Dynamic Reconfigurable FPU for Next-Generation Transprecision Computing

Guilherme Dias*†, Luís Crespo†, Pedro Tomas†, Nuno Roma†, Nuno Neves†

*Escola Politécnica, Universidade de São Paulo, Brasil
†INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
E-mail: guiadias@usp.br, {luis.crespo, pedro.tomas, nuno.roma, nuno.neves}@inesc-id.pt

***Abstract*—Recent limitations in technology scaling have emphasized the need for energy-efficient computing architectures that can dynamically adjust operand precision based on the real-time requirements of the application, without compromising result accuracy. This adaptability also creates opportunities to enhance throughput and optimize hardware utilization. Furthermore, the use of lower-precision formats (e.g., 16-bit) often releases portions of the arithmetic datapath, allowing these resources to be reallocated for increased vector parallelism. In this context, we present a novel transprecision Floating-Point Unit (FPU) that supports all IEEE 754 data types (double, single, half-precision), as well as the bfloat16 and DLFloat formats. The unit features dynamic precision tuning of operands, enabling increased throughput through vectorization and improved energy efficiency. The proposed design was implemented using a 28nm UMC technology process, achieving a peak energy efficiency of 152 GOPS/W as a result of new precision adaptation capabilities.**

***Index Terms*—Floating-point arithmetic; Dynamic Transprecision; Vectorization/SIMD; Energy Efficiency.**

I. Introduction

To guarantee general-purpose deployment and numerical accuracy, conventional computing architectures statically define the precision of data types (e.g. 32, 64 bits), justifying the ubiquitous adoption of the IEEE-754 floating-point standard. Under this premise, the transprecision computing paradigm recognizes the opportunity to use approximate (narrower) data types for the required computations without significantly degrading the attained results. For instance, 8-bit and 16-bit data types, which are already supported in some floating-point formats such as Posit [1], DLFloat [2], Microsoft Floating-Point [3], among others have been adopted in the domain of ML/AI applications [3]–[6]. In fact, this trend has even pushed the creation of an IEEE working group to define a set of new standard low-precision formats [7]. By taking advantage of such lower precision formats, transprecision computing frameworks attempt to dynamically control and adjust the approximation through software and/or hardware, with the goal of optimizing the energy efficiency [8]–[11].

Many recent works have attempted to tackle the hardware side of this problem. Mach et al. [10] introduced an architecture with support for multiple input formats and arbitrary bit widths, as well as mixed format operations. Crespo et al. [11] presented a vector multiply-accumulate (MAC) unit with efficient unified support for both IEEE-754 and Posit operands. Zhang et al. [12] implemented a vector IEEE-754 fused multiply-add (FMA) unit with support for up to quadruple-precision, mixed-format FMA and dot-product operations. However, all these units are not capable of autonomously determining the most appropriate operand precision at runtime.

While several solutions already support multi-precision architectures, they are often configured statically. In contrast, the dynamic FPU re-configuration, in runtime, has been explored to a lesser extent, and usually only with arithmetic units that do not support multiple formats. As an example, Linhares et al. [9] presented an IEEE-754 transprecise FPU expanded with a tag field that determines, manually or dynamically, the precision that its operands should be in for the operation to be performed. However, this unit does not support the alternative formats previously mentioned, which may arguably limit its performance in relevant transprecision workloads.

Kaul et al. [8] implemented an FMA unit with support for 1-way 24-bit, 2-way 12-bit and 4-way 6-bit mantissas. It also includes a runtime certainty tracking circuit, where inaccurate results trigger the operation to be repeated in a higher precision mode. Despite being highly performant, its transprecision applications are once again limited due to only supporting single-precision operands.

Accordingly, there is an opportunity to further advance the design of new FPU architectures with wider support for low-precision formats and the ability to dynamically determine the most appropriate operand precision at runtime, aiming to increase throughput and energy efficiency through in-place vectorization. Thus, this paper proposes a new dynamically vectorized fused multiply-accumulate (FMAC) FPU architecture with variable precision and multi-format floating-point arithmetic. Besides providing RISC-V vector [13] compliant and accumulation operations. The main contributions are:

- An efficient vector 64-bit FMAC FPU architecture with

Work supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project 2022.04020.PTDC (DOI: 10.54499/2022.04020.PTDC). We acknowledge contributions from projects 2022.06780.PTDC (DOI: 10.54499/2022.06780.PTDC), UIDB/50021/2020 (DOI: 10.54499/UIDB/50021/2020), and from the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 (EPI SGA2). The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland.

