optimisation

	IEEE754	Morris	Posit	MorrisBiasHEB	MorrisHEB	MorrisUnaryHEB
Uniform Accuracy	56.27%	56.35%	56.48%	56.17%	56.17%	56.55%
Uniform Size (KB)	10957	16435	15066	15066	15066	16435
Uniform MR	75%	62.5%	65.62%	65.62%	65.62%	62.5%
LPO Accuracy	55.38%	55.49%	55.49%	55.24%	55.23%	55.62%
LPO Size (KB)	10217	16412	15035	14662	14763	15688
LPO MR	6.75%	0.14%	0.20%	2.68%	2.01%	4.54%
LPO+Uniform MR	76.68%	62.55%	65.69%	66.54%	66.31%	64.20%

Table 8: CPML Results

and layer precision optimised (LPO) network is less than 10% on all systems (LPO MR). The overhead of keeping the metadata and processing layer precision optimisation might surpass the benefits. Suppose the best sizes from all the number representation systems for every layer are considered for representing the final model. In that case, the size of the network will be only 1 KB less than the IEEE754 counterpart. The presented results show that IEEE754 uniform precision optimisation is the best choice. It reduces the storage space by 75% by losing less than 1% in accuracy. This is explained by the fact the initial training and inference testing are done on 32-bit IEEE754. Future work must analyse the usage of other number representation systems for training. This is improbable given the lack of specialised hardware.

5.2 Insights on Machine Learning Frameworks

LPML framework offers a way of training deep neural networks with Posit and Fixed-Point. Given the current software and hardware implementations, this type of training is inefficient regarding computation time. Posit is five times, and Fixed-Point is six times orders of magnitude slower. Fixed-Point might be used for inference and storage on energy-constraint devices. The results on small networks show an accuracy degradation of 0.19% for a storage size reduction of 62.5% for fully connected layers. Posit offers a better trade-off for accuracy. On a small network with two fully connected layers followed by one LogSoftmax layer, 16-bit Posit ($es = 1$) reduces the fully connected size by 50% with a loss in accuracy of 0.02%.

For the deep neural networks, the LPML knowledge distillation training combined with layer precision optimisation reduces the size of a Resnet34 network by 66.75% with a loss of 1.38% in accuracy or without layer precision optimisation, a size reduction of 41.19% with a loss of only 0.25%.

The accuracy increase for knowledge distillation training was validated by 0.87% on a Resnet18 trained with a Resnet34 versus a classic trained Resnet18. The use of layer precision optimisation reduces the size of fully connected and convolutional layers by 43.47% with a loss of 1.13%.

CPML framework offers uniform or layer precision optimisation in six different number repre-

sentation systems. The accuracy degradation for layer precision optimisation is less than 2% with a memory reduction of 76.68% in the best case (IEEE754). All the number representation systems have over 60% memory reduction on uniform and layer precision optimisation. The uniform precision optimisation using the IEEE754 has the best results reducing the storage size by 75% and losing only 0.74% in accuracy. Layer precision optimisation does not show significant benefits compared to uniform precision optimisation, and the cost of keeping the metadata for every layer might make the gap even smaller. The interesting aspect of the results is that an 8-bit IEEE754 ($es = 4$ and $fs = 3$) is used in uniform precision. It is the only number representation system with a uniform precision equal to or less than 8. These results are influenced by the fact that the training and inference testing is done on 32-bit IEEE754.

Future work must concentrate on implementing hardware accelerators to make training under other number representation systems feasible in time. The current work is concentrated on embedded systems, but given the good results in accuracy, more complex deep neural networks and data sets can be analysed in the future.

6 THE DEVELOPEMENT OF THE NRSS HARDWARE GENERATOR LIBRARY (NRS-HGL)

6.1 FPGA evaluation

The FPGA evaluation has targeted the board Virtex-7 VC709 Evaluation Platform. The conditions are 25C Temperature, Heat sink high, 500 LFM Airflow, and Process type - maximum. The Xilinx Vivado 2022.1 runs on Windows 11 Education 21H2 22000.1219.

A number of 25 number representation systems (8-bit 5, 16-bit 8, 19-bit 1, 24-bit 2, 32-bit 7, 64-bit 2) are evaluated. Their internal exponent and fraction sizes are in Table 9. The IEEE754 derivated number representation systems use the same amount of bits for internal sizes as in their binary representation. MorrisHEB trades one exponent bit for one fraction bit compared to Morris. MorrisBiasHEB is close to the IEEE754 counterparts. MorrisUnaryHEB uses the most bits out of all number representation systems of the same size. Morris and MorrisHEB follow it. The unary/binary operations, TFPU, KAU and GFPU modules are evaluated on power consumption, hardware resource utilisation, delay, period and maximum frequency. All the units use only one cycle. The performance has room for improvement. The results presented must be considered as a baseline.

Starting with 8-bit addition/subtraction modules, the one which has the lower power consumption is Morris. Morris has multiple representations, and it can be a significant amount in lower bit sizes. MorrisBiasHEB is the most effective regarding frequency, delay, period, and hardware resources. The internal exponent size is 2, so it can only represent $-3, -2, -1, 0, 1, 2, 3$ values for the exponent. The dynamic range might need to be bigger for some applications to be considered usable NRS. The trade-off of taking one bit of exponent to transform it into one bit of mantissa is seen in the differences between Morris and MorrisHEB. The number of LUTs decreases, and the maximum frequency increases for lower bit sizes. Posit and MorrisUnaryHEB have almost double the cost of LUTs and a decrease in maximum frequency. Posit has the highest power consumption out of them. Going to 16-bit, the dominance of MorrisBiasHEB in speed (maximum frequency, delay, period) is obvious. Posit and MorrisUnaryHEB have almost double the amount of LUTs compared to the other number representation systems. The difference between Morris and MorrisHEB regarding power consumption and LUTs is minimal. Google's brainfloat (bfloat16) is better than half-precision IEEE754 in all departments (lower power consumption, fewer hardware resources, better speed). Regarding speed, IEEE754 are the slowest in this situation. An interesting result is that the rule of power consumption and the number of LUTs for trading off exponent bits with mantissa bits change for larger sizes. A transformation of an exponent bit in a mantissa bit for Posit increases the power

