View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Archivio della Ricerca - Università di Pisa

Exploiting Posit Arithmetic for Deep Neural Networks in Autonomous Driving Applications

Marco Cococcioni[1], Emanuele Ruffaldi[2] and Sergio Saponara[1]

***Abstract*— This paper discusses the introduction of an integrated Posit Processing Unit (PPU) as an alternative to Floating-point Processing Unit (FPU) for Deep Neural Networks (DNNs) in automotive applications. Autonomous Driving tasks are increasingly depending on DNNs. For example, the detection of obstacles by means of object classification needs to be performed in real-time without involving remote computing. To speed up the inference phase of DNNs the CPUs on-board the vehicle should be equipped with co-processors, such as GPUs, which embed specific optimization for DNN tasks. In this work, we review an alternative arithmetic that could be used within the co-processor. We argue that a new representation for floating point numbers called Posit is particularly advantageous, allowing for a better trade-off between computation accuracy and implementation complexity. We conclude that implementing a PPU within the co-processor is a promising way to speed up the DNN inference phase.**

I. Introduction

Assisted and Autonomous Driving (AD) require the understanding of the physical environment outside the vehicle as shown in Figure 1. New on-board automotive computing architectures [1]–[6] will exploit powerful embedded High Performance Computing (eHPC) platforms, such as NVDIA Pegasus or Intel GO, implementing in real-time the following vehicle perception and autonomous decision tasks:

- Observation: building a model of the surrounding environment, where inputs are the direct observations produced by sensors (visual and infrared cameras, radar, sonar, lidar [7]) and output is a geometrical and topological representation of the environment.
- Perception: localization of the vehicle, i.e. estimating its path, position and orientation within a map, by fusing global and relative data (e.g. Global Navigation Satellite signals fused with local accelerometer and gyro inertial sensors); detection of all static and dynamic obstacles (e.g. landmarks, road and traffic signs, vehicles, pedestrian, bikers) and their classification depending on how well they match up with a library of pre-determined shape and motion descriptors.
- Planning and decision: move the car, which requires Artificial Intelligence (AI) for adaptive route planning and trajectory control used to direct the vehicle to its destination, avoiding obstacles, following traffic rules, and predicting the behavior of neighbor vehicles, bikers and pedestrian.

Online map data is required to provide long range planning information such as lane end, speed limits, construction sites and other changing road conditions. All these operations have to be repeated in a time scale of about 5 ms (200 Hz) with stringent low-latency requirements [8, 9]. To support the functions of mapping, localization and object identification, a perception process must be implemented (e.g. [10]). Perception results from fusion of all surround sensing and online map data into single surround model. For data-fusion a grid-based approach may be used to determine the occupancy probability (Bayesian approach) of a cell, or the belief function (Dempster-Shafer approach), by evaluating the current sensor reading and the history from past cycle [11]. Grid occupancy is calculated from sensor data, with explicit modeling of uncertainties. Grid cells can bear additional information such as moving object speed, which can be used to predict likely behavior.

The current and future trend is solving such difficult tasks by using Deep Neural Networks (DNNs) [12]–[17] trained on millions of frames on a supercomputer based on GPUs with units specialized for tensor operations. Once the DNN has been trained and validated, it can be used in real-time on-board a vehicle (a car or truck or a public transport bus/taxi), to understand the environment and help and inform higher-level of the decision and control process. The speed of the DNN inference is crucial in this kind of latency-critical applications. But even more crucial is the fact that autonomous driving is a safety critical application, thus it requires a reliable software implementation of DNNs on a reliable hardware. In summary, a fast and reliable DNN module is required, either implemented in software or in hardware. Regarding the inference phase (the only phase that need to be performed on-board the vehicle), 8 bit or less are enough on non-safety critical applications [17]–[19], by using vector quantization or integer/fixed-point arithmetic. On the contrary, safety critical applications (see critical automotive standards, such as the ISO 26262 functional safety specification) still require 16 or 32 bits and thus floating point representations must be preferred over fixed point/integer representations.