979-8-3315-2212-4/25/$31.00 ©2025 IEEE

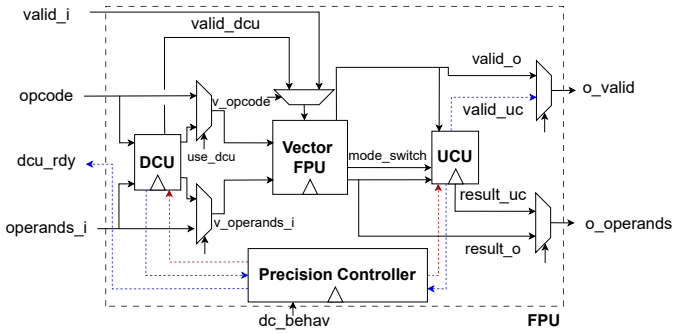

Fig. 1. FPU block diagram. Red and blue arrows represent the control and status signals, respectively. The *mode_switch* format selection signal is encoded in the *opcode* input signal.

simultaneous support for operands in the IEEE 754 (double, single, and half-precision), bfloat16, and DLFloat formats, with support for accumulation operations;
- A fully vectorized arithmetic datapath, capable of dynamically configuring to use operands of different vector precisions (1x64-bit/2x32-bit/4x16-bit operand vectors);
- A new precision controller unit capable of automatically and dynamically modifying the FPU precision format of the input operands and the adopted level of vectorization;
- New precision adjustment policies that allow the controller unit to configure the FPU according to user-selected modes of operation and specific characteristics of the input data, with the goal of maximizing arithmetic throughput and efficiency.

II. Dynamic Floating-Point Unit Architecture

The proposed architecture comprises four parts (see Fig. 1): a **vector FPU**, deploying a variable-precision, multi-format floating-point arithmetic datapath; a **Downcast Unit** (DCU); and a **Upcast Unit** (UCU), which perform format conversions on the operands; and a **Precision Controller**, which decides on format conversions, balancing precision with the considered vectorization, depending on user-defined execution policies.

A. Vector FPU Architecture

The vector FPU (see Fig. 2) supports 16 operations, such as addition, multiplication, MAC, and comparison (see Table I). It operates on either IEEE-754 double, single and half-precision formats, or on DLFloat and bfloat16 16-bit formats. By using a dynamically vectorizable datapath, the input operands may be vectorized two-way (single-precision) or four-way (half-precision), always being encoded in 64-bit words.

Accordingly, it features two operating modes: *fixed precision* and *dynamic precision*. In the former, a set of three 64-bit vector operands (A, B, C) of any supported format may be input to the FPU, where each vector is composed of a 1x64, 2x32 or 4x16 bits, and in the latter up to twelve vectors in IEEE double precision may be input. However, while on the *fixed precision* mode the operation is performed using the operands native precision, using *dynamic precision* the DCU will try to downcast each set of operands to a lower precision

TABLE I
Vector FPU operations. MANT sets the exponent vector to zero; NEGEXP negates it; Acc denotes the accumulated value.

MUL $A \times B$	ADD $A + B$	SUB $A - B$	MADD3 $(A \times B) + C$
MSUB3 $(A \times B) - C$	NMADD3 $-(A \times B) + C$	NMSUB3 $-(A \times B) - C$	MADD2 $(A \times B) + Acc$
NMADD2 $-(A \times B) + Acc$	MAX3 $(A > B)?A : C$	MIN3 $(A < B)?A : C$	EQ3 $(A == B)?A : C$
NEQ3 $(A! = B)?A : C$	MANT	NEGEXP	NOPSHF Shuffle if < 64b

(see section II-B2). Hence, when converted to 16-bit format, a peak throughput of 4 operations per cycle is achieved.

1) Input Processing: The vectorized FPU has three independent decode units, one for each input, responsible for extracting the sign, exponent and fraction fields from each operand, packing them in a dedicated internal unified representation, and detecting special operands (subnormals, infinities and NaNs). This unified vector representation (see Fig. 3) was especially designed to allow packing vectors of floating-point operands using distinct precisions and formats. It is composed of three packed vectors (one for each of the floating-point format fields). The sign vector is 4 bits wide, to account for the supported 4-way vectorization. The input exponent vector is 40 bits wide, allowing four bfloat16 exponent operands to be represented, with two additional padding bits to account for overflows during exponent addition. Lastly, the mantissa vector is 56 bits wide, to accommodate all the combinations of supported precisions of mantissa operands, plus one implicit bit and an overflow bit. The rightmost zero bits fill the empty spaces in the representation.

