Experimental Results of Vectorized Posit-Based DNNs on a Real ARM SVE High Performance Computing Machine

Marco Cococcioni[1], Federico Rossi[1(✉)], Emanuele Ruffaldi[2], and Sergio Saponara[1]

[1] Department of Information Engineering, University of Pisa, Pisa, Italy
{marco.cococcioni,sergio.saponara}@unipi.it, federico.rossi@ing.unipi.it
[2] MMI s.p.a., Calci, Pisa, Italy
emanuele.ruffaldi@mmimicro.com

Abstract. With the pervasiveness of deep neural networks in scenarios that bring real-time requirements, there is the increasing need for optimized arithmetic on high performance architectures. In this paper we adopt two key visions: i) extensive use of vectorization to accelerate computation of deep neural network kernels; ii) adoption of the posit compressed arithmetic in order to reduce the memory transfers between the vector registers and the rest of the memory architecture. Finally, we present our first results on a real hardware implementation of the ARM Scalable Vector Extension.

Keywords: ARM SVE · Vectorization · Alternative representation of reals · Posit arithmetic · HPC

1 Introduction

Nowadays, Deep Neural Networks (DNNs) face new problems and challenges: on one hand, there is a need to reduce network design and computation complexity in order to better accomplish real-time tasks in resource-constrained devices. On the other hand, the trend is to address specific platform accelerators (for example, NVIDIA cuDNN for NVIDIA Graphics Processing Units (GPUs)) to significantly accelerate neural network processing in both the training and inference phases.

DNNs extensively use matrix multiplications, dot products and convolutions, highlighting the need for vectorization routines capable of increasing throughput for these operations. Although the use of GPUs in this field is important, high implementation costs and low-power requirements may prevent such components from being used. Several implementations of vector CPUs are available in most of the common processor architectures: i) Intel/AMD AVX2/SSE for ×86 processors, ii) RISC-V "V" vector extension for the RISC-V architecture, iii) ARM SVE for the ARMv8 architecture. In [1–3] we were able to produce binaries that

S. Saponara and A. De Gloria (Eds.): ApplePies 2021, LNEE 866, pp. 61–68, 2022.
https://doi.org/10.1007/978-3-030-95498-7_9

employ, respectively, ARM SVE and RISC-V vectorization. In particular, for the ARM SVE platform we were able to enable vectorization in two different tiers: one tier was the auto-vectorization approach that relies on automatic optimization from the compiler (for example, loop unrolling). The second tier involved was the use of intrinsic functions that allowed us to explicitly use ARM SVE instructions in the C++ code.

The idea behind vector extensions is to fit as much data as possible in the vector registers, that acts as very low latency memory for the vector processor. In order to increase the data we can fit in the registers and reduce the memory transfer, it is crucial to minimize the number of bits used to represent the weights of the DNNs. Several alternatives to the standard IEEE 754 32-bit floating point have been proposed: Google (Brain Float—BFloat16—[4]), Intel (Flexpoint—FP16—[5,6]) have already suggested several concepts. BFloat8 [7] is also very interesting, adding the support for stochastic rounding.

The posit^{TM} format [8–10] is one of the most promising representations that deviates from the IEEE 754 standard. In machine learning, this kind has been shown to be a great drop-in replacement for 32-bit IEEE 754 floats, using only 16 bits [11–16]. Furthermore, it has been successfully used in low-precision inference down to 8-bit posit representation with minimal network inference accuracy degradation. Moreover, as explained in [12], this number system can be used to create quick, approximated, and efficient activation functions for neural networks such as the sigmoid function by simply using the already existing CPU integer arithmetic operations.

While in [1] we were not able to profile our code on a real hardware (we used the ARM Instruction Emulator for SVE (ARMIE)), in this paper we will instead evaluate the performance of the vectorized extension of the cppPosit C++ posit arithmetic library targeting a real hardware implementation of the ARM SVE architecture.

2 Posit Number and cppPosit Library

Posit numbers are represented by a fixed length format. The overall length (*nbits*) and exponent length (*esbits*) can be modified. The posit format can have a maximum of 4 fields as in Fig. 1. Hereafter we describe the format fields:

- Sign field: 1 bit;
- Regime field: variable length, composed by a string of bits equal to 1 or 0 ended, respectively by a 0 or 1 bit;
- Exponent field: at most *esbits* bits (it can even be absent);
- Fraction field: variable length mantissa (it can even be absent too).