To accelerate DNN computing in automotive applications, this paper discusses the introduction of the Posit Processing Unit (PPU) as alternative to the Floating-point Processing Unit (FPU). After discussing in Section II alternative representations for real numbers, the new Posits format is

[1]University of Pisa, Department of Information Engineering, Via Caruso 16, Pisa - 56123 Italy {marco.cococcioni, sergio.saponara}@unipi.it
[2]MMI s.p.a., Via del Paduletto 10A, 56011, Calci (PI), Italy emanuele.ruffaldi@mmimicro.com

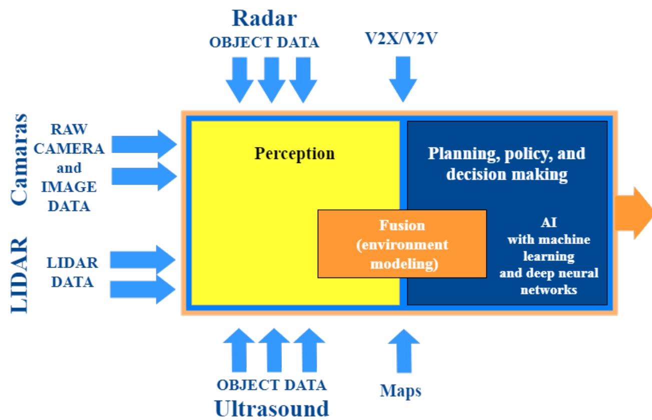

Fig. 1. AD architecture with perception (on the left) based on sensor data fusion and planning (on the right). Both functions can take advantage of AI approaches in particular deep learning techniques.

proposed in Section III. In Sections IV we argue that 16-bit Posits can replace conventional FPUs, since they are more accurate if the same number of bits is used. Moreover, Posits enjoy other interesting properties, like the possibility to compare two Posits by treating them as two integers on ALU. Hardware/ Software implementations issues of a PPU are also discussed, together with results we obtained by emulating a PPU exploiting a General Purpose microprocessor (GPP) or a microcontroller. Conclusions and future hints of the use of PPU for the DNN training phase are presented in Section V.

II. Alternative Real Number Representations

Representing real numbers in electronic computers requires the selection of a method to map real numbers into a sequence of bits. In addition, a circuitry is typically required to perform in hardware (and thus, in a fast way) both the basic four arithmetic operations ($+$, $-$, $\times$, $/$) and the comparison operators of two numbers ($<$, $==$, $>$, $\leq$, $\geq$). Hardware realization of specific elementary functions are also desired.

A. Type-I Unums

Type-I uniform numbers have been introduced in Gustafson's book [20]. The Type-I Unums data format is a superset of IEEE 754 Standard floating-point format; it uses a "ubit" at the end of the fraction to indicate whether a real number is an exact float or lies in the open interval between adjacent floats. While the sign, exponent, and fraction bit fields take their definition from IEEE 754, the exponent and fraction field lengths vary automatically, from a single bit up to some user specified maximum (see Figure 2). Type-I Unums provide a compact way to express interval arithmetic, but their variable length demands extra management. They can express IEEE float behavior, via an explicit rounding function.

- **Advantages:** superset of floats.
- **Disadvantages:** variable length; require complex interval arithmetic.

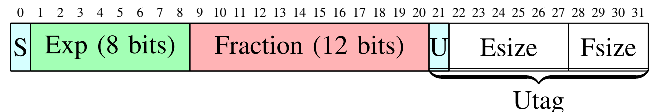

Fig. 2. An example of Type-I Unum. The first three fields are the same of floating point numbers, the remaining ubit, esize, fsize are additional fields.

