THE EFFECTS OF NUMERICAL PRECISION IN SCIENTIFIC APPLICATIONS

Raul Murillo
Alberto A. Del Barrio
Guillermo Botella

Department of Computer Architecture and Automation
Complutense University of Madrid
Madrid 28040, SPAIN
{ramuri01, abarriog, gbotella}@ucm.es

ABSTRACT

Throughout the last decades, multiple formats have appeared as alternatives to the IEEE 754™ standard for floating-point arithmetic. The use of low-precision formats, such as bfloat16, or of new arithmetic systems, such as posit™, has an increasing interest not only in areas like machine learning or HPC, but also general-purpose numerical algorithms can benefit from those non-standard formats, reducing computation time, power consumption or memory requirements. However, dedicated hardware for new arithmetics is not always available, and simulating different numerical precisions is essential to experiment with such alternative formats not yet implemented in hardware. In this work, we examine via software emulation the effects that different arithmetic formats and numerical precisions have in a wide variety of scientific applications. The experiments reveal that, under the same bitwidth, posit arithmetic provides up to two orders of magnitude less error than floating-point format.

Keywords: floating-point, posit, computer arithmetic, low-precision.

1 INTRODUCTION

Historically, most scientific applications have been built on top of the IEEE 754 standard for floating-point arithmetic (IEEE Computer Society 2019), which has been for decades the format for representing real numbers in computers. Nevertheless, the IEEE 754 possesses some problems that are inherent to its construction, such as rounding, reproducibility, the existence of signed zero, the denormalized numbers or the wasted patterns for indicating Not a Number (NaN) exceptions (Goldberg 1991). All in all, IEEE 754 is far from being perfect, as different CPUs may produce different results, and all these special cases must be dynamically checked, which increases the hardware cost of IEEE 754 units.

Such inefficiencies in the IEEE 754 standard, combined with the end of Dennard scaling and Moore's Law and the deceleration of performance gains for general-purpose processors have lead to the development of domain-specific languages and architectures through the last decade (Hennessy and Patterson 2019). In conjunction with these new architectures, several custom number systems and formats, such as the High-Precision Anchored (HPA) numbers from ARM, the Hybrid 8-bit Floating Point (HFP8) format from IBM, bfloat16, and many more have been considered as an alternative to IEEE 754-2019 compliant arithmetic (Guntoro et al. 2020), which also includes a 16-bit IEEE 754 version. Nonetheless, the appearance of the disruptive posit™ arithmetic (Gustafson and Yonemoto 2017) in 2017 has shaken the board. While the

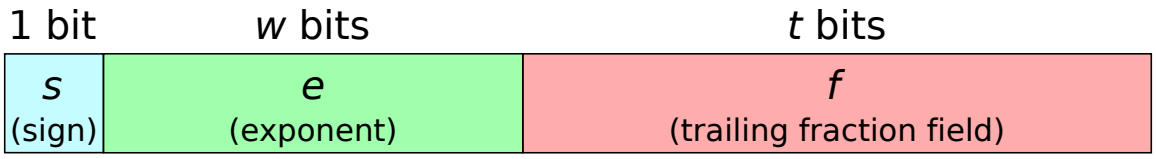


Figure 1: General floating-point binary encoding.

aforementioned approaches (except for the half-precision IEEE floats) are vendor-specific, posits aim to be standard. This novel way of representing reals mitigates and even solves the aforementioned IEEE 754 drawbacks.

Reducing the numerical precision of data and computation is extremely effective in accelerating scientific computation workloads. It can decrease the required amount of memory, which allows shortening the execution time and hardware requirements, two key factors in the design of domain-specific accelerators. Also alternative arithmetic formats can accelerate general purpose scientific computing, or provide more accurate results. In this context, it is essential to evaluate the impact that a certain number system or precision has on the result of the calculation to be performed. Naturally, the appropriate hardware is not always available, and developing new hardware is always much more expensive than performing software simulations. Therefore, the relevance of validating the accuracy of such non-standard formats at the software level is clear.

This work presents a detailed comparison of multiple scientific applications and the results obtained when computed under different precision and arithmetic formats other than the IEEE 754 standard. In particular, low-precision formats of floating-point and posit arithmetic are simulated in order to validate their accuracy in computer-intensive applications.

2 BACKGROUND

2.1 Floating-Point Arithmetic

The floating-point arithmetic format is used in computers to represent and operate with real numbers in computers. It closely resembles the traditional scientific notation, as floating-point numbers (or floats) consist of three parts, as shown in Figure 1: a sign s, an exponent e in a given base, and a fraction part f. To derive the value of the floating-point number x, the fraction is multiplied by the base (usually two in computers) raised to the power of the exponent, as Equation (1) shows:

$$x = (-1)^s \times 2^e \times f. \tag{1}$$

The current standard for binary floating-point representation is the IEEE 754 (IEEE Computer Society 2019), and it mainly defines the length of the aforementioned fields, the required operations, rounding rules, and exceptions, among other details. Standard floating-point numbers include a bias to the exponent e, and an implicit hidden bit (usually equal to 1) to the fraction f. Also, it considers subnormal numbers (which add a leading zero to the fraction when the exponent bits are all 0's) and NaN exceptions (when the exponent bits are all 1's). The standard defines binary floating-point formats for 16 bits (*half precision*, with 5 exponent bits), 32 bits (*single precision*, with 8 exponent bits), 64 bits (*double precision*, with 11 exponent bits), 128 bits (*quadruple precision*, with 15 exponent bits) and 256 bits (*octuple precision*, with 19 exponent bits). Non-standard formats might define different lengths for the exponent and fraction bits (like bfloat16, with 8 exponent bits and 7 fraction bits), and usually do ignore subnormal numbers and NaN cases.