Number representation system	Internal Exponent	Internal Fraction	Accumulator Size	Accumulator Fraction
Morris(2, 8, RoundZero)	4	3	62	30
Morris(3, 16, RoundZero)	8	10	1028	516
Morris(4, 32, RoundZero)	16	25	262162	131090
MorrisHEB(2, 8, RoundZero)	3	4	34	18
MorrisHEB(3, 16, RoundZero)	7	11	520	264
MorrisHEB(4, 32, RoundZero)	15	26	131094	6558
MorrisUnaryHEB(8, RoundEven)	6	5	130	64
MorrisUnaryHEB(16, RoundEven)	14	13	32770	16384
MorrisUnaryHEB(32, RoundEven)	30	29	2×10^9	1×10^9
MorrisBiasHEB(2, 8, RoundEven)	2	5	20	12
MorrisBiasHEB(3, 16, RoundEven)	4	12	66	34
MorrisBiasHEB(4, 32, RoundEven)	8	27	808	296
Half Float(Round Even)	5	10	80	48
Google Brainfloat(Round Even)	8	7	522	266
Nvidia Tensorfloat(Round Even)	8	10	528	272
AMD FP24(Round Even)	7	16	286	156
Pixar PXR24(Round Even)	8	15	538	282
Float(Round Even)	8	23	554	298
Double(Round Even)	11	52	4196	2148
Posit(8, 0, RoundEven)	3	5	26	12
Posit(16, 1, RoundEven)	5	12	114	56
Posit(16, 2, RoundEven)	6	11	226	112
Posit(32, 2, RoundEven)	7	27	482	240
Posit(32, 3, RoundEven)	8	26	960	480
Posit(64, 3, RoundEven)	9	58	1986	992

Table 9: Number Representation Systems Internal Size

Number representation system	Power (W)	Hardware Resources	Delay (ns)	Period (ns)	Frequency (MHz)
Morris(2, 8, RoundZero)	1.001	207 LUT	16.213	16.240	61.57
MorrisHEB(2, 8, RoundZero)	1.007	197 LUT	14.731	14.758	67.75
MorrisUnaryHEB(8, RoundEven)	1.01	434 LUT	18.772	18.799	53.19
MorrisBiasHEB(2, 8, RoundEven)	1.006	186 LUT	14.406	14.433	69.28
Posit(8, 0, RoundEven)	1.012	313 LUT	18.210	18.247	54.80
Morris(3, 16, RoundZero)	1.018	674 LUT	17.728	17.755	56.32
MorrisHEB(3, 16, RoundZero)	1.019	678 LUT	17.441	17.468	57.24
MorrisUnaryHEB(16, RoundEven)	1.02	1258 LUT	20.890	20.918	47.80
MorrisBiasHEB(3, 16, RoundEven)	1.021	605 LUT	16.057	16.085	62.16
Google Brainfloat(Round Even)	1.008	541 LUT	22.830	26.855	37.23
Half Float(Round Even)	1.012	678 LUT	26.168	30.193	33.12
Posit(16, 1, RoundEven)	1.024	932 LUT	22.897	22.921	43.62
Posit(16, 2, RoundEven)	1.015	836 LUT	27.740	27.367	36.54
Nvidia Tensorfloat(Round Even)	1.011	641 LUT	25.957	25.984	38.48
AMD FP24(Round Even)	1.01	988 LUT	28.211	31.246	32.00
Pixar PXR24(Round Even)	1.011	874 LUT	28.763	28.790	34.73
Morris(4, 32, RoundZero)	1.034	1680 LUT	19.828	19.855	50.36
MorrisHEB(4, 32, RoundZero)	1.039	1647 LUT	20.524	20.551	48.65
MorrisUnaryHEB(32, RoundEven)	1.039	3302 LUT	23.914	23.941	41.76
MorrisBiasHEB(4, 32, RoundEven)	1.039	1394 LUT	19.226	19.279	51.86
Float(Round Even)	1.017	1611 LUT	39.017	39.044	25.62
Posit(32, 2, RoundEven)	1.036	2098 LUT	29.740	29.807	33.54
Posit(32, 3, RoundEven)	1.031	1943 LUT	31.802	31.829	31.41
Double(Round Even)	1.027	3537 LUT	49.796	53.917	18.54
Posit(64, 3, RoundEven)	1.068	4053 LUT	33.165	33.257	30.06