- **Software implementations**: Type-I Unums have been implemented in Matlab and applied to a Model Predictive Control Problems [21]. In that work, the authors showed a save of 80% of storage space with respect the use of Floats, without loosing in accuracy.
- **Hardware implementations:** Type-I Unums have been recently implemented in hardware, on FPGA [16, 22]. The authors conclude that implementing in hardware Type-I Unums requires a bigger area than that of an FPU. This is due to both the variable length operands and the interval arithmetic, which is more complex than arithmetic between floats. We believe that Type-I Unums are not particularly appealing for DNN in autonomous driving applications, due to the disadvantages mentioned above.

B. Type-II Unums

Type-II Unums have been introduced again by Prof. Gustafson in [23]. The basic requirement in designing Type-II Unums was a fixed length data type. Fixed length is very important when working with problems showing data parallelism and, more in general, to efficiently work with arrays (vectors, matrices, tensors, etc.). Another important designing criteria was the desire to not overlook any real number. All real numbers from $-\inf$ to $+\inf$ should be represented, although with different precisions. Here the key idea was to represent both exact real numbers and the open interval between two consecutive exact real numbers represented. Then Gustafson exploited the idea of "projective reals", e.g., the idea of mapping the real line to a circle, with the intent of constructing a bijection between the line and the circle. By using this approach, Gustafson was able to express the opposite of a number in the circle via "horizontal flipping", while, more interestingly, the reciprocal of a number is obtained exactly by "vertical flipping" of the representation.

This is a very important property of Type-II Unums, since reciprocating a number is straightforward and thus computing the division requires a similar amount of time of computing the product (by computing a/b as $a \times b^{-1}$). This means that the "bad boy" of computing (dividing two real numbers) is removed, and all the four arithmetic operations require similar amount of time. Figure 3 shows Type-II Unums when using only 4 bits, represented on the mentioned circle. Finally, the idea of Type-II Unums was the computation of accurate and reproducible results, even on parallel computers. Based on this Gustafson introduced the idea of Set of Real Numbers (SORNs), an intriguing way to represent any set of Type-II Unums. Unfortunately operations on SORNs require a look-up table, thus limiting

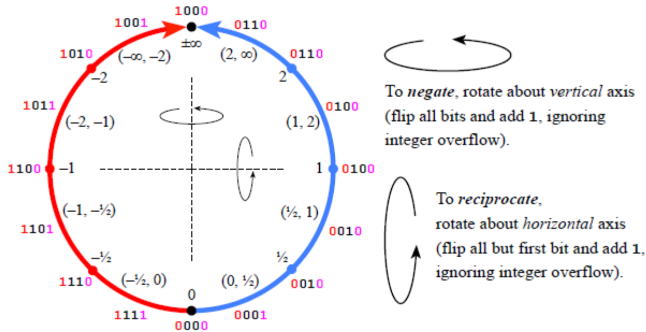

Fig. 3. Type-II Unums on 4 bits. This kind of numbers can be represented on a circle. Even more interesting, the representation of the reciprocal of a number can be easily obtained by working on the representation itself, by performing a rotation on the horizontal axis.

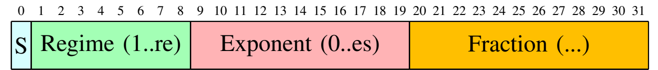

Fig. 4. An example of Posit data type.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

S	R	E	F
0	0 0 0 1	1 0 1	1 1 0 1 1 1 0 1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

S	R	E	F
0	1 1 0	0 1 1	0 0 0 1 0 1 0 0 0

Fig. 5. Two examples of 16-bit Posit with 3 bits for exponent (es=3). In the upper the numerical value is: $+256^{-3} \cdot 2^{5} \cdot (1 + 221/256)$ (221/256 is the value of the fraction, 1 + 221/256 is the value of the mantissa). The final value is therefore $1.907348 \times 10^{-6} \cdot (1 + 221/256) = 3.55393 \times 10^{-6}$. In the lower the numerical value is: $+256^{+1} \cdot 2^{3} \cdot (1 + 40/512)$ (40/512 is the value of the fraction, 1 + 40/512 is the value of the mantissa). The final value is therefore $2048 \cdot (1 + 40/512) = 2208$.

the total length of Type-II Unums on current hardware to 20 bits or less.