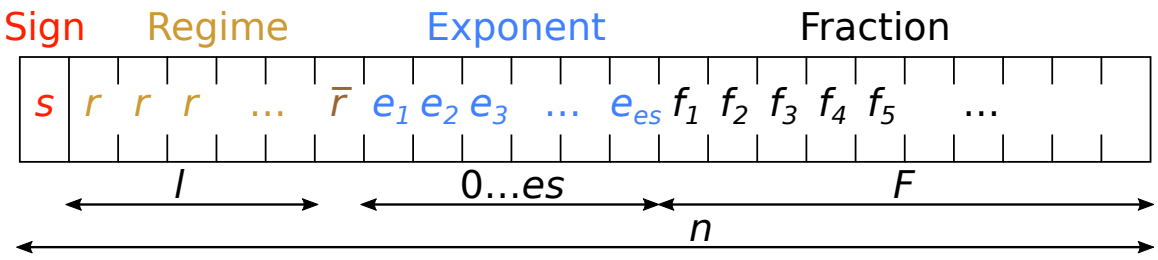


Figure 2: General Posit$\langle n, es\rangle$ binary encoding.

2.2 Posit Arithmetic

Posit numbers were introduced in 2017 as an alternative to the ubiquitous IEEE 754 floating-point standard (Gustafson and Yonemoto 2017). Posits provide reproducible results across platforms and few special cases. Furthermore, they do not support overflow or underflow, which reduces the complexity of exception handling.

A posit number configuration is defined as a tuple $\langle n, es\rangle$, where n is the total bitwidth and es is the maximum number of bits reserved for the exponent field. As Figure 2 shows, n-bit posit numbers are encoded with four fields: a single bit (s) which indicates the sign, several bits that encode the regime value (k), up to es bits for the unsigned exponent (e), and the remaining bits for the unsigned fraction (f). Although in literature (de Dinechin et al. 2019, Murillo, Del Barrio, and Botella 2020a, Murillo et al. 2021) different es sizes were used according to the total bitwidth ($es = 0$ for 8-bit posits, 1 for 16-bit posits, 2 for 32-bit posits, etc.), in the latest Posit Standard Draft (Posit Working Group 2022), the value of es is fixed to 2. This has the advantage of simplifying the hardware design and facilitates the conversion between different posit sizes. This configuration is the one considered in this work too.

The regime consists on a sequence of identical bits r of length l, which is finished with a negated bit $\bar{r}$. The regime encodes a scaling factor k given by Equation (2),

$$k = \begin{cases} -l & \text{if } r_0 = 0 \\ l-1 & \text{if } r_0 = 1 \end{cases}. \tag{2}$$

As this field has a variable length, some exponent or fraction bits might not fit in the n-bit string, so 0 would be assigned to them. However, the variable length of the regime allows posit arithmetic to have more fraction bits for values close to ± 1 (which increases the accuracy within that range), or to have less fraction bits for the sake of more exponent bits for values with large or small magnitudes (increasing this way the range of representable values). This is known as *tapered accuracy*, and contrasts with the constant accuracy that IEEE 754 floats present, due to the fixed length of the exponent and fraction fields, as can be seen in Figure 3 (here, the left part of the IEEE floating-point format corresponds to the gradual underflow that subnormal numbers produce). For applications handling data within the golden region of Figure 3, using posit arithmetic might result in more accurate results, or it could help to reduce the bitwidth, leading to lower resource consumption while keeping similar accuracy.

Posit arithmetic only distinguish two special cases: zero, that is represented with all bits equal to 0, and Not-a-Real (NaR) exception, represented by all the bits except the sign bit equal to 0. The rest of the bit patterns are used to represent a different real value.

The real value p of a generic posit is given by Equation (3). The main differences with the IEEE 754 floating-point format are the existence of the regime field, the use of an unbiased exponent, and the value of the fraction hidden bit. Usually, in floating-point arithmetic, the hidden bit is considered to be 1. However,

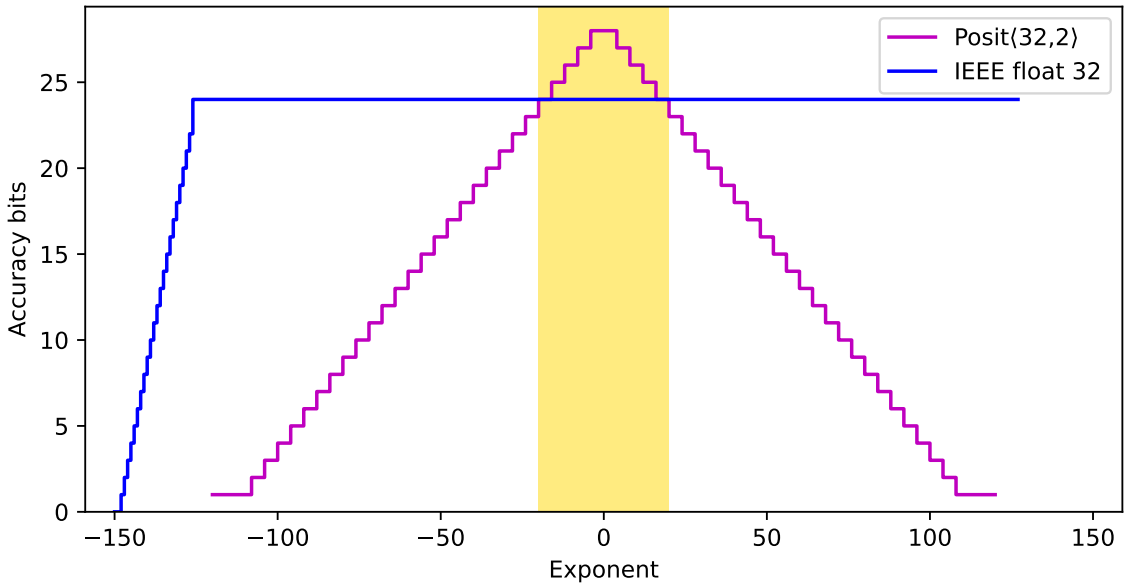


Figure 3: Accuracy binary digits for 32-bit formats.