Table 10: Addition + Subtraction Generators FPGA Results

Number representation system	Power (W)	Hardware Resources	Delay (ns)	Period (ns)	Frequency (MHz)
Morris(2, 8, RoundZero)	1.002	439 LUT	17.296	17.323	57.72
MorrisHEB(2, 8, RoundZero)	1.003	484 LUT	18.161	18.188	54.98
MorrisUnaryHEB(8, RoundEven)	1.003	916 LUT	26.037	26.064	38.36
MorrisBiasHEB(2, 8, RoundEven)	1.002	454 LUT	20.062	20.089	49.77
Posit(8, 0, RoundEven)	1.003	894 LUT	26.007	26.034	38.41
Morris(3, 16, RoundZero)	1.004	1374 LUT, 1 DSP	36.048	36.075	27.72
MorrisHEB(3, 16, RoundZero)	1.006	1557 LUT, 1 DSP	41.585	41.622	24.02
MorrisUnaryHEB(16, RoundEven)	1.005	2262 LUT, 1 DSP	44.282	44.309	22.56
MorrisBiasHEB(3, 16, RoundEven)	1.007	1456 LUT, 1 DSP	39.705	39.732	25.16
Google Brainfloat(Round Even)	1.005	1527 LUT	33.281	33.308	30.02
Half Float(Round Even)	1.004	1892 LUT, 1 DSP	47.395	47.422	21.08
Posit(16, 1, RoundEven)	1.008	2399 LUT, 1 DSP	58.177	58.204	17.18
Posit(16, 2, RoundEven)	1.01	2178 LUT, 1 DSP	52.312	52.339	19.10
Nvidia Tensorfloat(Round Even)	1.005	1911 LUT, 1 DSP	48.206	48.233	20.73
AMD FP24(Round Even)	1.008	2208 LUT, 1 DSP	54.446	54.473	18.35
Pixar PXR24(Round Even)	1.008	2196 LUT, 1 DSP	53.402	53.429	18.71
Morris(4, 32, RoundZero)	1.014	4124 LUT, 4 DSP	95.787	95.814	10.43
MorrisHEB(4, 32, RoundZero)	1.015	4384 LUT, 4 DSP	115.435	115.462	8.66
MorrisUnaryHEB(32, RoundEven)	1.012	6733 LUT, 4 DSP	113.903	113.930	8.77
MorrisBiasHEB(4, 32, RoundEven)	1.016	3945 LUT, 4 DSP	89.558	89.585	11.16
Float(Round Even)	1.009	5129 LUT, 2 DSP	103.478	103.505	9.66
Posit(32, 2, RoundEven)	1.022	4652 LUT, 4 DSP	87.890	87.917	11.37
Posit(32, 3, RoundEven)	1.022	4784 LUT, 4 DSP	88.823	88.850	11.25
Double(Round Even)	1.037	17989 LUT, 9 DSP	214.545	214.572	4.66
Posit(64, 3, RoundEven)	1.065	13304 LUT, 15 DSP	250.921	250.948	3.98

Table 11: TFPU Generators FPGA Results

consumption and the number of LUTs. This is validated by the 24-bit number representation systems (FP24, PXR24). On 32 bits, MorrisUnaryHEB has almost triple the number of LUTs compared to the others. The internal sizes are similar to 64-bit Posit and IEEE754. A 64-bit MorrisUnaryHEB is considered an over-utilisation of hardware resources, even with the benefits of addition/subtraction exact results, which makes it an excellent inexact accumulator. IEEE754 has the worst results, even if the others have greater internal exponent and fraction sizes. The overhead can only come from decoding and encoding modules. Posit is second to MorrisUnaryHEB in the number of LUTs and slower than all the counterpart Morris divided number representation systems. MorrisBiasHEB keeps its better speed and better resource utilisation on 32-bit. All the Morris hidden exponent bit divided number representation systems have a higher power consumption. On the other hand, IEEE754 has the lowest power consumption. On 64-bit, Posit has higher power consumption and uses more LUTs (14.58%) but has almost double the maximum frequency (30.06 versus 18.54). The following modules are the TFPUs in Table 11. On 8-bit number representation systems, taking into account the number of different values it can represent (Morris) and the dynamic range (MorrisBiasHEB), the best choice is MorrisHEB, followed by Posit, which has double the number of LUTs and almost half the maximum frequency. On 16-bit, it is no surprise that bfloat16 wins easily, being the only one which does not require a DSP. Neither less, bfloat16 is more an inexact number representation system suitable for computation than can correct itself like machine learning (its scope also). For a more general arithmetic approach, MorrisBiasHEB is the best trade-off being close to bfloat16 in hardware performance. Posit and MorrisUnaryHEB require the most LUTs, but at least MorrisUnaryHEB has a better maximum frequency than half-precision IEEE754. Posit with $es = 1$ has worse results than 24-bit IEEE754 systems. In the 32-bit department, it is a surprise that Posit has the better speed results. It also has the high-

Number representation system	Power (W)	Hardware Resources	Delay (ns)	Period (ns)	Frequency (MHz)
Morris(2, 8, RoundZero)	0.993	931 LUT	16.762	16.789	59.56
MorrisHEB(2, 8, RoundZero)	0.993	542 LUT	14.301	14.328	69.79
MorrisUnaryHEB(8, RoundEven)	0.996	2035 LUT	24.456	24.483	40.84
MorrisBiasHEB(2, 8, RoundEven)	0.992	353 LUT	12.896	12.913	77.44
Posit(8, 0, RoundEven)	0.993	609 LUT	20.338	20.365	49.10
Morris(3, 16, RoundZero)	1.054	23215 LUT, 3 DSP	56.353	56.380	17.73
MorrisHEB(3, 16, RoundZero)	1.021	11706 LUT, 3 DSP	52.346	52.373	19.09
MorrisUnaryHEB(16, RoundEven)	1.111	50985 LUT, 3 DSP	68.988	69.015	14.48
MorrisBiasHEB(3, 16, RoundEven)	0.995	1474 LUT, 3 DSP	18.997	19.024	52.56
Google Brainfloat(Round Even)	1.013	13656 LUT	74.764	74.791	13.37
Half Float(Round Even)	1.008	1939 LUT, 3 DSP	39.543	39.570	25.27
Posit(16, 1, RoundEven)	0.999	2595 LUT, 3 DSP	35.008	35.035	28.54
Posit(16, 2, RoundEven)	1.048	5639 LUT, 3 DSP	41.725	41.752	23.95
Nvidia Tensorfloat(Round Even)	1.036	14707 LUT, 3 DSP	56.037	56.064	17.83
AMD FP24(Round Even)	1.012	7493 LUT, 3 DSP	43.264	43.291	23.09
Pixar PXR24(Round Even)	1.025	13055 LUT, 3 DSP	50.964	50.991	19.61
Morris(4, 32, RoundZero)	1.101	50047 LUT, 12 DSP	72.134	72.161	13.85
MorrisBiasHEB(4, 32, RoundEven)	1.049	20149 LUT, 12 DSP	53.383	53.410	18.72
Float(Round Even)	1.043	16313 LUT, 6 DSP	56.072	56.099	17.82
Posit(32, 2, RoundEven)	1.04	14639 LUT, 12 DSP	60.295	60.322	16.57
Posit(32, 3, RoundEven)	1.091	27952 LUT, 12 DSP	67.068	67.095	14.90
Double(Round Even)	1.23	62370 LUT, 27 DSP	82.051	82.078	12.18
Posit(64, 3, RoundEven)	1.267	61636 LUT, 45 DSP	84.637	84.664	11.81