2) Vector Multiplication: The FPU's multiplier unit is 56 bits wide, being composed of 16 individual 14-bit multipliers as illustrated in Fig. 4. When multiplying 16-bit operands, only multipliers 0, 5, 10 and 15 are active. For single-precision, multipliers 1, 4, 11 and 14 are also enabled and for double-precision all partial multipliers are activated. The final product is obtained by adding the corresponding partial products (pp) according to the alignment layout shown in Fig. 4. After multiplication, the 112-bit product is conveniently shifted to follow the internal unified format and expanded to full 128-bit precision, along with the mantissa of operand C. Lastly, the exponent vectors of the operands are added and the product is corrected in case of any eventual overflow.

3) Vector Accumulator: Before addition, the addend is selected (either operand C or an accumulated value) and a vector barrel shifter aligns both operands by right shifting the mantissa of the operand with the smallest exponent, in an extent given by their exponent difference. They are then added in a vectorized 128-bit adder. Afterwards, the result sign is calculated along with any exception cases.

4) Normalization and Encoding: A leading zero anticipator (LZA - adapted from [14]) and vector barrel shifters are used to normalize the exponent and the fraction. The fraction is left shifted for normalized exponent values and right shifted for subnormal ones. The exponent is corrected according to the LZA output or set to zero for subnormal values. The full

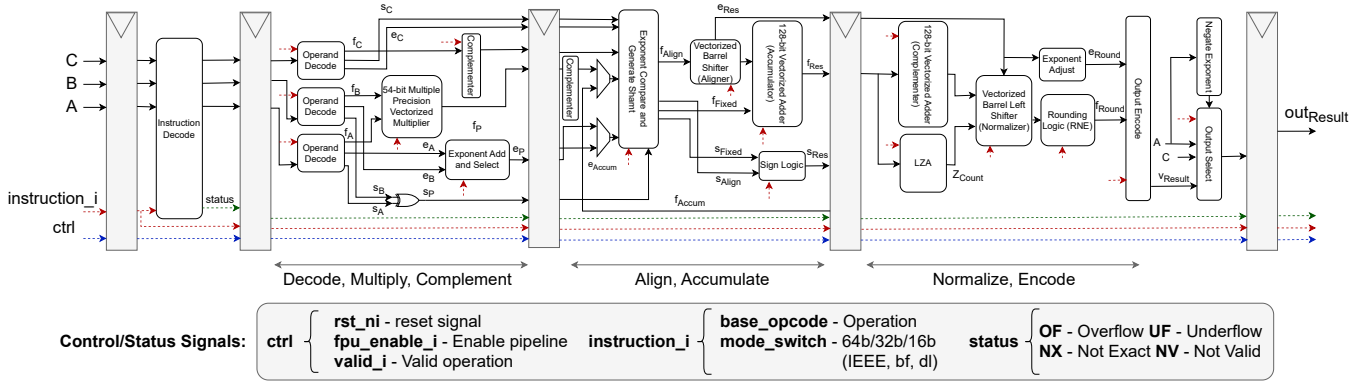

Fig. 2. Block diagram representing the datapath of the vectorized FPU.

precision mantissa is rounded (in round to nearest, ties to even mode). Finally, exception logic asserts the necessary flags and the final 64-bit vector result is sent to the output ports.

B. Automatic Precision Controller

The reconfigurable datapath consists of three units, as depicted in Fig. 1. The first is the DCU, which is responsible for downcasting IEEE double-precision operands to any other of the supported formats, sending these to the vector FPU. The second is the UCU, which performs the reverse operation, upcasting the FPU results back to double-precision before being sent to the output ports. The final unit, the Precision Controller, coordinates the last two through a FSM.

1) Precision Control Unit: These units are managed by a precision control unit, driven by a finite state machine. This control unit defines the most appropriate precision level, checks for overflows during the conversion, and organizes the operand output. Its behavior can be set to one of three modes:

- The first mode forces the operands into the format provided in the format selection signal (*mode_switch*).
- The second mode attempts to convert the input operands to the specified format. If this conversion would lead to overflow in any of the operands, the next upper precision format that does not overflow is used instead. If one or more operands may not be represented in a single half-precision set (IEEE half-precision, bfloat16, DLFloat), it is split into two single-precision ones, sent to the vector FPU one after the other.
- The third mode is fully autonomous. The mantissas of each input are analyzed to determine an appropriate common format. For each one, the number of zeroes after each leading one is counted. If this number is equal to or greater than a predefined threshold (specified during synthesis as a design parameter) for any of them, then the number of bits up to that leading one is counted. This number is then used to choose a new format. The selected format is the one with the smallest mantissa bit-width that is still equal to or greater than this value.