in the latest representation of posits, it is considered to be 1 if the number is positive, or -2 if the number is negative. This simplifies the decoding/encoding stages of the posit representation (Mallasén et al. 2021).

$$p = ((1-3s)+f) \times 2^{(1-2s)\times(2^{es}k+e+s)}. \tag{3}$$

In posit arithmetic, NaR has a unique representation that maps to the most negative 2's complement signed integer. Consequently, if used in comparison operations, it results in less than all other posits and equal to itself. Moreover, the rest of the posit values follow the same ordering as their corresponding bit representations. These characteristics allow posit numbers to be compared as if they were 2's complement signed integers, eliminating additional hardware for posit comparison operations.

Although posit arithmetic was designed to have similar circuitry to floating-point format, the variable length of the fields and the signed hidden bit of the faction requires redesigning some of the logic when implementing posit operators. However, as already mentioned, this redesign in the arithmetic can provide further improvements in accuracy or performance of scientific applications.

3 RELATED WORK

The use of custom or low-precision arithmetic in hardware has experimented an increasing interest in the last decades with the rise of machine learning. Several studies show that 16-bit floats have enough accuracy and precision for most of the situations in this area (Gupta et al. 2015). Also, some works have simulated the use of posit arithmetic in deep learning (Ho et al. 2021, Murillo, Del Barrio, and Botella 2020b, Murillo et al. 2021), which seem to provide similar results with less bits than floating-point arithmetic. Alternative number formats are also attractive for accelerating general purpose scientific computing. Weather forecasting and climate modeling (Hatfield et al. 2018, Klöwer, Düben, and Palmer 2020) or solving systems of equations (Carson and Higham 2018, Saxena et al. 2021) are some examples.

Regarding emulation of custom arithmetics, floating-point and similar formats have been much more widely explored than the posit format due to the antiquity of the former. GNU MPFR is probably the most well-known library for arbitrary-precision binary floating-point computation. Authors in (Higham and Pranesh

2019) discuss how to simulate low-precision floating-point arithmetic using arithmetic of higher precision and different rounding modes.

On the other hand, posit is a much more recent format, and fewer works focused on the emulation of this arithmetic can be found so far. The most popular software libraries that support computation with posit arithmetic are SoftPosit (Leong 2020), which supports posits up to 32 bits with two exponent bits, and Universal (Omtzigt et al. 2020). The latter allows emulation of posits for any arbitrary configuration of bitwidth n and exponent size es, but also emulation of custom floating-point formats. Finally, another useful framework that is used in this work is QPyTorch (Zhang et al. 2019), which adds support for simulating low-precision arithmetic to the PyTorch machine learning framework. Originally, it was designed for floating-point arithmetic, but authors in (Ho et al. 2021) added support for posits as well.

4 EVALUATION

In this section, we evaluate the accuracy of various posit and floating point formats under multiple scientific applications. For each case, we compare the obtained result with the higher-precision IEEE floating-point format. As posit and low-precision floating-point formats do not have support in commercial hardware devices, we do not consider timing performance for this evaluation, since computation of such formats is done via software emulation.

4.1 Benchmark Description

In this work, we propose a wide variety of physical simulations and training of deep learning models. Each experiment is computed using different formats for floating-point—32-bit single precision, 16-bit half precision, and bfloat16 format (Kalamkar et al. 2019)—, and posit—from 32 down to 8 bits length—number formats.

The classical physical problems considered for the experiments are based on dynamical systems, so we compare not just the final result, but the error along all the steps. We consider the following problems:

- Simulation of Brownian motion in both 1 and 2 dimensions.
- Computation of the Fast Fourier Transform (FFT) from a corrupted signal having zero-mean random noise. The original signal is defined as

$$S(t) = 0.7 \times \sin(2\pi \times 50t) + \sin(2\pi \times 120t).$$

- Simulation of a 3D Lorenz system with parameter values $\beta = 8/3$, $\rho = 28$ and $\sigma = 10$, and initial conditions $x = 8$, $y = 1$, $z = 1$.

The solutions of such problems might correspond to data in one, two or three dimensions. Thus, the euclidean distance (in the corresponding dimension) with the solution provided by double precision (64-bit) floats is considered at each time-step (frequency in the case of FFT), and the Mean Squared Error (MSE) for the whole sequence of data is computed. The MSE metric becomes useful when comparing the precision of different arithmetic formats, as it effectively measures of how much two simulations deviate from each other by penalizing large errors and giving less importance to small differences.

Since neither posit nor 16-bit floating-point formats are supported in commercial general-purpose processors, such number formats are emulated via software with Universal Numbers Library (Omtzigt et al. 2020), a header-only C++ template library that supports arithmetic operations for any arbitrary posit and floating-point configuration.

Table 1: Error comparison for different number formats.