Table 12: KAU Generators FPGA Results

est power consumption of all of them. MorrisBiasHEB has comparable performance (11.16 versus 11.37 and 11.25) using 15.2% fewer LUTs and more negligible power consumption (1.016 vs 1.022). The most power consumption efficient is IEEE754, which uses two fewer DSPs. The 64-bit department has a more energy-efficient and faster IEEE754 than Posit. The KAUs results are in Table 12. The accumulator and fraction sizes for every number representation system are in Table 9. For all number representation systems, the accumulator fraction size is limited to 1024, and the accumulator size is limited to 2048. The number of bits necessary for Morris divided number representation systems are not feasible, except with MorrisBiasHEB. Because their accumulator sizes are caped, the accumulators are inexact. On 8-bit, the high dynamic range of MorrisUnaryHEB, Morris, and Posit gives a high number of LUTs for implementing an exact accumulator. Taking into account the dynamic range, Posit is the best choice. MorrisHEB has a maximum exponent value of 7 (24 Posit), and MorrisUnaryHEB needs more LUTs (2053 vs 609). In the 16-bit department, again, the choice is between MorrisBiasHEB and bfloat16. The first has a lower power consumption and a better maximum frequency. The latter does not need DSPs. Posit remains an alternative solution. The other Morris variants become unfeasible because of their extremely high dynamic range and are excluded from the 32-bit evaluation. On 32-bit, Posit with $es = 2$ is the best choice for power consumption and resource utilisation, and the speed is close to the second choice MorrisBiasHEB. The 64-bit results confirm the high cost of hardware resources for accumulators in this department. Even with this high cost, the maximum frequency has acceptable results (10MHz). The last module tested is GFPU. It needs almost three times the hardware resource of the specific counterpart FPU. These values come from the multiple encoding and decoding modules. The power consumption for 32-bit is similar to TFPU counterparts. The 64-bit GFPU has a higher power consumption of 30.9%. The number of DSPs is similar to their counterpart. The maximum frequency is almost half (6.4 vs 8.6 and 2.2 vs 3.9).

Size (bits)	Power (W)	Hardware Resources	Delay (ns)	Period (ns)	Frequency (MHz)
32	1.021	12008 LUT, 4 DSP	155.646	155.673	6.42
64	1.394	34613 LUT, 16 DSP	449.650	449.677	2.22

Table 13: GFPU Generators FPGA Results

6.2 Insights on Proposed Units Results

The current chapter proposes a hardware research infrastructure for number representation systems. It is validated by adding three new number representation systems: MorrisHEB, MorrisBiasHEB and MorrisUnaryHEB, but also implementing classics like IEEE754, Morris and Posit. Generating procedures for FPUs, KAUs, and GFPUs were implemented. The implementation for adding a new number representation system is reduced to encoding and decoding modules. The benchmarks and the units will be generated without additional effort.

The FPGA evaluation results presented are a baseline from which future improvements can be made. The internal floating-point system can be improved with the optimisations done for the IEEE754 in recent decades. Such improvement will improve every number representation system implemented in the library.

The usage of Posit for implementing Kulisch accumulators was validated. Still, 8-bit MorrisUnaryHEB can be an alternative if there is a need for a higher dynamic range with the cost of hardware resources and power consumption. On 16-bit, MorrisBiasHEB is a more efficient and faster number representation system for a Kulisch accumulator. Google's bfloat16 is an alternative for having less complex hardware. For 32-bit, the choice is also between MorrisBiasHEB and Posit (speed versus resource and energy). The cost of 64-bit Posit and IEEE754 accumulators is high (over 60000LUTS with 27 DSPs or 45 DSPs).

A GFPU is a proposed unit that can have operands and results in different number representation systems of the same bit-width size. The number of LUTs is tripled, and the maximum frequency is halved compared to a specific FPU. The benefits of using different number representation systems depending on the application or parts of the application can overrun the cost.

For the specific FPUs (called TFPUs), Posit shows promising results for 32-bit implementation. The 32-bit MorrisBiasHEB is close to the 32-bit Posit on performance and has better hardware resource utilisation and smaller power consumption. On 16-bit, MorrisBiasHEB is the better solution for general arithmetic, and bfloat16 is better for machine learning utilisation. In the 64-bit department, IEEE754 shows better results than the Posit counterpart.

Optimising the internal floating point in future work is a good path for improving the library's performance. Artificial intelligence and digital signal processing accelerators can be added to the proposed units.

7 NRS INSIDE PROCESSORS

7.1 Processor Evaluation

This section evaluates and analyses the proposed approach based on accuracy, efficiency, estimated area and power. A comparison is done between the original 32-bit FPU of Rocket Chip which claims to implement the IEEE 754 standard (**FP32**) with the proposed *E-PAU* operating with Posit systems of three bit-widths, namely 8-bit with 1-bit exponent denoted by **Posit(8,1)** or **P8**, 16-bit with 2-bit exponent denoted by **Posit(16,2)** of **P16**, and 32-bit with 3-bit exponent denoted by **Posit(32,3)** or **P32**. The proposed *E-PAU* is written in Chisel and integrated with Rocket Chip [1] and uses SiFive's Freedom E310[1] development platform to implement and synthesise the code to run on an Arty A7-100T FPGA.