- **Advantages:** they have fixed size, in contrast to Type-I Unums. In addition, computing the opposite and the reciprocal takes the same time
- **Disadvantages:** The SORN approach needs look-up tables requiring large amounts of ROM/RAM.
- **Hardware implementations:** Type-II Unums have not been implemented in hardware yet. The SORN approach they are based allows using a look-up table, for numbers up to 20 bits.

C. Other formats

Jorgensen has recently patented a format which retains the correct bounds after each computation and the corresponding hardware [24]. This proposal is another evidence of the need of novel formats for real numbers and novel hardware implementations, and, although interesting, is too similar to Type-I Unums. On the other hand Type-I Unums are free from patents rights. In conclusion of this section, none of these newly proposed floating point representations are satisfactory for DNNs. On the contrary, the Posit data format, explained in the next section has promising properties.

III. The Posit representation

Posits are a by-product of the Type-II Unums numbers described in previous section and they have been introduced very recently in [25], again by Gustafson. The Posit representation is depicted in Figure 4. A Posit contains a maximum of four fields: the sign field (1 bit), the regime field (variable-length field), the exponent bit (fixed-length field which can also be of zero bits), and the fraction field (variable-length field, which can even be absent for some configurations). The length of the exponent field is decided a-priori, together with the total length for Posits. These two lengths characterize different types of Posit representations. The length of the regime field is determined using a run-length method: the number of consecutive 0 after the sign bit and before the first 1 bit is the regime length (a regime field can be also made by a sequence of 1, until the first 0 is encountered: in that case the number of consecutive 1 is the regime length, but this time its value is negative). Once the length of the regime is known, the length of the mantissa can be determined, as the number of remaining bits (after skipping the exponent bits). The formula that allows to retrieve the real number is available [25] and two examples of its application are shown in Figure 5. Please observe how the two Posits representations shown in the figure have a different number of bits reserved for the fraction field (8 and 9, respectively), having different lengths for the regime fields (4 vs 3).

Similarly to Type-II Unums, Posits can be put on a circle sharing the concept of projection of reals over a circle, but different design decisions allow to implement Posit operation without imposing the use of a look-up table. In Figure 6 the (semi)-circles of 3-bit, 4-bit and 5-bit Posits are shown.

Posits enjoy many really interesting properties, such as:

- unique representation for zero
- no representations wasted for Not-A-Number (when using Posits, an exception is raised instead of reserving one or more representations for NaNs). Please notice that the floating-point standard wastes a lot of representations for NaNs, which also make the hardware for comparing floats very complex.

Even more interestingly, ***Posits are sorted like signed integers*** (when the latters are represented using the two's complement). Thus comparing two Posits can be done in ALU (by *type re-casting* to signed integers): negative Posits are expressed using complement two as integers, and the other three fields allow direct ordering.

We think the brightest idea of the Posit representation is to ***reserve more bits to the mantissa for small real numbers*** (closer to zero) and ***less for large real numbers***, within a fixed length format (the total length is fixed, although the length of the regime and that of the mantissa vary).

Finally observe that, since the Posit format standardization process is still on its infancy, we are on time to provide useful feedbacks to the standardization committee, providing them suggestions based on the experience in their use with DNN.