Benchmark	Float32	Float16	Bfloat16	Posit32	Posit28	Posit24	Posit20	Posit16	Posit14	Posit12	Posit10	Posit8
Brownian 1D	3.13e-12	1.95e-05	1.03e-03	6.11e-13	4.55e-12	1.74e-10	6.92e-08	8.58e-06	4.30e-05	6.02e-03	1.73e-02	1.45
Brownian 2D	8.77e-12	1.39e-04	2.39e-03	1.01e-12	1.11e-11	3.83e-10	2.11e-07	1.97e-05	1.13e-03	5.39e-03	2.63e-01	1.39
FFT	8.56e-15	7.70e-09	5.55e-07	5.31e-17	1.67e-15	1.52e-13	2.52e-11	5.12e-09	1.01e-07	1.38e-06	2.88e-05	3.49e-04
Lorenz	80.27	63.64	159.39	39.95	72.82	74.01	90.10	76.34	142.03	325.91	319.73	315.35

With regard to Deep Neural Networks (DNNs) (Sze et al. 2017), these are machine learning systems with two different modes or stages of execution: training (when these models learn by processing examples) and inference (when the models do predictions based on the learned parameters). Training is the most challenging and time-consuming process of creating deep learning models. Thus, in this work we focus on the effects the different numerical formats have in the training stage. We selected different models for image classification. As this is a supervised learning problem, the accuracy of each model is given by the amount of total images correctly classified. The architectures and datasets considered in this paper are the following:

- MNIST dataset of grayscale images of digits. The classical LeNet-5 architecture is trained for this dataset (LeCun et al. 1998).
- SVHN dataset of RGB images of digits. This is classified using ResNet-18 architecture (He et al. 2016a).
- CIFAR-10 dataset, containing RGB images of different objects classified in 10 classes. A 20-layer pre-activation ResNet (PreResNet) (He et al. 2016b) is trained on this dataset.

In the case of DNNs, 32-bit floating-point is the standard format to perform models training, while certain studies have shown that lower-precision formats are suitable to perform inference (Gupta et al. 2015). Training DNNs with lower-precision formats is also possible when performing certain modifications in the training loop. In this work, we perform loss scaling to train DNNs with less than 32 bits. In addition, regarding to posit arithmetic, previous works (Murillo, Del Barrio, and Botella 2020b, Murillo et al. 2021) have shown that 16 bits are enough to train such models without accuracy degradation, so in this case no models with more than 16 bits are considered.

Training large DNN models is often done with the help of accelerators such as GPUs, which do not support most of the low-precision formats considered in this work. Therefore, to emulate such formats and take advantage of hardware acceleration, we used QPyTorch (Zhang et al. 2019), which is a low-precision arithmetic simulation package in PyTorch. It supports quantizing weights and inputs with customized low-precision formats like posits (Ho et al. 2021), while performing accumulation (such as in matrix multiplication) in a larger precision format to avoid overflows, which is necessary when dealing with low-precision formats.

4.2 Accuracy Results

The MSE of dynamical systems in different simulated arithmetics is shown in Table 1. 32-bit posits provide more accurate results (up to two orders of magnitude) than the corresponding floats. This is due to the range the data of such applications belongs to. It falls within the golden zone in Figure 3, where the posits have higher accuracy than floats.

In the case of Brownian motion, most of the trajectories (except Posit10 and Posit8) overlap with the double-precision result, and no major visual differences are found. Even less error for each number format if obtained in the case of FFT. As can be seen in Figure 4, the two frequencies (50 and 120 Hz) and amplitudes

