ANALYSIS OF SCIENTIFIC COMPUTING ALGORITHMS UNDER DIFFERENT NUMBER REPRESENTATION SYSTEMS

Ștefan-Dan Ciocîrlan, Radu-Georgian Soare, Nicolae Țăpuș, Ebru Resul, Răzvan-Victor Rughiniș[3]

Scientific computing-based applications need a high load of computations. These computations are closely linked to the number representation system (NRS). A benchmark for decimal accuracy, storage space, computation time and energy consumption using multiple NRSs is needed for the base scientific computing algorithms. Eleven NRSs were evaluated under four types of benchmarks: matrix multiplication, solving a linear system of equations using the gradient conjugate method, integral calculation using Simpson's formula and N-body simulation. The results validate the posit "golden zone" and offer a perspective on different NRSs accuracy related to input ranges. IEEE754 performed well regarding decimal accuracy in all four benchmarks (top 3).

Keywords: Number representation systems, IEEE754, Scientific Computing, N-body, Posit

1. Introduction

Scientific computing is an interdisciplinary field that combines technologies such as computer science and data science and it is used in physics, mathematics, and biology. Physical phenomena are modelled, simulations are run and the data obtained is computed mathematically and statistically to reach the optimal parameters of a system or to be able to make predictions about the evolution of a phenomenon. This comes in handy when solving a problem that cannot be determined experimentally (e.g. evolution of the weather), the task is too dangerous (e.g. classification of toxic chemical elements), or when simulations that do not allow physical trial and error are needed to get to an optimal result (e.g. designing a satellite).

Regardless of the field from which the problem comes, it can be reduced to numbers and operations. The scope is to have values with the best precision to obtain measurements and results as accurate as possible. For a long time, the impediment to scientific computing was the limited computing power available to scientists. With the exponential increase in computing power, the bottleneck moved toward the accuracy with which the computers can perform computations and how much their energy consumption is for doing them.

[3]University POLITEHNICA of Bucharest, Dept. of Computer Science, Romania, e-mail `stefan_dan.ciocirlan@upb.ro`

IEEE754 is the standard for the last 37 years. At this time, academics and engineers had asserted its effectiveness and had found out the use cases where it under-performs.

John Gustafson analysed the IEEE754 [1] in his book [8] and he proposed new number representation systems as alternatives. The academia and industry did not queue to adapt the proposed NRSs: Unum type 1 and Unum type 2. The next proposed NRS by Gustafson, Posit [12], is more familiar with the IEEE754 standard and had higher success in adoption. Academics started to analyse its attributes in different domains as artificial intelligence [13, 3, 7, 4, 19], **specific** scientific computing applications [17, 18, 20], hardware implementation [25, 10, 21, 11], digital signal processing [24, 14]. In terms of low-precision IEEE754 has half-precision. Industry came with their alternative NRSs: bfloat16 (Google [2]), TensorFloat-32 (NVIDIA [16]), fp24 (AMD), PXR24 (Pixar).

Currently, scientific computing is dominated by the IEEE754 standard. For other NRSs, such as Posit, to be considered as a direct replacement, proper research, and extensive testing are needed. The first step is to analyse where a NRS is bringing improvements and where it under-performs compared to the current standard. This article proposes a benchmark with base scientific computing algorithms and analyses their performance under different NRSs. The benchmarks use the Scala number representation systems software library (NRS-SL) [1]. The library offers the most diverse palette of NRSs with the possibility of custom modifications in terms of sizes, in contrast with other NRSs libraries: SoftPosit [2], sfpy [3], Flexfloat [23], GMP [4], FloPoCo [6].

Tested algorithms within the application were chosen from relevant fields of scientific computing (mathematics and physics): matrix multiplication: one of the most common operations in mathematics; Solving linear system of equations: implemented using the gradient conjugate method; definite integral computing: implemented using the approximation based on Simpson's rule; N-body Simulation.

Eleven NRSs were used for evaluation under all of the above four algorithms. The decimal accuracy, bit-wise size, and computation time were considered. The Posit standard offered better accuracy in most cases. An interesting finding is the performance of low-precision NRSs.

The presented article is structured in six sections. The first section explains the motivation, introduces the concepts of scientific computing, NRSs, decimal accuracy and briefly presents the solution proposed. The second section presents the terminology from the NRS-SL. In the third section, the implementation of the four base scientific computing algorithms is found. The

[1]https://repsy.io/mvn/sdcioc/nrs/ro/upb.nrs

[2]https://gitlab.com/cerlane/SoftPosit

[3]https://pypi.org/project/sfpy/

[4]https://gmplib.org/

methodology is given in the fourth section. In the fifth section, the results of the benchmarks are discussed. The last section presents the conclusion and the future work of the article.