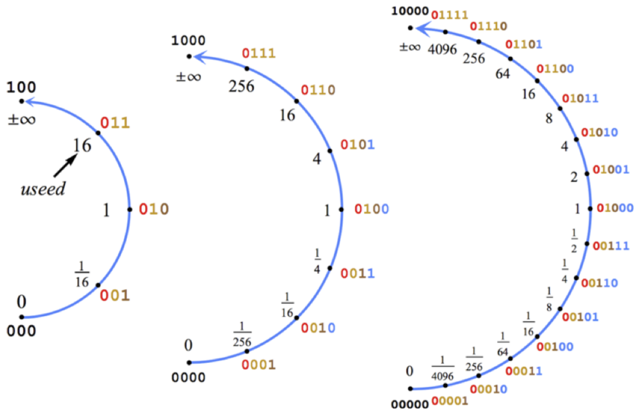

Fig. 6. Posits semi-circles for different numbers of bits (3, 4 and 5).

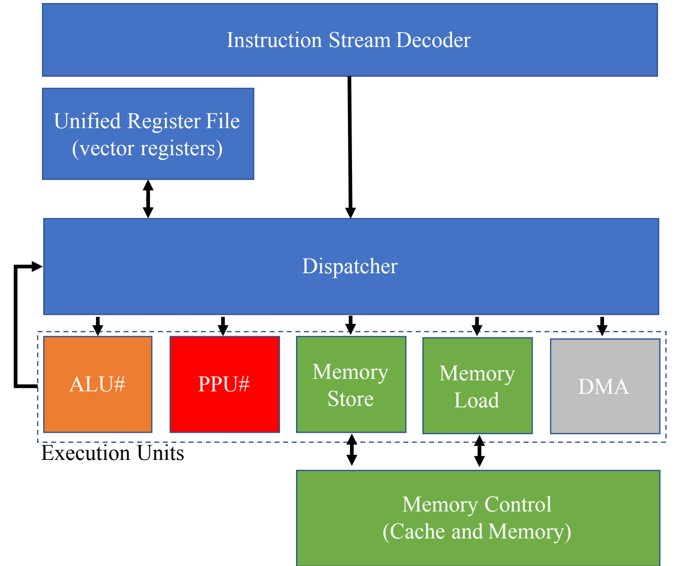

Fig. 7. Proposed superscalar architecture for a processor (or a co-processor) that integrates the PPU. Thanks to register sharing and the properties of Posit some operations can be performed using the ALU. Multiple ALU and PPU and execution units are possible thanks to the parallel structure of the superscalar architecture.

A. Dimensioning Posits

As mentioned the Posit representation is parametrized over the total number of bits to use and the number of them to reserve to the exponent. Depending on the task at hand, these two numbers can chosen accordingly. For instance, as regards the DNN inference phase we know that 16-bit floats are sufficient enough. Thus we know that 16-bit Posits are sufficient too. However, how many bits to reserve to the exponent part of the Posit can still be optimized on the task at hand. In addition we observe that the Posit representation is compatible with the *flexpoint* idea introduced in [15]. Indeed, it is possible to create Posits tensors that share an extra exponent field. Thus, when dimensioning Posits this extra exponent field must be dimensioned together with the dimensioning of Posits. Finally we point out how that *stochastic rounding* [26] could be integrated in Posit arithmetic too (even if this adds an extra logic and thus additional transistors).

IV. Hardware Implementation Issues

The implementation of a PPU will take advantage of the operations over Posits that can be traced back to three main levels of Posit representation: i) binary form, ii) decoded form, and iii) expanded form. Indeed we associate to any operation a **Posit representation level**. The first one is the binary representation of Posits discussed so far that can be used for some operations, with the effect of fastest speed and minimal complexity. The second form decodes the Posit in the three components of regime, exponent and fraction by parsing the variable length encoding of the regime field. Inversion, doubling and halving can be efficiently computed by working on this intermediate representation, in addition to the fundamental arithmetic operations when fraction is zero. The third level requires a more sophisticated logic that, anyway, in comparison to IEEE floating points, can take advantage of the separation between regime and exponent parts.