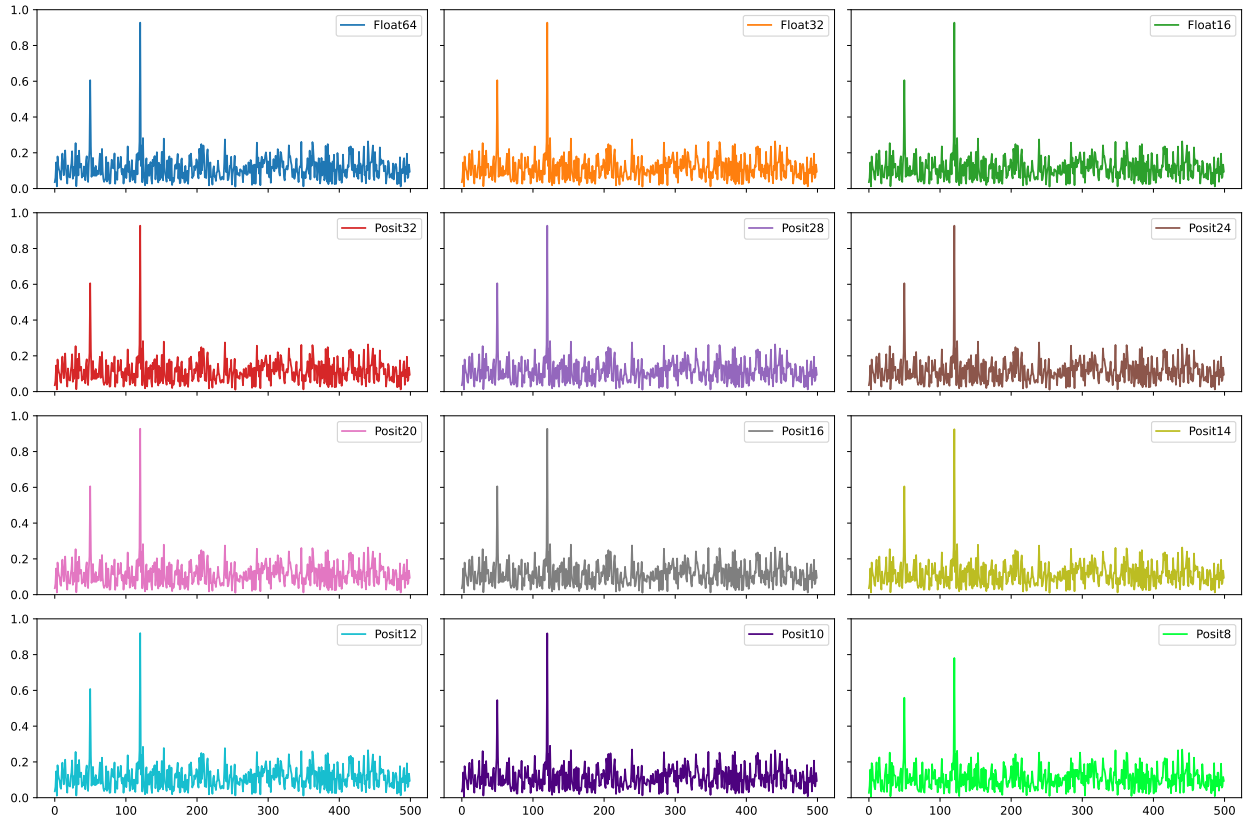


Figure 4: Single-sided amplitude spectrum for a signal applying the FFT with different arithmetic formats and number of bits.

(0.7 and 1) of the original signal are distinguishable in all cases, even in the case of Posit8 (but with certain attenuation in the amplitude).

The results obtained for the Lorenz system deserve special attention. Such a model is very sensitive on initial conditions and computation errors; that is, tiny differences in the starting condition for the system rapidly become magnified. Thus, even error in Table 1 can be indicative, the trajectories shown in Figure 5 reveal that even Float16 provides less MSE than Float32 or Posit20, 16 bits are not enough to solve this classical problem. Since the trajectory obtained with Float16/Posit16 falls in the midpoint of the real trajectory, their MSE is smaller than the obtained with other formats, although the overall results with such 16-bit formats are less accurate. Instead, 20-bit posits can still provide valid solutions, and the higher accuracy of 32-bit posits results in an attractor much more similar to the double-precision attractor than the obtained with 32-bit floats.

Regarding DNNs, the maximum accuracy reached by each model after 100 training epochs is shown in Table 2. All the models were trained using Stochastic Gradient Descent (SGD) algorithm and the low-precision format models perform loss scaling to prevent missing too much information. In all the cases, the random seed is fixed to the same value, so all the training processes proceeds from the same initial parameters. As can be seen, down to 12-bit posits are suitable to train deep models such as ResNets. Using less than 12 bits results in higher accuracy drop (in the case of Posit10), or even no convergence of the models (for Posit8). Since memory transfers are one of the main bottlenecks in DNN scenario, a lower precision format like 12-bit posit might increase the throughput of these algorithms.

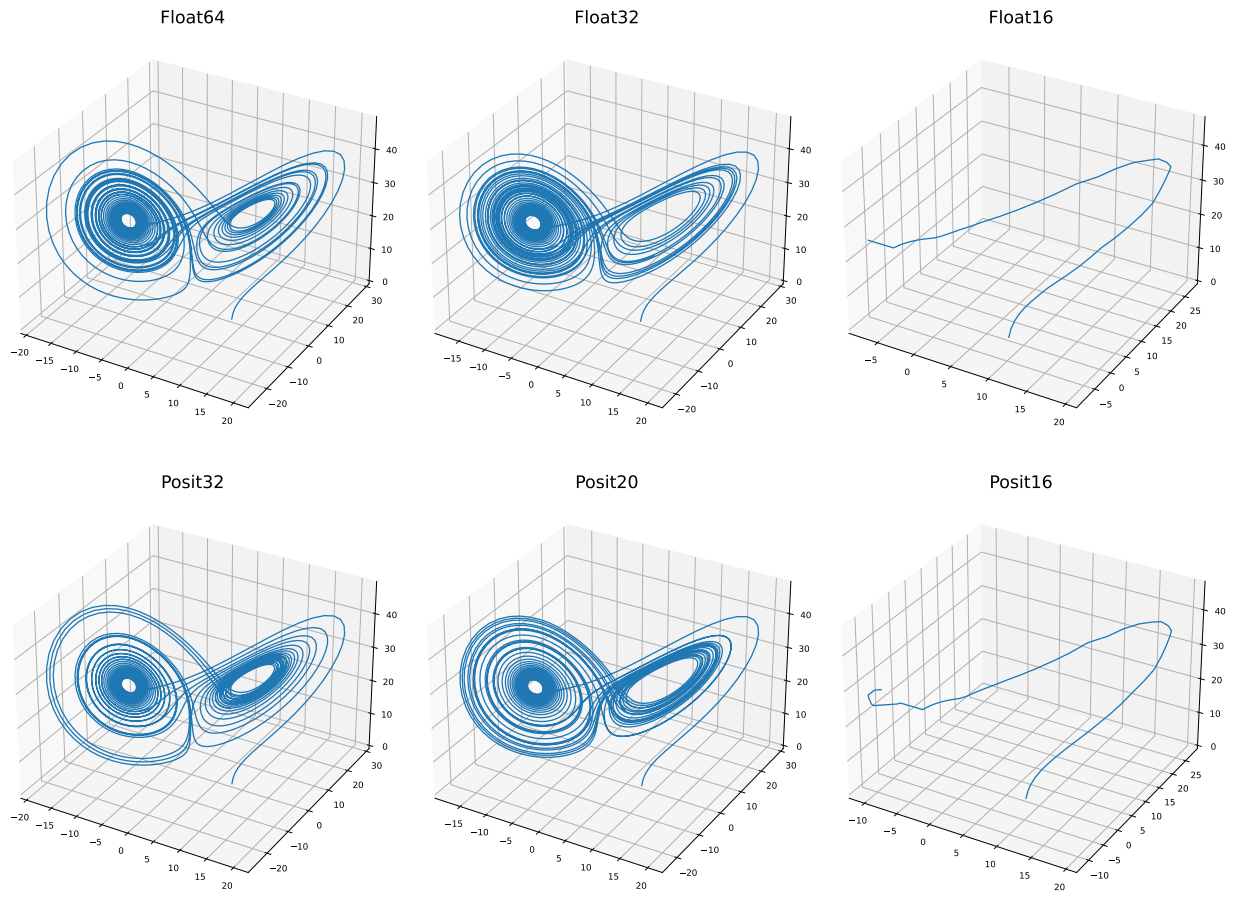


Figure 5: Solutions of the Lorenz system for different arithmetic formats and number of bits.