7.1.1 Benchmarks

The next benchmarks that use floating-point operations are selected to evaluate the approach. These benchmarks are organised into three levels as follows. Level one benchmarks are used to evaluate both the accuracy and efficiency, in terms of cycles, of the proposed *E-PAU* versus the original IEEE 754 FPU of Rocket Chip. These benchmarks represent the computation of well-known mathematical constants using series and sequences. In particular, the following constants are computed: π and e (Euler's number), using numerical series, as shown in Table 14. For π, Leibniz and Nilakantha series [5] is used. Since the Leibniz series converges slowly, it runs for two million iterations. In contrast, the Nilakantha series converges faster and runs for 200 iterations. For e, Euler's series is used, which is fast-converging. Thus, it runs for 20 iterations. In addition to π and e, the $sin(1)$ is computed for 100 iterations. Level two consists of kernels typically used in ML applications [15], as summarised in Table 16. For these kernels, the efficiency of the *E-PAU* versus the FPU in terms of cycles is evaluated. The correctness of the results is checked against reference outputs. Next, a brief description of each kernel is presented. Matrix Multiplication (**MM**) implements the multiplication of two square matrices, which is often used in ML and HPC workloads. In the tests, the matrices accommodate sizes up to $n = 182$. K-means (**KM**) groups a set of multi-dimensional points into k groups, or clusters, based on their Euclidean distance. KM is often used in ML and data analytics applications. K-nearest neighbours algorithm (**KNN**) classifies a multi-dimensional point based on the Euclidean distance to its k nearest neighbours. Linear Regression (**LR**) is a kernel used in ML and data analytics. Multivariate Linear Regression, which consists of

[1] https://github.com/sifive/freedom

matrix and vector operations, is also used. Naive Bayes (**NB**) implements a simple Bayesian model. The Classification (or Decision) Tree (**CT**) kernel is used in ML and data analytics to represent a target variable based on some input attributes. Both the creation (training) and usage (inference) of CT is evaluated. For this, the Iris data set[2] is used as input for level two benchmarks, except MM. This data set consists of $n = 150$ data points with $m = 4$ dimensions representing flowers. These points belong to $k = 3$ classes. Level three of the benchmarks suite represents full-fledged ML models. In the current work, Convolutional Neural Network (CNN) implemented in Caffe and trained on CIFAR-10[3] dataset. While the original CNN has 14 layers and the parameters file has a size of 351 kB, it cannot accommodate the testbed with limited memory size. Hence, only the last four layers of this CNN are taken, starting from *relu3*, and standard C code with static memory allocations is generated. By instrumenting the Caffe framework, all the parameters and the input of *relu3* layer as binary files with *FP32* values were collected. These binaries are converted to all three posit sizes, namely P8, P16, and P32, transformed into objects and link them with the generated C code to get the final RISC-V executable. The validation runs on all 10,000 images of the CIFAR-10 test dataset by running the executables on the FPGA. The prediction results are compared against the reference execution on an x86/64 host.

7.1.2 Accuracy and Efficiency

Level One. The accuracy and efficiency of Posit in comparison with 32-bit, single-precision IEEE 754 floating-point (*FP32*), using level one benchmarks summarized in Table 14 and Table 15 is evaluated. The accuracy is measured in exact fraction digits compared to the reference value. The efficiency represents the number of cycles taken by Rocket Chip running on the FPGA to execute the meaningful section of the program. For Posit, The speedup is computed with respect to the *FP32* execution. The results presented in Table 14 show that *Posit(32,3)* achieves similar or better accuracy compared to *FP32*, while saving up to 23% of the cycles taken by the *FP32* FPU, when π with Leibniz series is computed. On the other hand, [6] shows that any Posit can be accurately represented by an IEEE754 float of a bigger size. That is why 64-bit, double-precision IEEE 754 floating-point is used in the evaluation scripts. Posit operations take fewer cycles to complete. Thus, applications with higher numbers of iterations exhibit better efficiency. For example, *Posit(32,3)* is 30%, 9%, and 3% faster than *FP32* for π Leibniz with two million iterations, π Nilakantha with 200 iterations, and e with 20 iterations, respectively. The analysis revealed that this speedup results from faster multiplication and division operations on Posit. This, in turn, is the result of more straightforward exception and corner case handling in Posit. Intuitively, the efficiency gap grows with the number of iterations. Figure 17 shows that *Posit(32,3)* achieves the same accuracy as *FP32* with fewer cycles as the number of iterations increases. **Level Two.** *Posit(32,3)* and *Posit(16,2)* lead to the same final results as *FP32* when running level two

[2] `https://archive.ics.uci.edu/ml/datasets/iris`

[3] `https://www.cs.toronto.edu/~kriz/cifar.html`

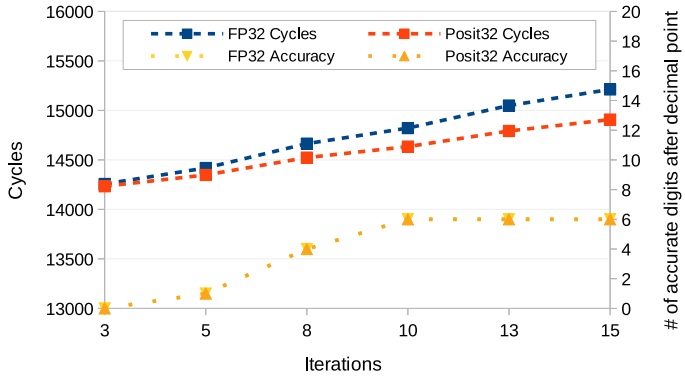


Figure 17: Accuracy and efficiency of Euler's number computation using *FP32* and *Posit(32,3)*