2. **Terminology**

Since the field of scientific computing has grown, the problem of optimizing the models used has arisen and one of the directions that development is now focusing on is the NRS used by models. There is a need for comparing the other NRSs with the current standard. There are multiple libraries that offer a diversity of NRSs like SoftPosit, sfpy, Flexfloat [23], GMP, FloPoCo [6]. For the current implementation, NRS-SL was chosen for its big palette of NRSs and the possibility of custom sizes and attributes. Also, every time a NRS will be added to the library the benchmarks for it can be run without writing specific code. The library uses the next terminology for the name of the NRS: precision, the existence of rounding and the mathematical set used. Precision can be infinite (IP) or fixed (FP). A NRS can have rounding (R) or not (NR). A NRS with rounding must also have set a rounding type. The NRS set can be N (unsigned integers), Z (integers), Q (fractional numbers), FixedP (fixed-point numbers), FloatP (floating point numbers, different from IEEE754 by not having subnormals), IEEE754 (the current standard), and Posit. IEEE754 half-precision, bfloat, TensorFloat-32, AMD's FP24 and PXR24 are considered derived from IEEE754. Fixed precision is the arithmetic used in most of the existing hardware and limits the binary size of the numbers usually to a value range between 8 and 64 (mostly a power of 2). Infinite precision on the other hand is a concept which states that the computations are only limited by the available memory, so in theory, with an infinite amount of memory, using this precision would translate to a result with the maximum accuracy possible under the chosen NRS. FixedP, FloatP, IEEE754 and Posit are the same in terms of accuracy in infinite precision. The n-order root, trigonometric, hyperbolic, logarithm, exponential and power functions are precise until a given precision of iterations so infinite computation can be avoided if the result of the operation is an irrational number (e.g. $\sqrt{2}$). There are also some number representations which are derived from IEEE754 by changing the bit layout: IEEE754 half-precision (5 exponent bits and 10 fraction bits), bfloat16 (8 exponent bits and 7 fraction bits), NVidia's TensorFloat-32 (8 exponent bits and 10 fraction bits), AMD's fp24 format (7 exponent bits and 16 fraction bits), Pixar's PXR24 format (8 exponent bits and 15 fraction bits).

3. **Tested Algorithms**

3.1. **Matrix multiplication algorithm**

We have two matrices: A of size $N_A \times M_A$, B of size $N_B \times M_B$ ($M_A = N_B$) and we want to compute the result (C) of their multiplication.

Algorithm 3.1 Matrix multiplication

```
Require: $N_A, M_A, N_B, M_B, A, B$
  for $i \leftarrow 1$ to $N_A$ do
      for $j \leftarrow 1$ to $M_B$ do
          $C_{i,j} = 0$
          for $k \leftarrow 1$ to $N_B$ do
              $C_{i,j} + = A_{i,k} \times B_{k,j}$
          end for
      end for
  end for
  return $C$
```

3.2. Gradient conjugate method algorithm

The gradient conjugate method is an algorithm commonly used for a particular type of linear systems, namely those that have positive definite matrices. For solving a linear system of equations with this method the following equation system needs to be solved:

$$Ax = b \tag{1}$$

Where A - matrix of size $N \times N$, symmetric and positive definite, known; b - column vector, known;x - column vector, the unknown system solution.

Algorithm 3.2 Gradient Conjugate Method [22]

```
Require: $x_0, A, b$
  $r_0 \leftarrow b - A\ x_0$
  $p_0 \leftarrow r_0$
  for $k = 0$, $k \leftarrow k + 1$, while $k \leq N$ do
      $\alpha_k \leftarrow \frac{r_k^T \times r_k}{p_k^T \times A \times p_k}$
      $x_{k+1} \leftarrow x_k + \alpha_k \times p_k$
      $r_{k+1} \leftarrow r_k - \alpha_k \times A \times p_k$
      $\beta_k \leftarrow \frac{r_{k+1}^T \times r_{k+1}}{r_k^T \times r_k}$
      $p_{k+1} \leftarrow r_{k+1} + \beta_k \times p_k$
  end for
  return $x_{k+1}$
```