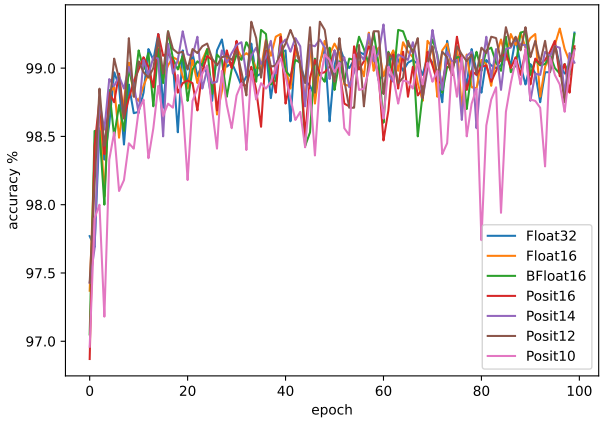
To better understand the effects of the different number formats in this deep learning context, the training processes on MNIST, SVHN and CIFAR-10 datasets are depicted in Figure 6, Figure 7 and Figure 8, respectively. The training curves for formats down to 12 bits are indistinguishable. On the other hand, models using Posit10 show much more variability in the training processes. Posit8 training plots are omitted from Figure 6 for their low values, but the results are similar as those for PreResNet-20. The case of ResNet-18 using 8-bit posits deserves more attention, since just a 2.5% accuracy drop is obtained. At first glance, the only difference with the other cases is the architecture of the neural network; the ResNet architecture might be tolerant to lower precision formats. However, further analysis would be needed to elucidate the reason why such precision provides sufficient accuracy for this specific case, but this is out of the scope of this work.

5 CONCLUSIONS AND FUTURE WORK

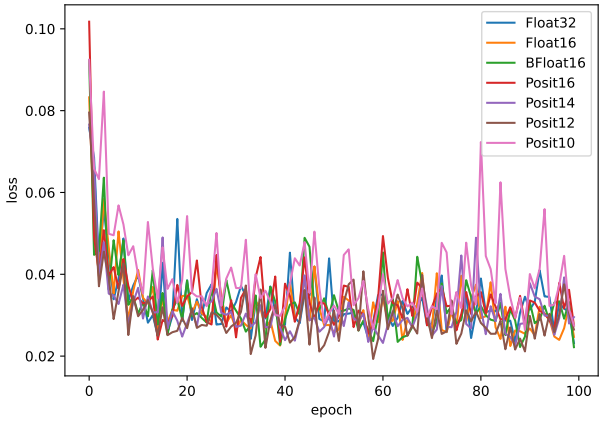
We have simulated floating-point and posit number systems under different precisions and analyzed the accuracy provided by each format in a set of scientific applications, including computation of dynamical systems and training of deep learning models. Software simulation of different arithmetic formats provides information on the acceptable precision limits for each application, which is essential in the development of domain-specific hardware accelerators.

Table 2: DNN training results (accuracy %) for different number formats.

Benchmark	Float32	Float16	Bfloat16	Posit16	Posit14	Posit12	Posit10	Posit8
MNIST	99.26	99.29	99.28	99.25	99.32	99.34	99.16	29.13
SVHN	95.70	96.09	95.89	95.99	95.86	95.98	95.63	93.42
CIFAR-10	86.50	86.40	85.99	86.29	85.71	86.79	80.52	38.67

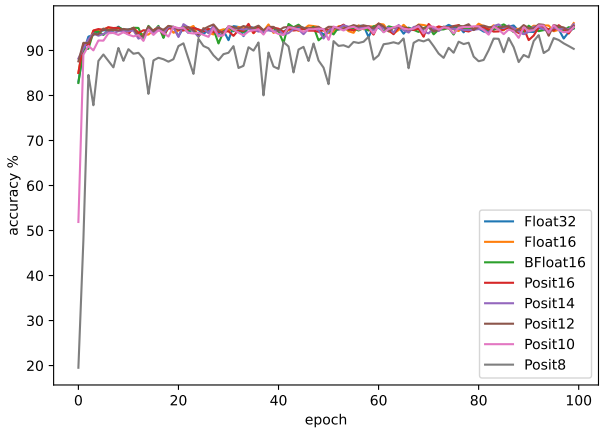


(a) Accuracy per epoch.

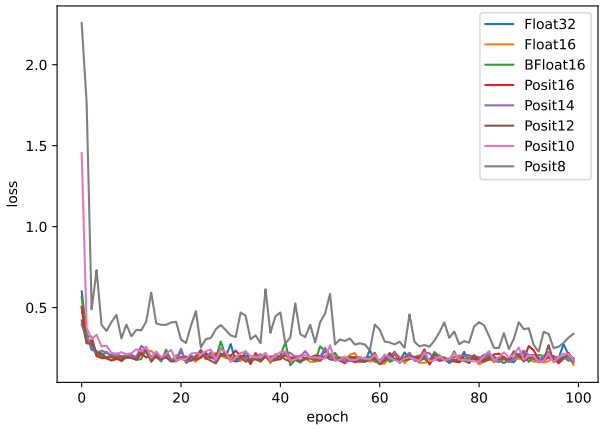


(b) Loss per epoch.