Table 14: Accuracy (Level One Benchmarks)

Application	**Iterations**	**Accuracy** [actual value | number of exact fraction digits]							
		FP32		**Posit(8,1)**		**Posit(16,2)**		**Posit(32,3)**	
π (Leibniz)	2,000,000	3.14159	5	3.5	0	3.14	2	3.14159	5
π (Nilakantha)	200	3.1415929	6	3.125	1	3.141	3	3.1415922	6
e (Euler)	20	2.7182819	6	2.625	0	2.718	3	2.7182817	6
$sin(1)$	100	0.8414709	7	0.78	0	0.8413	3	0.84147098	8

Table 15: Efficiency (Level One Benchmarks)

Application	**Iterations**	**Efficiency** [cycles | speedup]						
		FP32	**Posit(8,1)**		**Posit(16,2)**		**Posit(32,3)**	
π (Leibniz)	2,000,000	216,022,827	166,022,835	1.30	166,022,829	1.30	166,022,830	1.30
π (Nilakantha)	200	57,940	52,937	1.09	52,952	1.09	52,937	1.09
e (Euler)	20	15,598	15,177	1.03	15,177	1.03	15,177	1.03
$sin(1)$	100	16,663	16,270	1.02	16,273	1.02	16,298	1.02

Table 16: Efficiency (Level Two Benchmarks). A grey background means that the result is different from the reference.

Benchmark	**Input Size**	**Efficiency** [cycles | speedup]						
		FP32	**Posit(8,1)**		**Posit(16,2)**		**Posit(32,3)**	
Matrix Multiplication (MM)	n = 182	418,177,415	418,063,614	1.0	418,063,629	1.0	418,177,423	1.0
k-means (KM)	Iris dataset n = 150 m = 4 k = 3	19,150,075	18,879,618	1.01	18,971,747	1.01	19,011,507	1.01
k Nearest Neighbours (KNN)		151,402	138,140	1.10	143,313	1.06	144,136	1.05
Linear Regression (LR)		1,419,794	-	-	-	-	1,398,643	1.02
Naive Bayes (NB)		398,254	407,330	0.98	397,869	1.0	399,893	1.0
Classification Tree (CT)		633,560	101,940	6.2	615,792	1.03	629,936	1.01

benchmarks while saving up to 6% of the cycles, as shown in Table 16. However, LR with both *Posit(8,1)* and *Posit(16,2)* exhibits an out-of-range exception. This is because one of the determinants computed by the program exceeds the range of these two Posit sizes. Moreover, the programs operating with *Posit(8,1)* produce wrong results on all benchmarks, except CT. This shows that small-size Posit is unsuitable for some ML kernels that need high numerical accuracy. This observation is in contrast to some of the related works [2, 3, 13]. However, the evaluation is done on a different dataset, namely the Iris dataset. In the next tests *Posit(8,1)* performs better on a partial CNN. On the other hand, *Posit(16,2)* offers a good alternative to 32-bit floating-point representations. **Level Three.** When compared to the reference execution on an x86/64 host, the CIFAR-10 CNN with *FP32*, *Posit(32,3)* and *Posit(16,2)* running on the FPGA with a Rocket Chip core exhibit the same Top-1 accuracy as the reference model, namely 68.15%. Even *Posit(8,1)* achieves a reasonable accuracy of 62.68%. Regarding speed, all three Posit representations are around 18% faster than the execution with *FP32*. The results with *Posit(16,2)* and *Posit(8,1)* are promising and open up a series of future optimisations. For example, these formats save half and three-quarters of the memory for representing inputs and parameters compared to 32-bit *FP32* or *Posit(32,3)*. Next, by packing two *Posit(16,2)* and four *Posit(8,1)* operands per instruction, the execution time can be reduced by two and four times, respectively. Another source of accuracy loss is due to underflow or overflow at runtime. For example, *prob* layer includes exponentiation, among other operations. On *Posit(8,1)*, exponentiation can easily result in underflow or overflow. To test this hypothesis, the parameters are kept in 8-bit posit format in memory, but the *E-PAU* use *Posit(16,2)* and convert between these two formats at runtime. The result is better than expected because the Top-1 accuracy of this approach is 68.47%, higher than the accuracy of the reference execution on *FP32*. This result confirms the hypothesis that the primary source of the inaccuracy of *Posit(8,1)* is at runtime and shows that using a hybrid approach with posits of different sizes can save memory without losing accuracy.

7.1.3 Resource Utilization

As a proxy to the chip area taken by the implementation, the FPGA resource utilisation of the *E-PAU* is evaluated compared to the original FPU of Rocket Chip. The FPGA resource utilisation of the entire system is evaluated, namely SiFive Freedom E310 with a Rocket Chip core with an FPU/*E-PAU*, running on the Arty A7-100T FPGA. While the results here denote savings in terms of resources from an FPGA perspective, similar or even higher savings in terms of the area will be obtained when the design is implemented on an ASIC [7,14]. Savings in the chip area directly relate to static and dynamic power savings and thus are essential for low-power-constrained applications such as IoT-based edge devices. Table 17 shows the utilisation of the different FPGA resources with respect to both Posit and *FP32* implementations. All the implementations use the same amount of memory resources (Shift-register Look up table – SRL, LUTRAM, and BRAM), indicating that the comparison involves only the modified FPU, with the rest of the system being the same across all implementations. For significant

Table 17: FPGA Resource Utilization of the Entire Rocket Chip on SiFive Freedom E310