Given a posit$\langle nbits, esbits\rangle$, represented in 2's complement signed integer X and let e and f be exponent and fraction values, the real number r represented by X encoding is:

$$r = \begin{cases} 0, \text{ if } X = 0 \\ \text{NaN, if } X = -2^{(nbits-1)} \\ sign(X) \cdot useed^k \cdot 2^e \cdot (1+f), \text{ otherwise} \end{cases}$$

Where $useed = 2^{2^{esbits}}$ and k is strictly related to the regime length l and bitstring (b is the bit that composes the string of identical bits, e.g. in 00001 $b = 0$). If $b = 0$ the k is negative, otherwise the k is positive:

$$k = \begin{cases} -l, \text{ if } b = 0 \\ l - 1, \text{ otherwise} \end{cases}$$

In [17] we proved some interesting properties for the configuration ($esbits = 0$). Under this configuration, we could implement fast and approximated versions of common operations. We could evaluate these operations only using the arithmetic-logic unit (ALU) making them faster than the original ones computed using the FPU. These operations are the double and half operators ($2x$ and $x/2$), the inverse operator ($1/x$) and the one's complement ($1 - x$). In [1] we combined this idea with vectorization, obtaining several posit operations such as ELU and Tanh, exploiting the already existent vector integer operations in the ARM SVE vector environment.

We provide the software support for posit numbers through the cppPosit library, developed in Pisa and maintained by the authors of this work. We exploit templatization to configure the posit format. We classify posit operations into four different operational levels, identified with ($\mathcal{L}1$–$\mathcal{L}4$). Each level has increasing computational complexity (see [17]).

When in levels $\mathcal{L}3$–$\mathcal{L}4$ we need to use three different back-ends to accelerate posit operations that cannot be directly evaluated via ALU (waiting for proper posit hardware support):

- FPU back-end;
- Fixed back-end, exploiting big-integer support (64 or 128 bits) for operations;
- Tabulated back-end, generating lookup tables for most of the operations (suitable for $\langle [8, 12], * \rangle$ due to table sizes).

3 The Advantages of Vectorized CPUs

The newly introduced ARM SVE is a modern Single Instruction, Multiple Data (SIMD) for the 64-bit ARMv8 instruction set. It is intended as an evolution of the older ARM Neon vector instruction engine. The power of SVE lies in the Vector Length Agnostic (VLA) nature of the engine; indeed, there is no need to specify, at compilation time, the size of the vector registers. This dimension can be retrieved at run-time using a single assembly instruction. This design highly enhance portability of code across different ARM SVE platforms and revisions.

The VLA design is very similar to the one adopted RISC-V vector extension. The main differences between the RISC-V "V" extension and ARM SVE that we believe worht mentioning are:

- Maximum register size: while ARM SVE can only reach 2048-bits, RISC-V "V" can reach up to 16384-bits

– Register grouping: when dealing with different element sizes in the same vector loops, RISC-V can handle the wider element grouping registers so that it can be indexed as it was smaller (e.g. if we want to convert a vector of 16-bit posits to a vector of 32-bit floats).

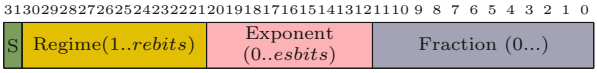

Fig. 1. Illustration of a posit$\langle 32, 9\rangle$ data type. Both the exponent and the fraction field can be absent, for specific configurations having a regime field particularly lengthy.

3.1 Vectorized CPUs and Deep Neural Networks

The most recurrent computations in deep neural networks are [1,2,17]:

– GEMM (general matrix-matrix multiplication) (training phase)
– matrix-vector multiplication (inference phase)
– matrix-matrix convolution product (both training and inference phases)
– vector-vector dot product (both training and inference phases, for computing the activations)
– non-linear activation function (both training and inference phases), computed over a vector of activations.