3.3. Integral calculation algorithm

To approximate the value of a definite integral, we chose to use Integration by Simpson's formula. We want to approximate the value of the integral:

$$\int_a^b f(x)\,dx \tag{2}$$

Algorithm 3.3 Integration by Simpson's formula [15]

```
Require: $f, a, b$
  $N \leftarrow 1000$
  $h \leftarrow \frac{b-a}{N}$
  $s \leftarrow f(a) + f(b)$
  for $k \leftarrow 1$ to $N - 1$ do
      $x \leftarrow a + h \times k$
      if $k$ is even then
          $s \leftarrow s + f(x) \times 2$
      else if $k$ is odd then
          $s \leftarrow s + f(x) \times 4$
      end if
  end for
  $s \leftarrow s * \frac{h}{3}$
  return s
```

3.4. N-Body simulation algorithm

N-Body simulation is done to observe how N particles are moving under the influence of gravity and under the attraction force of the other particles, tracking the evolution of their velocity and position in time. Assumptions: the initial velocity and position of the particles are known, and acceleration is uniform during an iteration.

$$v_{i+1} = v_i + a_x \times \Delta_t - \text{velocity change after an iteration} \tag{3}$$

$$p_{i+1} = p_i + v_{i+1} \times \Delta_t - \text{position change after an iteration} \tag{4}$$

$$F_{xy} = G\frac{m_x \times m_y}{r_{xy}^2} - \text{newton's law of universal gravitation} \tag{5}$$

$$F_x = \sum_{y \in B - \{x\}} F_{xy} - \text{total force acting on the body } x \tag{6}$$

4. Benchmarks methodology

The tests implement algorithms that solve problems often encountered in scientific computing and numerical methods. The benchmarks evaluate the number of exact decimals (decimal accuracy), decimal error, and computation time. Every algorithm was implemented in such a way that it can be run using any NRS implemented in the library. Firstly, the algorithm will run using the reference NRS (one representation that is considered the absolute truth) so that the other NRSs can be compared to it. Then every other NRS within the library will be used by the same algorithm in the same conditions. With the results calculated for every NRS, the metrics can be computed. Computation

Algorithm 3.4 N-Body Simulation Algorithm

```
Require: $N, M, p_0, v_0, m$ ▷ Number of particles, Number of iterations, initial position of particles, initial velocity of particles, mass of particles
  for $i \leftarrow 1$ to $M$ do
      for $j \leftarrow 1$ to $N$ do
          $F_{i,j} \leftarrow get\_total\_force(m, p_{i-1})$          ▷ Total force exerted on the $j^{th}$ particle
          $a_{i,j} \leftarrow \frac{F_{i,j}}{m_j}$
          $v_{i,j} \leftarrow v_{i-1,j} + a_{i,j} \times$ iteration time step          ▷ Update velocity
          $p_{i,j} \leftarrow p_{i-1,j} + v_{i,j} \times$ iteration time step          ▷ Update position
      end for
  end for
```

time is computed during the actual algorithm execution. Decimal error is computed using the formula suggested in [9] to the detriment of absolute error and relative error which were not considered relevant enough for this kind of test:

$$\text{decimal error} = |\log_{10} \frac{x_{computed}}{x_{exact}}|$$

Where x_{exact} is the value computed using the reference NRS and $x_{computed}$ is the value obtained using the currently tested NRS. Decimal accuracy is computed using the formula:

$$\text{decimal accuracy} = |\log_{10} \frac{1}{\text{decimal error}}|$$

Next, the general steps for every benchmark are presented: (1) random input values are generated, except for Simpson's Integration where the functions are chosen and the a, b and N are set; (2) the algorithm is run under the reference NRS (IP_NR_Q or FP_R_Q); (3) the algorithm is run under every tested NRS and the results are converted to reference NRS. The decimal error and decimal accuracy are computed.

For matrix multiply the specific steps are: (1) a list of randomly generated matrix pairs of type 64-bit IEEE754, size $N \times N$ with values in the range of $[-10^{power}, 10^{power}]$ (where *power* is a number between 0 and max power); (2) Every element of the result matrix is iterated and the decimal error and decimal accuracy are computed with the same position element in the reference result matrix; (3) the average is computed on the decimal accuracy and the decimal error matrices.

In the conjugate gradient method experiment the next steps are taken: (1) a list of randomly generated tuples (A, b, x_0) of type 64-bit IEEE754 with values in the range of $[0, 10^{power}]$ (where *power* is a number between 0 and max power); the next steps are similar to matrix multiply steps (2) and (3).