Figure 6: LeNet training on MNIST dataset.



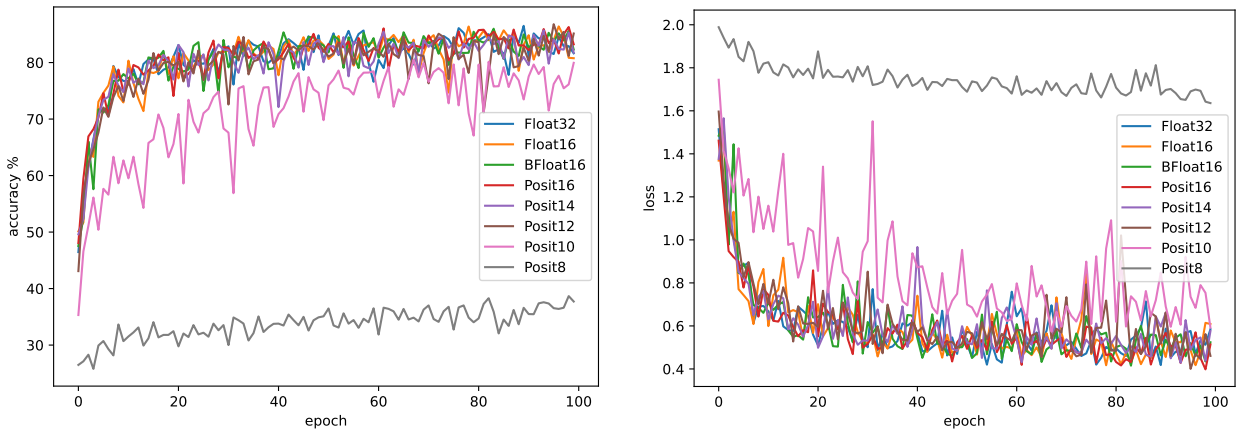
(a) Accuracy per epoch.



(b) Loss per epoch.

Figure 7: ResNet-18 training on SVHN dataset.

12-bit posits provide negligible accuracy drop in DNN training when compared with single or even half precision formats, which would have a great impact on the performance and energy consumption when deploying such systems on dedicated platforms. On the other hand, computation of physical systems might require more precision depending on the application, but simulation provides a useful tool for understanding the effects of lowering the precision in numerical computations or of formats not supported in hardware such



(a) Accuracy per epoch.

(b) Loss per epoch.

Figure 8: PreResNet-20 training on CIFAR-10 dataset.

as posit. In the evaluated applications, under the same number of bits, posit arithmetic provides up to two orders of magnitude less error than floating-point format.

Future work will also evaluate not just the accuracy of each numeric format, but also the hardware cost associated with them, either by single operators or by entire accelerators. Also, the effects of using multiple precisions or formats in each of the operations within a single problem have not been explored that much, especially in the case of posit arithmetic. Therefore, this seems a promising line of research for future works.

ACKNOWLEDGMENTS

This work was supported by a 2020 Leonardo Grant for Researchers and Cultural Creators, from BBVA Foundation, whose id is PR2003_20/01, by the EU(FEDER) and the Spanish MINECO under grant RTI2018-093684-B-I00, and by the CM under grant S2018/TCS-4423.

REFERENCES

Carson, E., and N. J. Higham. 2018. "Accelerating the solution of linear systems by iterative refinement in three precisions". *SIAM Journal on Scientific Computing* vol. 40 (2), pp. A817–A847.

de Dinechin, F., L. Forget, J.-M. Muller, and Y. Uguen. 2019. "Posits: the good, the bad and the ugly". In *2019 Conference for Next Generation Arithmetic (CoNGA)*, pp. 1–10, ACM.

Goldberg, D. 1991. "What every computer scientist should know about floating-point arithmetic". *ACM Computing Surveys (CSUR)* vol. 23 (1), pp. 5–48.

Guntoro, A., C. De La Parra, F. Merchant, F. De Dinechin, J. L. Gustafson, M. Langhammer, R. Leupers, and S. Nambiar. 2020. "Next Generation Arithmetic for Edge Computing". In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1357–1365, IEEE.

Gupta, S., A. Agrawal, K. Gopalakrishnan, and P. Narayanan. 2015. "Deep Learning with Limited Numerical Precision". In *Proceedings of the 32nd International Conference on Machine Learning*, edited by F. Bach and D. Blei, Volume 37 of *Proceedings of Machine Learning Research*, pp. 1737–1746. Lille, France, PMLR.

Gustafson, J. L., and I. T. Yonemoto. 2017. "Beating Floating Point at Its Own Game: Posit Arithmetic". *Supercomputing Frontiers and Innovations* vol. 4 (2), pp. 71–86.

Hatfield, S., P. Düben, M. Chantry, K. Kondo, T. Miyoshi, and T. Palmer. 2018. "Choosing the optimal numerical precision for data assimilation in the presence of model error". *Journal of Advances in Modeling Earth Systems* vol. 10 (9), pp. 2177–2191.

He, K., X. Zhang, S. Ren, and J. Sun. 2016a. "Deep residual learning for image recognition". In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CPVR)*, Volume 2016-Decem, pp. 770–778.

He, K., X. Zhang, S. Ren, and J. Sun. 2016b. "Identity Mappings in Deep Residual Networks". In *European Conference on Computer Vision (ECCV 2016)*, edited by Springer, pp. 630–645, Springer International Publishing.