In this work we have used posit, since they allow to save memory for storing the weights. Moreover we and other authors have proved that posit16 is as accurate as 32-bit IEEE Floats for machine learning applications. In machine learning application even a posit8 can be accurate enough compared to 32-bit floats, thus saving 4× storage space (both on disk and, more importantly, on RAM and caches) with minimal accuracy loss.

4 Test-Bench, Methodologies and Benchmarks

In [1] we tested ARM SVE capabilities using the ARM Instruction Emulator. We ran the emulator on a HiSilicon Hi1616 CPU with 32@2.4 GHz ARMv8 Cortex-A72 cores. This emulator was able to trap all the illegal instruction interruptions coming from the execution of binaries compiled using the SVE instruction set extension. These instructions were then executed via software by the ARM Instruction Emulator. During *emulation* we were able to modify the vector register length from 128-bit to 2048-bit.

Instead, in this work we were able to use an actual hardware implementation of the ARM SVE architecture using the HPE Apollo80 machine available at University of Pisa. The Apollo80 is based on the ARMv8 A64FX core [18], the first commercial implementation of the ARM SVE architecture. In particular, the ARMv8 A64FX is the first processor to support the full feature set of the ARM SVE architecture without emulating any instruction.

This platform is particularly interesting since it will be employed in the European Processor Iniative framework [19], and it will be used as a base computing platform for the EUPEX and TEXTAROSSA EuroHPC projects.

In detail, this platform consists of 4 different blades equipped with 48 ARMv8 A64FX Cores with 512-bit vector registers, running, respectively, at 1.8 and 2.0 GHz. Each blade has access to 32 GB of High Bandwidth Memory.

In order to evaluate SVE-related performance on this machine, we used following benchmarks: i) vectorized activation functions only using posits and integer vector instructions, ii) vectorized matrix-matrix multiplication and convolution.

Moreover, we employed posit numbers to compress and decompress data across the kernels, in order to reduce memory transfers by a factor 4 (with posit$\langle 8, 0\rangle$) or 2 (with posit$\langle 16, 0\rangle$). Also compression and decompression phases were implemented using vectorization, exploiting vector integer arithmetic.

Benchmarks were compiled using the `armclang++ 20.3` compiler, based on `LLVM 9.0.1` and executed on the aforementioned Apollo80 machine, running `CentOS Linux release 7.9.2009`.

5 Experimental Results and Discussion

Figure 2 shows the performance of the activation function benchmarks on the Apollo80 machine. The benchmarks consisted in the computation of sigmoid and extended linear unit (ELU) on 4096-wide vectors (even if the hardware only supports 512-bit). Each computation was repeated 100 times and the average computation time was reported.

As reported, the computation of the two activation functions using posit$\langle 8, 0\rangle$ benefits from the reduction in size of the format. This is because most of the steps of the activation function computation is performed using `int8_t` for posit$\langle 8, 0\rangle$ and `int16_t` for posit$\langle 16, 0\rangle$.

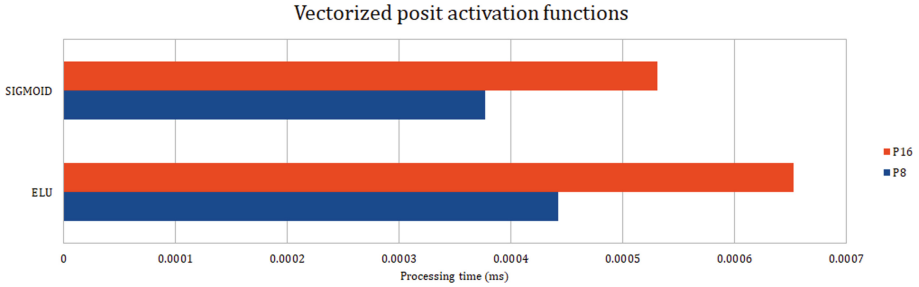

Fig. 2. Processing time of vectorized activation functions on a 4096-element vector with a 512-bit vector register length.

Figure 3 shows the performance of the kernel benchmarks, using posit$\langle 8, 0\rangle$ and posit$\langle 16, 0\rangle$. These benchmarks consisted in the computation of: i) dot product between vector of 4096 elements, ii) convolution on a 128×128 image with a

3×3 filter, iii) matrix-matrix multiplication between square matrices of 128×128 elements.