Resource	FP32	Posit(8,1)	Posit(16,2)	Posit(32,3)
Logic LUT	29,335	19,367 (-34%)	25,598 (-13%)	38,155 (+30%)
FF	14,756	11,596 (-21%)	12,031 (-19%)	12,951 (-12%)
DSP	15	5 (-67%)	8 (-47%)	19 (+27%)
SRL	58	60	60	60
LUTRAM	924	924	924	924
BRAM	14	14	14	14

savings in area and power without much loss in accuracy *Posit(16,2)* is a viable option that saves almost 50% of the DSPs, which translates to the multiply-accumulate (MAC) units in an ASIC flow. These savings in the area should translate to a 50% drop in power as the MACs account for a higher power compared to flops or other logic [8]. In contrast, *Posit(32,3)* uses 30% more LUTs and 27% utilisation compared to *FP32*. These results are worse than those reported in [4], which needs only 4% more LUTs compared to the FPU, but similar to the ones reported in [12]. On the other hand, the original FPU of Rocket Chip is a work in progress. It may not implement all the corner cases of the IEEE754 standard. Nonetheless, the higher resource utilisation of *Posit(32,3)* may be counterbalanced by its speedup, leading to higher time and energy efficiency compared to *FP32*.

7.2 Insights on Processor Results

This chapter explores the opportunity of replacing the traditional IEEE754 standard with the newly-proposed Posit system [11] in the context of machine learning at the edge. The proposed implementation of an *E-PAU* to replace the original FPU in a RISC-V core is shown. This is the first work to do a thorough evaluation of Posit versus *FP32* to determine (i) whether hardware or software conversion between posit and *FP32* is better, (ii) the time-energy performance for both mathematical and ML kernels, and (iii) insights on choosing the bit-width of Posit for different types of applications. The implementation is evaluated on an FPGA using the SiFive Freedom E310 platform. The accuracy, efficiency in terms of cycles, FPGA resource utilisation and power of three Posit sizes are compared to 32-bit, single-precision IEEE754.

In contrast to previous works [2, 3, 13], the 8-bit Posit is not producing the required accuracy to replace *FP32* in common ML applications. On the other hand, 32-bit Posit is not exhibiting spectacular improvements in terms of efficiency over *FP32*. While they achieve the same or higher accuracy and can speed up the execution when the program has multiplications and divisions, they need around 30% more FPGA resources and use 6% more power compared to *FP32*. However, 16-bit Posit exhibits the best results. Even if they show lower accuracy in scientific computations, they produce correct results for most ML kernels and applications while requiring less area and power compared to *FP32*. For example, *Posit(16,2)* achieves the same Top-1 accuracy as *FP32* on a CIFAR-10 CNN while exhibiting 18% speedup.

8 CONCLUSION

The current thesis approaches a research infrastructure for the discussions regarding number representation systems. The software library proposed reduces the time and effort of proposing a new number representation system to implementing the binary representation and the limits of the numbers it contains. The benchmarks and the libraries implemented over the NRS-SL interface benefit from working with any new number representation system proposed. The thesis offers a statistical methods library, scientific computing benchmarks, a Fast-Fourier-Transform function, and two low-precision machine learning frameworks. All of them were used to evaluate different number representation systems. The hardware generator library improves the path from an idea to multiple hardware prototype units. Implementing a number representation system diminishes the cost of the encoding and decoding modules. Together with complementary libraries from literature [17], the two libraries improve the research infrastructure. The new research infrastructure helps discuss number representation systems by significantly reducing the entry level of proposing and testing a new number representation system.

The software library proposed was used for adding three new number representation systems. MorrisBiasHEB combines the benefits of the classic floating point and tapered floating point and has the first or second best result on the literature benchmarks. It is a significant competitor in general computation for IEEE754. MorrisUnaryHEB is a tremendous tapered floating-point system defined only by its $size$. It has the most exact results on addition (37.6%), the numerous 'golden zone', better decimal accuracy on unary operations, and higher dynamic range than IEEE754, Posit and MorrisBiasHEB. These characteristics make it a formidable opponent in the artificial intelligence domain. Adding a hidden exponent bit started a new path in developing number representation systems.

Nine finite precision number representation systems were evaluated using four scientific computing benchmarks: matrix multiplication, gradient conjugate method, Simpson's Integration, and N-Body simulation. The results on small numbers validate the "golden zone" of tapered floating point systems. The difference between IEEE754 and other number representation systems results is not significant, except N-Body simulation. On N-Body simulation 32-bit Posit ($es = 2$) offers two more correct decimals for low-magnitude numbers and one more for high-magnitude numbers than 32-bit IEEE754. Neither less on scientific computing applications, IEEE754 might remain the standard without significant benefits from the hardware implementation of other number representation systems.

The statistical methods library offers 14 statistical methods. The 16-bit Posit ($es = 1$) shows results similar to 32-bit counterparts. On the other hand, 16-bit IEEE754 (half-precision) can

not produce a valid result on 6 of them. Even when it produces one, it has a lower decimal accuracy than 16-bit Posit. The decimal accuracy degrades with the number of entries and higher values of the data set. The reader must understand that contrary to the beliefs, a result of a statistical method on a more extensive data set might be further away from the correct answer than expected. The number representation system used has an impact on it. Posit offers better decimal accuracy than IEEE754 on all methods, except the ones based firmly on variance, like the T-test and variance itself.

The two low-precision machine learning frameworks help intelligent embedded systems by reducing the storage size of a deep neural network. They reduce the size by 66.75% respectively 76.68% with an accuracy degradation lower than 2%. The first framework (LPML) uses knowledge distillation and layer precision optimisation with a limited set of number representation systems: generic Fixed-Point, 8-bit Posit ($es = 0$), 16-bit Posit ($es = 1$), 32-bit Posit ($es = 2$), 32-bit IEEE754). The accuracy increase of using knowledge distillation instead of classic training is validated by a value of 0.87%. The Resnet34 network (accuracy: 95.23%) is used as the teacher network. The student network Resnet18 obtains an accuracy of 94.98% after training (classic accuracy: 94.11%). Layer precision optimisation reduces the storage size of the Resnet18 network by 43.41%. The final accuracy is 93.25%. The memory reduction compared to Resent34 is 66.75%. The second framework (CPML) uses uniform and layer precision optimisation with number representation systems implemented in NRS-SL. Uniform precision optimisation reduces the size of Resent18 (accuracy: 57.01%) with at least 62.25% (Morris) and a maximum of 75% (IEEE754) without losing more than 1% in accuracy. The next step of layer precision optimisation is not significantly efficient, and the overhead of different layer precision might surpass the benefits.