A. The advantage of using shared registers between ALU and PPU

The property of Posit representation that allows to perform comparisons using the integer comparison can be exploited for a subset of DNN operations. We are referring on particular to the *argmax* operation at the end of the inference, to the ReLu activation function, and to Max pooling in convolutional neural networks (see next subsection). The manipulation of a Posit number using integer operations or bitwise operations justifies the proposal of adopting PPU as an integrated part of a CPU, namely as an Execution Unit. Some Posit operation could require custom instructions while others (such as the comparison) could be obtained via integer instructions. Figure 7 shows the schematic of registers within the co-processor, to reuse the ALU for comparing Posits.

In the instructions' opcode we must be able to specify whether the 32-bit registry contains an operand of 8, 16, or 32 bits, being agnostic on its content (an address, an unsigned, an integer or a posit). This approach, depicted in Figure 7, is common to modern architecture (e.g. Intel Core CPUs) that generalize this properties to larger register banks, up to 512 bits.

B. Specific DNN functions to implement in hardware

In existing custom hardware for Tensor operations (e.g. Nvidia Tensor units and Google TPU) there are specific operations that are structured in hardware due to their repetitive nature. In the following we are considering some fundamental DNN operations that can take advantage of Posit representation and that can be directly executed. DNN functions that are interesting to implement in hardware are the following:

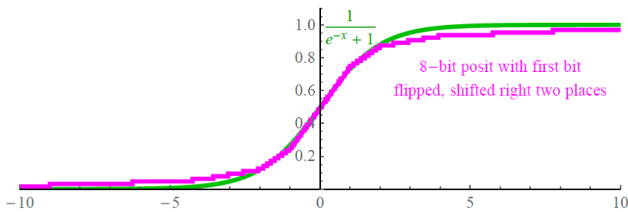

Fig. 8. The Sigmoid function and its approximation using Posits.

TABLE I
DIFFERENT SOLUTIONS TO COMPUTE WITH POSITS

	Posits up to 14 bits	**Larger Posits**
Solution A	Hardware PPU	Hardware PPU
Solution B	Look-up table on ROM/RAM plus some ALU	Emulated PPU using ALU only (not the FPU), by including the developed C++ Posit class

- Sigmoid function: $\frac{1}{e^{-x}+1}$. This function is widely used as activation function in DNNs. However, its evaluation is time consuming, since it requires and exponentiation and a division. As recently observed [25], this function can be easily approximated in hardware by doing simple bit-cloning and masking on the Posit representation. Figure 8 shows how closely the approximated version compares with the exact counterpart.
- ReLu: max(0, x) (*Rectified Linear Unit*). This function is an appealing alternative to the exact Sigmoid function, reducing the problem of vanishing gradient while being faster. As such, it has been widely used in DNNs. However, when using Posits, the approximated sigmoid function discussed above can be used on its place, since the two require comparable time (being both executed in hardware). Thus computer scientist are no longer forced to use a non-differentiable and less expressive function, like the ReLu, and can continue to use the more expressive Sigmoid function (although approximated).
- MaxPool: max pooling is a mathematical function widely used in convolutional neural networks. MaxPool can be thought as a *max* filter applied to an image block-wise and thus it requires many comparisons between real numbers. Having the possibility to compare two Posits using the ALU we saving a significant of transistors (and thus we save energy consumption at run-time) by not having to implement the comparison operators within the PPU.
- dot product: $\sum_i a_i \cdot b_i$. This function is crucial in multi-layer perceptrons DNNs, since the dot product between two vectors of real numbers **a** and **b** is routinely computed between the inputs (or each intermediate representation) and the layer weights. In addition, the standard (row-by-column) matrix by vector product is nothing more that a series of dot products, and it is the basic operation in convolutional DNNs. The dot product of two real vectors represented using the Posit data format needs to be carefully designed in hardware. This is a special function that falls within the category of *fused operations*. The key observation is that, although the final result of the dot product fits a given number of bits, the intermediate term may requires many more, in order to not loose accuracy [27]. First versions of the IEEE standard for floats (such as the 754 of 1985) did not specify the number of bits to use for fused operations. Only in the last revision of the standard (the 2008 version) it has been standardized.