As reported, the benefit in reducing the information size is not as effective as in the previous case. The issue is that in this case, we could not use posits in every step of computation (of course ARM SVE lacks dedicated posit instructions). This means that we used posits just for data compression and decompression at the start and at the end of the kernels. For example, in the convolution kernel, we decompress the posit input to float using our vectorized routine, then we compute the convolution using native vectorized float support from ARM SVE and finally we compress the result back to posits.

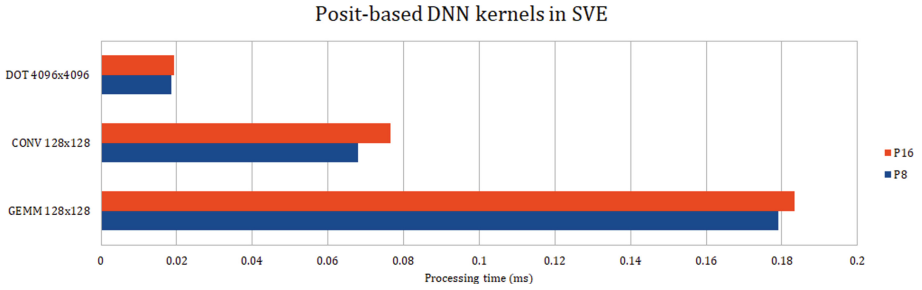

Fig. 3. Processing time of vectorized DNN kernels with a 512-bit vector register length (DOT: vector-vector dot product, CONV: matrix-matrix convolution, GEMM: General matrix matrix multiplication).

Figures 4 and 5 show the measured speedup from the emulated machine to the real hardware implementation. The speedup is computed as $t_{\text{HPE80}}/t_{\text{ARMIE}}$. Since we already proved that we can get better timing performance using posit$\langle 8, 0\rangle$ instead of posit$\langle 16, 0\rangle$, we reported the speedup relative to posit$\langle 8, 0\rangle$ computations. As reported, the speedup spans from at least $\sim$11, in the case of the GEMM function, up to $\sim$1500 in the case of the ELU function.

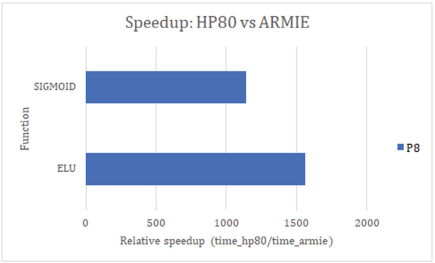

Fig. 4. Relative speedup of activation functions with 512-bit vector register length.

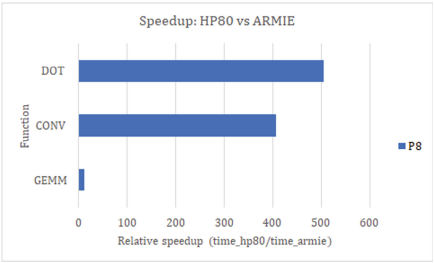

Fig. 5. Relative speedup of DNN kernels with a 512-bit vector register length.

6 Conclusions

In a previous work, we designed posit-based and vectorized operations on an ARM 64bit SVE emulator. The operations considered are the most time consuming ones in deep neural networks. In the present work we compared the impact of vectorization on a real machine. By using such machine, we were able to assess the true speedup due to vectorization, which turned out to be remarkable (with a speedup factor from 11× to 1500×, depending on the executed task). Future works will involve the combination of presented algorithms with MPI, to enable multi-processor or multi-node computation of deep neural networks (e.g. exploiting all the blades and cores of the HP80).

Acknowledgments. Work partially supported by H2020 projects (EPI grant no. 826647, https://www.european-processor-initiative.eu and TEXTAROSSA grant no. 956831, https://textarossa.eu) and partially by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence). We thank the personnel of the Green DataCenter of the University of Pisa (https://start.unipi.it/en/computingunipi). In particular, we thank Prof. P. Ferragina, Dr. M. Davini and Dr S. Suin, for having provided us with the computational resources that have been used in the experimental section.