Hennessy, J. L., and D. A. Patterson. 2019. "A new golden age for computer architecture". *Communications of the ACM* vol. 62 (2), pp. 48–60.

Higham, N. J., and S. Pranesh. 2019. "Simulating Low Precision Floating-Point Arithmetic". *SIAM Journal on Scientific Computing* vol. 41 (5), pp. C585–C602.

Ho, N.-M., D.-T. Nguyen, H. D. Silva, J. L. Gustafson, W.-f. Wong, and I. J. Chang. 2021. "Posit Arithmetic for the Training and Deployment of Generative Adversarial Networks". In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1350–1355, IEEE.

IEEE Computer Society 2019. "IEEE Standard for Floating-Point Arithmetic". *IEEE Std 754-2019 (Revision of IEEE 754-2008)* vol. 2019, pp. 1–84.

Kalamkar, D., D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey. 2019. "A Study of BFLOAT16 for Deep Learning Training". *arXiv e-prints*, pp. 1–10.

Klöwer, M., P. D. Düben, and T. N. Palmer. 2020. "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* vol. 12 (10), pp. e2020MS002246.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner. 1998. "Gradient-based learning applied to document recognition". *Proceedings of the IEEE* vol. 86 (11), pp. 2278–2323.

Leong, Siew Hoon 2020. "SoftPosit". https://gitlab.com/cerlane/SoftPosit. Accessed Jun. 20, 2022.

Mallasén, D., R. Murillo, A. A. Del Barrio, G. Botella, L. Piñuel, and M. Prieto. 2021. "PERCIVAL: Open-Source Posit RISC-V Core with Quire Capability". *arXiv e-prints*, pp. 1–11.

Murillo, R., A. A. Del Barrio, and G. Botella. 2020a. "Customized Posit Adders and Multipliers using the FloPoCo Core Generator". In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE.

Murillo, R., A. A. Del Barrio, and G. Botella. 2020b. "Deep PeNSieve: A deep learning framework based on the posit number system". *Digital Signal Processing: A Review Journal* vol. 102, pp. 102762.

Murillo, R., A. A. Del Barrio Garcia, G. Botella, M. S. Kim, H. Kim, and N. Bagherzadeh. 2021. "PLAM: a Posit Logarithm-Approximate Multiplier". *IEEE Transactions on Emerging Topics in Computing*, pp. 1–7.

Murillo, R., D. Mallasén, A. A. Del Barrio, and G. Botella. 2021. "Energy-Efficient MAC Units for Fused Posit Arithmetic". In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pp. 138–145, IEEE.

Omtzigt, T., P. Gottschling, M. Seligman, and B. Zorn. 2020. "Universal numbers library: Design and implementation of a high-performance reproducible number systems library". *arXiv e-prints*, pp. 1–7.

Posit Working Group 2022. "Standard for Posit™ Arithmetic". https://github.com/posit-standard/Posit-Standard-Community-Feedback. Accessed Jun. 20, 2022.

Saxena, V., A. Reddy, J. Neudorfer, J. Gustafson, S. Nambiar, R. Leupers, and F. Merchant. 2021. "Brightening the Optical Flow through Posit Arithmetic". In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pp. 463–468, IEEE.

Sze, V., Y.-H. Chen, T.-J. Yang, and J. S. Emer. 2017. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". *Proceedings of the IEEE* vol. 105 (12), pp. 2295–2329.

Zhang, T., Z. Lin, G. Yang, and C. De Sa. 2019. "QPyTorch: A Low-Precision Arithmetic Simulation Framework". In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pp. 10–13.

AUTHOR BIOGRAPHIES

RAUL MURILLO is currently a Ph.D. candidate in Computer Science at the Complutense University of Madrid (UCM), Spain. He studied Mathematics and Computer Science at UCM, where he also received a M.Sc. degree in Computer Science in 2021. His main research interests include new Computer Arithmetic, Approximate Computing, and Deep Neural Networks (DNNs). His email address is ramuri01@ucm.es.

ALBERTO A. DEL BARRIO received the Ph.D. degree in Computer Science from the Complutense University of Madrid (UCM), Spain, in 2011. He has performed stays at Northwestern University, University of California at Irvine and University of California at Los Angeles. Since 2021, he is an Associate Professor (tenure-track, civil-servant) of Computer Science with the Department of Computer Architecture and Automation, UCM. His main research interests include Design Automation, Arithmetic and their application to the field of Artificial Intelligence. He is leading the PARNASO project, funded by the Leonardo Grants program by Fundación BBVA. The main objective is to natively integrate the posit format in a hardware/software platform. Since 2019 he is an IEEE Senior Member and since December 2020 he is an ACM Senior Member, too. His email address is abarriog@ucm.es.

GUILLERMO BOTELLA received the M.A.Sc. degree in Physics (Fundamental) in 1998, the M.A.Sc. degree in Electronic Engineering in 2001 and the Ph.D. degree (Computer Engineering) in 2007, all from the University of Granada, Spain. He was a research fellow funded by EU working at University of Granada, Spain and the Vision Research Laboratory at University College London, UK. After that, he joined as Assistant Professor at the Department of Computer Architecture and Automation of Complutense University of Madrid, Spain where he is currently Associate Professor. He has performed research stays acting also as visiting professor from 2008 to 2012 at the Department of Electrical and Computer Engineering, Florida State University, Tallahassee, USA. His current research interests include Image and Video Processing for VLSI, FPGAs, GPGPUs, Embedded Systems, and novel computing paradigms such as analog and quantum computing. Since 2019 he has become an IEEE Senior Member. His email address is gbotella@ucm.es.