For Simpson's Integration Rule first step (1) is to choose the functions, the range $[a, b]$ and the number of steps N. (2) A vector with the result for every function is computed. With the tested NRS vector and reference NRS vector, two vectors for decimal accuracy and decimal error are computed. The last step (3) is to add all the elements inside a vector.

In N-body simulation there are the next steps: (1) the number of particles is set, the number of iterations is set, the iteration time step is set, a list of particles is randomly generated by initial position, initial velocity and mass; (2) the next formula is used for computing "normalised" velocity $\sqrt{velocity.x^2 + velocity.y^2}$ using IP_NR_Q; (3) the decimal accuracy and decimal error for every particle is computed. The average of the results is done; (4) the euclidean distances between tested NRS and reference NRS velocities are computed. The average of the results is done.

5. 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 (15 bits integer and 16 bits fractional) called FixedP, infinite precision fixed-point called IP_NR_FixedP, infinite precision floating-point called IP_NR_FloatP. The rounding method used was rounding to the nearest tie to even.

5.1. Matrix multiplication benchmark

For matrix multiplication benchmark the reference NRS is IP_NR_Q, $N_A = N_B = M_A = M_B = 3$, MAX POWER $= 8$. Figure 1 confirms the

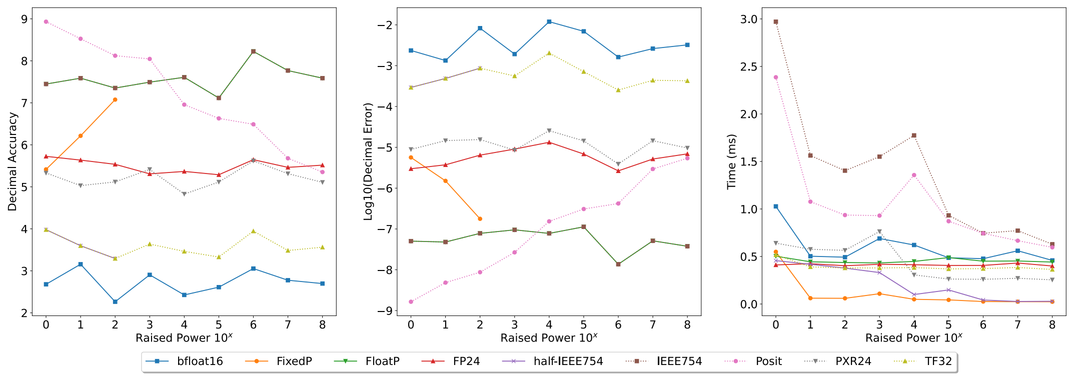

FIGURE 1. Decimal accuracy, decimal error and computation time for matrix multiplication benchmark

existence of the Posit Golden Zone [5]. It can be seen that Posit has higher accuracy compared to the following most accurate representation when the elements of the matrix are generated in the range $[10^{-3}, 10^{3}]$. This means that the resulting elements are in the range $[10^{-6}, 10^{6}]$ because in the worst case two numbers of magnitude order 3 are multiplied. Another thing worth noting is the comparison between the fixed-point and floating-point. FloatP cannot be seen on the graph because it overlaps with IEEE754 which makes it the representation with the second-best accuracy. On the other hand, FixedP ends abruptly because after the values become excessively high it cannot represent them so it returns **not representable** (NR). The good results obtained by FloatP to the detriment of FixedP are related to the large number range from which the input data was chosen which FloatP handles much better. In Figure 1, the half-IEEE754 overlaps with TF32 in the range $[10^{-2}, 10^{2}]$ even though the first one uses 3 bits less than the second one. However, at one point half-IEEE754 could no longer represent the values, so it returned NR. Regarding the error, the values are inversely proportional to those of accuracy, which means that high accuracy results in a low error. Regarding the computing time, there are no significant differences between the representations even though we can see FixedP as an outlier. That is explained by the fact that it returned NR which means that from that point every operation would result in NR, so no heavy computing was done. IP_NR_FloatP and IP_NR_FixedP were not plotted on the chart because they obtained the same result as the reference NRS, which implies that the accuracy was theoretically infinite and the error was 0. This happens because the input is in 64-bit IEEE754 and this means that in fractional NRS they are numbers with a denominator power of two. If two of these numbers are multiplied, the result is also a number with a denominator power of two.