References

1. Cococcioni, M., Rossi, F., Ruffaldi, E., Saponara, S.: Fast deep neural networks for image processing using posits and ARM scalable vector extension. J. Real-Time Image Process. **17**(3), 759–771 (2020). https://doi.org/10.1007/s11554-020-00984-x
2. Cococcioni, M., Rossi, F., Ruffaldi, E., Saponara, S.: Vectorizing posit operations on RISC-V for faster deep neural networks: experiments and comparison with ARM SVE. J. Neural Comput. Appl. **33**, 575–585 (2021). https://doi.org/10.1007/s00521-021-05814-0
3. Cococcioni, M., Rossi, F., Ruffaldi, E., Saponara, S.: Faster deep neural network image processing by using vectorized posit operations on a RISC-V processor, In: Real-Time Image Processing and Deep Learning 2021, Kehtarnavaz, N., Carlsohn, M.F. (Eds.,) International Society for Optics and Photonics. SPIE, vol. 11736, pp. 19–25 (2021). https://doi.org/10.1117/12.2586565
4. Burgess, N., Milanovic, J., Stephens, N., Monachopoulos, K., Mansell, D.: Bfloat16 processing for neural networks. In: 2019 IEEE 26th Symposium on Computer Arithmetic (ARITH), pp. 88–91 (2019)
5. Koster, U., et al.: Flexpoint: an adaptive numerical format for efficient training of deep neural networks. In: Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017) (2017)
6. Popescu, V., Nassar, M., Wang, X., Tumer, E., Webb, T.: Flexpoint: predictive numerics for deep learning. In: Proceedings of the 25th IEEE Symposium on Computer Arithmetic (ARITH 2018), pp. 1–4 (2018)
7. Mellempudi, N., Srinivasan, S., Das, D., Kaul, B.: Mixed precision training with 8-bit floating point (2019)

8. Gustafson, J.L.: The End of Error: Unum Computing. Chapman and Hall/CRC (2015)
9. Gustafson, J.L.: A radical approach to computation with real numbers. Supercomput. Front. Innov. **3**(2), 38–53 (2016)
10. Gustafson, J.L., Yonemoto, I.T.: Beating floating point at its own game: posit arithmetic. Supercomput. Front. Innov. **4**(2), 71–86 (2017)
11. Cococcioni, M., Rossi, F., Ruffaldi, E., Saponara, S.: Novel arithmetics to accelerate machine learning classifiers in autonomous driving applications. In: Proceedings of the 26th IEEE International Conference on Electronics Circuits and Systems (ICECS 2019)
12. Cococcioni, M., Rossi, F., Ruffaldi, E., Saponara, S.: A fast approximation of the hyperbolic tangent when using posit numbers and its application to deep neural networks. In: Saponara, S., De Gloria, A. (eds.) ApplePies 2019. LNEE, vol. 627, pp. 213–221. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-37277-4_25
13. Cococcioni, M, Ruffaldi, E, Saponara, S.: Exploiting posit arithmetic for deep neural networks in autonomous driving applications. In: 2018 International Conference of Electrical and Electronic Technologies for Automotive, pp. 1–6. IEEE (2018)
14. Carmichael, Z., Langroudi, H.F., Khazanov, C., Lillie, J., Gustafson, J.L., Kudithipudi, D.: Conference exhibition (DATE), pp. 1421–1426. IEEE (2019)
15. Langroudi, H.F., Carmichael, Z., Gustafson, J.L., Kudithipudi, D.: Positnn framework: tapered precision deep learning inference for the edge. In: 2019 IEEE Space Computing Conference (SCC), pp. 53–59. IEEE (2019)
16. Cococcioni, M., Rossi, F., Ruffaldi, E., Saponara, S., de Dinechin, B.D.: Novel arithmetics in deep neural networks signal processing for autonomous driving: challenges and opportunities. IEEE Signal Processing Magazine. **24**, 38(1), 97–110 (2020)
17. Cococcioni, M., Rossi, F., Ruffaldi, E., Saponara, S.: Fast approximations of activation functions in deep neural networks when using posit arithmetic, Sensors, **20**(5) (2020). www.mdpi.com/1424-8220/20/5/1515
18. Fujitsu Processor A64FX. www.fujitsu.com/global/products/computing/servers/supercomputer/a64fx/ Accessed 4 June (2021)
19. European Processor Initiative, an H2020 project. www.european-processor-initiative.eu/ (2019)