C. PPU emulation library

In this sub-section we report our experience with C++ implementation of Posits on an emulation platform represented by a GPP processor, e.g. an Intel Core i7. Posits library has been implemented using a C++ template library parametrized over Posit length and exponent length. The implementation takes advantage of the representation levels discussed in Section IV. Integer ALU are used for the level 3 operations much like the SoftFloat [28] does with IEEE floating points. Moreover level 3 operations of small Posits (e.g. less than 14 bits) can be implemented in the library using lookup tables. The use of a look-up table is interesting for those applications requiring a low precision and for those computers having sufficient cache memory, like current desktop PCs. Table I summarizes the different implementation options. Thanks to the library, computations with real numbers can be performed using a fully software implementation of Posits, being able to save memory and bandwidth. In particular the library can be used for working with the C++ Eigen template matrices library [29].

D. Summary: the advantages of Posits over Floats

When performing the inference of a trained DNN in an autonomous driving application the advantages are:

1) More efficient use of bits (less bit/higher accuracy)
2) 16-bit Posits are more accurate than 16-bit Floats
3) The PPU within the processor/co-processor requires less transistors, because the comparison operator does not need to be implemented (use the one of the ALU)
4) The Sigmoid function can be easily computed in hardware, due to a numerical property of Posits.

More in general, we confirm that Posits are an interesting data format for low-precision arithmetic. In particular, using Posit16 we are widening the range of applications for which 16 bits are enough, thus saving bandwidth, storage and energy consumption in comparison to Float32.

V. CONCLUSIONS AND FUTURE WORK

To accelerate DNN computing in automotive applications, the paper has discussed the introduction of PPU as alternative to classic FPU. Once introduced the Posits as alternative representation for real numbers, we showed that 16-bit Posits should replace conventional FPUs, since they are more

accurate in case the same number of bits is used. Hardware & Software implementation issues of a PPU have been discussed, highlighting how the use of PPU would allow saving energy, mainly because it requires less transistors than an FPU.

The hardware implementation will be investigated in the context of the H2020 EPI (European Processor Initiative) project participated by relevant research and industrial partners such as CEA Tech and Kalray Corporation.

As further evolution of the work we are analyzing whether the use of Posits can be helpful even during the training phase of DNNs. Concerning the autonomous drive application, the training happens on remote servers. Such servers can be equipped with co-processors as well. Even if the training phase usually requires more bits (Float32 instead of Float16, for instance), the use of Posit32 as a replacement of Float32 for the co-processor of the remote cluster workers is advantageous for the same reasons provided for the inference phase. Assessing whether or not Posit16 are enough for the training phase of DNNs is an interesting issue left in on-going work.

ACKNOWLEDGMENTS

We would like to thank John Gustafson for his precious support in understanding the new Posit format and for having granted the right for using Figure 3, 6 and 8, which are taken from his open-access papers.

This work has been partially supported by the PRA2017 E-TEAM project (funded by University of Pisa) and the H2020 European Processor Initiative project (funded by the European Commission).

REFERENCES

[1] Intel, "Technology and computing requirements for self-driving cars," 2016.

[2] Fraunhofer(ESK), "Future vehicle software architectures," https://www.esk.fraunhofer.de/en/research/projects/adaptives_bordnetz.html, 2018.

[3] F. Pieri, C. Zambelli, A. Nannini, P. Olivo, and S. Saponara, "Consumer electronics is redesigning our cars? challenges of integrated technologies for sensing, computing, and storage," *IEEE Consumer Electronics Magazine*, vol. 7, no. 5, 2018.

[4] S. Brunner, J. Roder, M. Kucera, and T. Waas, "Automotive e/e-architecture enhancements by usage of ethernet tsn," in *Intelligent Solutions in Embedded Systems (WISES), 2017 13th Workshop on*. IEEE, 2017, pp. 9–13.