5.2. Conjugate gradient method benchmark

For conjugate gradient method benchmark, the reference NRS is 1024-bit FP_R_Q (512 bits integer nominator and 512 bits unsigned integer denominator), $N = 5$, MAX POWER = 3. Because multiple matrix multiplications are executed on every iteration of the algorithm, the magnitude order of the values used grows fast during the execution. This is the reason 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 numbers stands out even more in 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 a little harder to predetermine. After multiple iterations of the algorithms as can be seen in the Figure 2, 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

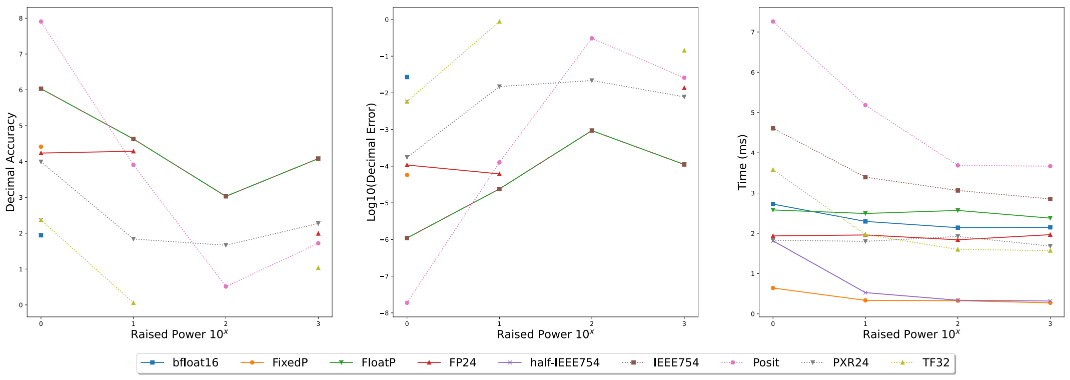

FIGURE 2. Decimal accuracy, decimal error and computation time for conjugate gradient method benchmark

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. IP_NR_FloatP and IP_NR_FixedP were not plotted on the chart because they return NR regardless of the order of the numbers used.

5.3. Simpson's Integration benchmark

For Simpson's integration benchmark the reference NRS is 1024-bit FP_R_Q (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

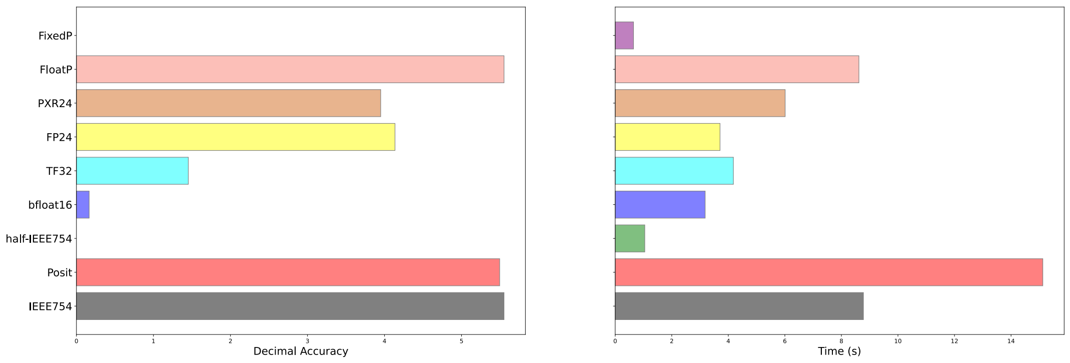

FIGURE 3. Decimal accuracy and computation time for Simpson's integration benchmark

calculations, the accuracy does not differ too much between the representations with the same number of bits (Figure 3 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.

5.4. N-body simulation benchmark

For N-body simulation benchmark the reference NRS is 1024-bit FP_R_Q (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 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 4 suggests the superior-

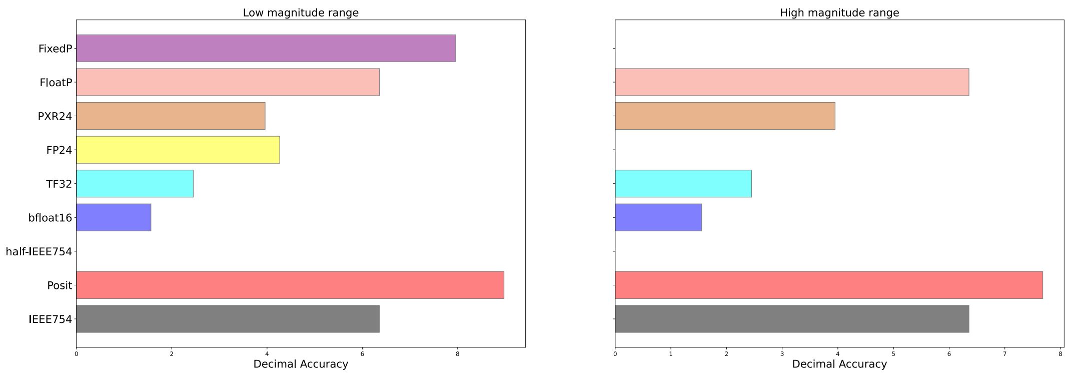

FIGURE 4. Decimal accuracy fro N-body simulation benchmark

ity 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. Posit kept its high accuracy compared with the other NRSs even though the values used were theoretically outside of its best performance range.