The hardware generator library generates functional units as FPUs, KAUs and unary/binary modules. For number representation systems with a high dynamic range, inexact accumulator units are a solution if the insignificant error is accepted. The general floating point unit (GFPU) can operate simultaneously with multiple number representation systems as the operands or the result. The hardware cost is triple, and the speed is half compared to a specific number representation system FPU. The benefits of using a particular number representation system for a specific part of the application might surpass the cost. In the 16-bit department, Google's bfloat16 shows the best hardware performance in a floating point unit (fewer LUTS, fewer DPSs, better speed). Its scope is machine learning applications. For general computing, MorrisBiasHEB and Posit are the best alternatives. The best choice in the 32-bit department is between MorrisBiasHEB and Posit. For accelerators which use mainly addition and multiplication, MorrisBiasHEB is the best decision possible. It has good accuracy, the best computation time, and the smallest necessary hardware resources. The Kulisch accumulator unit for 16-bit MorrisBiasHEB has almost double the maximum frequency (52.56 MHz versus 28.54 MHz) and nearly one-quarter fewer LUTS (1474 versus 1939) than the second-best in that category.

A complete hardware-software system to test a number representation system can give a

different perspective. Posit tested its performance against IEEE754 on a RochetChip CPU [1]. Posit validated the expectation of being faster than IEEE754, but with the cost of hardware resources and power consumption. The 16-bit Posit shows an 18% speedup on the third level benchmark, and it requires fewer hardware resources and consumes less energy.

New, alternative or optimised operations will be added to the software library in future work. The category of interval number representation systems is another path worth going, which can offer solutions for the problems which the classic number representation system can not solve. The pallet of units for the hardware generator library will grow by adding artificial intelligence accelerators, digital signal processing accelerators or GPUs. The statistical methods library needs to add more methods to be tested. The scientific computing benchmarks can increase their complexity to offer better insights. The low-precision frameworks allow testing on more complex networks that are not dedicated to embedded systems. The most significant future work will be creating and evaluating more hardware-software systems to develop a proper opponent for IEEE754. MorrisBiasHEB has excellent hardware and general computation performance, which needs to be investigated in a SoC (System on the Chip) in future work. On another path, MorrusUnaryHEB might change the domain of artificial intelligence computation.

BIBLIOGRAPHY

[1] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, et al. The Rocket Chip Generator. *University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[2] Zachariah Carmichael, Hamed F Langroudi, Char Khazanov, Jeffrey Lillie, John L Gustafson, and Dhireesha Kudithipudi. Deep positron: A deep neural network using the posit number system. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1421–1426, 2019.

[3] Zachariah Carmichael, Hamed F Langroudi, Char Khazanov, Jeffrey Lillie, John L Gustafson, and Dhireesha Kudithipudi. Performance-efficiency trade-off of low-precision numerical formats in deep neural networks. In *Proceedings of the Conference for Next Generation Arithmetic*, pages 1–9, 2019.

[4] Rohit Chaurasiya, John L. Gustafson, Rahul Shrestha, Jonathan Neudorfer, Sangeeth Nambiar, Kaustav Niyogi, Farhad Merchant, and Rainer Leupers. Parameterized Posit Arithmetic Hardware Generator. In *Proc. of 36th IEEE International Conference on Computer Design*, pages 334–341, 2018.

[5] Jose Cintra. Calculating the Number PI Through Infinite Sequences. http://archive.today/2Nf1G, 2014.

[6] Florent De Dinechin, Luc Forget, Jean-Michel Muller, and Yohann Uguen. Posits: the good, the bad and the ugly. In *Proceedings of the Conference for Next Generation Arithmetic*, pages 1–10, 2019.

[7] Andreas Ehliar and Dake Liu. An ASIC Perspective on FPGA Optimizations. In *Proc. of International Conference on Field Programmable Logic and Applications*, pages 218–223, 2009.

[8] James Garland and David Gregg. Low Complexity Multiply-accumulate Units for Convolutional Neural Networks with Weight-sharing. *ACM Transactions on Architecture and Code Optimization*, 15(3):1–24, 2018.

[9] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991.

[10] John L. Gustafson. *The End of Error: Unum Computing*. Chapman & Hall/CRC Computational Science. Taylor & Francis, 2015.

[11] John L Gustafson and Isaac T Yonemoto. Beating floating point at its own game: Posit arithmetic. *Supercomputing Frontiers and Innovations*, 4(2):71–86, 2017.

[12] M. K. Jaiswal and H. K. . So. Universal Number Posit Arithmetic Generator on FPGA. In *Proc. of Design, Automation Test in Europe Conference Exhibition*, pages 1159–1162, 2018.

[13] Jeff Johnson. Rethinking floating point for deep learning. *arXiv preprint arXiv:1811.01721*, 2018.

[14] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, 2007.

[15] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. PuDianNao: A Polyvalent Machine Learning Accelerator. In *Proc. of 20th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 369–381, 2015.

[16] Robert Morris. Tapered floating point: A new floating-point representation. *IEEE Transactions on Computers*, 100(12):1578–1579, 1971.

[17] E. Theodore L. Omtzigt, Peter Gottschling, Mark Seligman, and William Zorn. Universal Numbers Library: design and implementation of a high-performance reproducible number systems library. *arXiv:2012.11011*, 2020.