[5] C. Ebert and J. Favaro, "Automotive software," *IEEE Software*, vol. 34, no. 3, pp. 33–39, 2017.

[6] J. Schroeder, C. Berger, A. Knauss, H. Preenja, M. Ali, M. Staron, and T. Herpel, "Predicting and evaluating software model growth in the automotive industry," in *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 2017, pp. 584–593.

[7] S. Saponara and B. Neri, "Radar sensor signal acquisition and multidimensional fft processing for surveillance applications in transport systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 66, no. 4, pp. 604–615, 2017.

[8] *ISO 26262-1:2011:Road Vehicles - Functional Safety*, http://www.iso.org/, International Organization for standards (ISO), 2011.

[9] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K.-L. Lundbäck, "Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints," *Software & Systems Modeling*, pp. 1–31, 2017.

[10] P. J. Navarro, C. Fernández, R. Borraz, and D. Alonso, "A machine learning approach to pedestrian detection for autonomous vehicles using high-definition 3d range data," *Sensors*, vol. 17, no. 1, p. 18, 2016.

[11] G. Tanzmeister and D. Wollherr, "Evidential grid-based tracking and mapping," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 6, pp. 1454–1467, 2017.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[13] M. L. Gallo, I. Boybat, B. Rajendran, A. Sebastian, E. Eleftheriou *et al.*, "Mixed-precision training of deep neural networks using computational memory," *arXiv preprint arXiv:1712.01192*, 2017.

[14] P. Micikevicius, "Mixed-precision training of deep neural networks," NVidia White Paper, October 2017.

[15] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 1742–1752.

[16] J. Hou, Y. Zhu, Y. Shen, M. Li, Q. Wu, and H. Wu, "Enhancing precision and bandwidth in cloud computing: Implementation of a novel floating-point format on fpga," in *Cyber Security and Cloud Computing (CSCloud), 2017 IEEE 4th International Conference on*. IEEE, 2017, pp. 310–315.

[17] A. Rodriguez, E. Segal, E. Meiri, E. Fomenko, Y. Jim Kim, H. Shen, and Z. Barukh, "Lower numerical precision deep learning inference and training," Intel White Paper, January 2018.

[18] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on cpus," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[19] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *International Conference on Learning Representations*, 2018.

[20] J. L. Gustafson, *The End of Error: Unum Computing*. Chapman and Hall/CRC, 2015.

[21] D. Ingole, M. Kvasnica, H. De Silva, and J. Gustafson, "Reducing memory footprints in explicit model predictive control using universal numbers," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11 595–11 600, 2017.

[22] F. Glaser, S. Mach, A. Rahimi, F. K. Gürkaynak, Q. Huang, and L. Benini, "An 826 mops, 210 uw/mhz unum alu in 65 nm," *arXiv preprint arXiv:1712.01021*, 2017.

[23] J. L. Gustafson, "A radical approach to computation with real numbers," *Supercomputing Frontiers and Innovations*, vol. 3, no. 2, pp. 38–53, 2016.

[24] A. A. Jorgensen, "Apparatus for calculating and retaining a bound on error during floating point operations and methods thereof," November 2017, uS Patent 9,817,662.

[25] J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, 2017.

[26] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37, Lille, France, 07–09 Jul 2015, pp. 1737–1746.

[27] J. Koenig, D. Biancolin, J. Bachrach, and K. Asanovic, "A hardware accelerator for computing an exact dot product," in *Computer Arithmetic (ARITH), 2017 IEEE 24th Symposium on*. IEEE, 2017, pp. 114–121.

[28] J. R. Hauser, "Softfloat library, version 3e," http://www.jhauser.us/arithmetic/SoftFloat.html, 2018 (accessed March 22, 2018).

[29] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," "http://eigen.tuxfamily.org", 2010.