6. Conclusion

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

The gold zone of Posit [5] is 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 interesting 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 article.

The replacement of the current standard with a new NRS might seem necessary given the results presented. However, the lack of hardware implementation and the lack of industry adoption will delay the process.

Acknowledgment

This work was partly supported by the Bitdefender's University PhD Grants Program 2019-2022 and by the Google IoT/Wearables Student Grants 2022. The results presented in this article has been funded by the Ministry of Investments and European Projects through the Human Capital Sectoral Operational Program 2014-2020, Contract no. 62461/03.06.2022, SMIS code 153735.

REFERENCES

[1] Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[3] 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 *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1421–1426. IEEE, 2019.

[4] 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 2019*, pages 1–9, 2019.

[5] 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 2019*, pages 1–10, 2019.

[6] Florent de Dinechin and Bogdan Pasca. Designing custom arithmetic data paths with FloPoCo. *IEEE Design & Test of Computers*, 28(4):18–27, July 2011.

[7] Hasan Genc, Ameer Haj-Ali, Vighnesh Iyer, Alon Amid, Howard Mao, John Wright, Colin Schmidt, Jerry Zhao, Albert Ou, Max Banister, et al. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. *arXiv preprint arXiv:1911.09925*, 3:25, 2019.

[8] John L Gustafson. *The end of error: Unum computing.* Chapman and Hall/CRC, 2017.

[9] John L. Gustafson. Posit arithmetics, 2017.

[10] Manish Kumar Jaiswal and Hayden K-H So. Architecture generator for type-3 unum posit adder/subtractor. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[11] Manish Kumar Jaiswal and Hayden K-H So. Universal number posit arithmetic generator on fpga. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1159–1162. IEEE, 2018.

[12] Isaac Yonemoto John L. Gustafson. Beating floating point at its own game: Posit arithmetic, 2017.

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

[14] Manash Kant and Rajeev Thakur. Implementation and performance improvement of posit multiplier for advance dsp applications. In *2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pages 1730–1736. IEEE, 2021.

[15] Autar K Kaw. Numerical methods with applications, chapter 07.03, 2008.

[16] P Kharya. Tensorfloat-32 in the a100 gpu accelerates ai training hpc up to 20x. *NVIDIA Corporation, Tech. Rep*, 2020.

[17] M Klöwer, PD Düben, and TN Palmer. Number formats, error mitigation, and scope for 16-bit arithmetics in weather and climate modeling analyzed with a shallow water model. *Journal of Advances in Modeling Earth Systems*, 12(10):e2020MS002246, 2020.

[18] Milan Klöwer, Peter D Düben, and Tim N Palmer. Posits as an alternative to floats for weather and climate models. In *Proceedings of the Conference for Next Generation Arithmetic 2019*, pages 1–8, 2019.

[19] Jinming Lu, Chao Fang, Mingyang Xu, Jun Lin, and Zhongfeng Wang. Evaluations on deep neural networks training using posit number system. *IEEE Transactions on Computers*, 70(2):174–187, 2020.

[20] Christofer Nolander and Amanda Strömdahl. A comparative study on the accuracy of ieee-754 and posit for n-body simulations, 2021.

[21] Artur Podobas and Satoshi Matsuoka. Hardware implementation of posits and their application in fpgas. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 138–145. IEEE, 2018.

[22] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[23] Giuseppe Tagliavini, Andrea Marongiu, and Luca Benini. Flexfloat: A software library for transprecision computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(1):145–156, 2018.

[24] E Ternovoy, Mikhail G Popov, Dmitrii V Kaleev, Yurii V Savchenko, and Alexey L Pereverzev. Comparative analysis of floating-point accuracy of ieee 754 and posit standards. In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 1883–186. IEEE, 2020.

[25] Yohann Uguen, Luc Forget, and Florent de Dinechin. Evaluating the hardware cost of the posit number system. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 106–113. IEEE, 2019.