2) Downcast Unit: The DCU, depicted at the left of Fig. 5, comprises the following sections: an input decode section, to split the double-precision operands stored in the input registers; a mantissa analysis section; and an exponent processing section to subtract the required bias from the exponents ($bias_{DP} - bias_{Fmt}$). This section also includes a vector adder and detects eventual overflows, sending this information to the control unit. Once defined, the proper precision is used to set the output exponent vectors and to round the necessary leftmost mantissa bits. The final section packs the operands appropriately and handles special values, preserving them after the downcast, flushing underflows to zero.

3) Upcast Unit: The UCU, depicted at the right of Fig. 5, receives the operands from the vector FPU (together with their

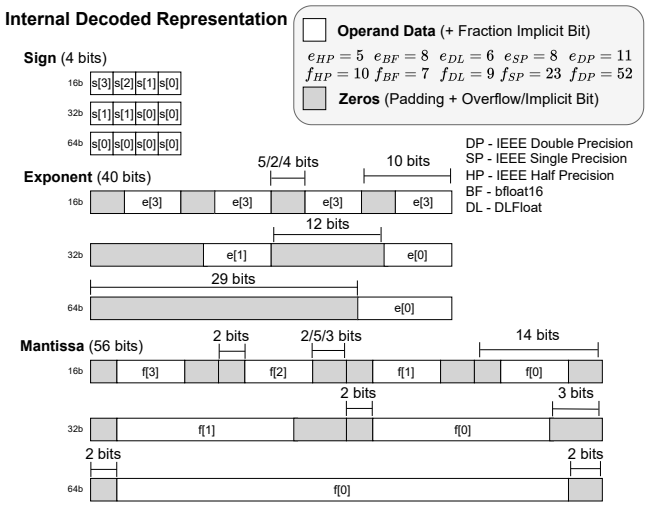

Fig. 3. Unified representation of decoded signs, exponents and mantissas. The padding for 16-bit operands varies depending on format (HP/BF/DL).

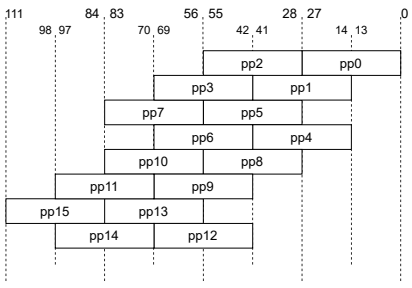

Fig. 4. Layout of the 16 14-bit partial multipliers in the multiplication unit.

TABLE II
IMPLEMENTATION METRICS AND FUNCTIONALITY COMPARISON BETWEEN OUR AND OTHER RELEVANT STATE-OF-THE-ART FPUS.

FPU	Tech. (nm)	Input Formats	Format Conversions	Delay (ns)	Num. Bits	Area (μm^2)	Power (mW)	Energy Efficiency (GOPS/W)	Area Efficiency (10^{-6}GOPS/μm^2)	EDP ($10^{-22} J.s$)
Ours	28	IEEE, bfloat16, DLFloat	Yes (Automatic)	2.5	16/32/64	53580	19.9	80.4/40.2/20.1	29.9	3.2/6.2/12.4
[12]	90	IEEE	No	1.5	16/32/64	180610	43.8	61.6/30.8/15.9	14.9	2.5/4.9/9.9
[11]	28	IEEE, Posit	Yes (Static)	1.5	8/16/32	51563	99	13.1/6.5/3.3	25.2	5.6/11.1/22.3
[8]	32	IEEE	No	0.69	32	45000	72	162	32.2	3.4
[10]	22	IEEE, Custom FP16/FP8	Yes (Static)	1.08	8/16/32/64	49153	57.4	129.3/64.6/32.3/16.2	151	0.8/1.7/3.4/6.7

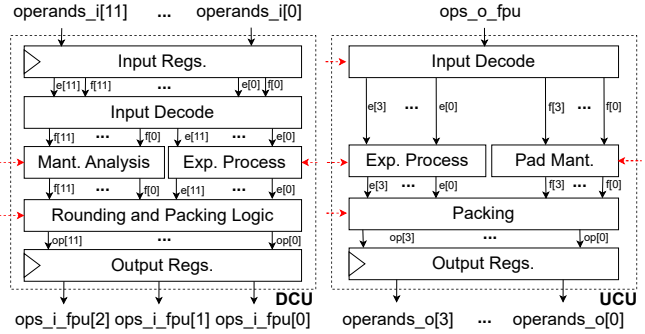

Fig. 5. Block diagrams of the DCU and UCU. The red arrows represent the format selection signal from the control unit. All depicted input and output operands are 64-bits wide, potentially vectorized into two or four lanes.

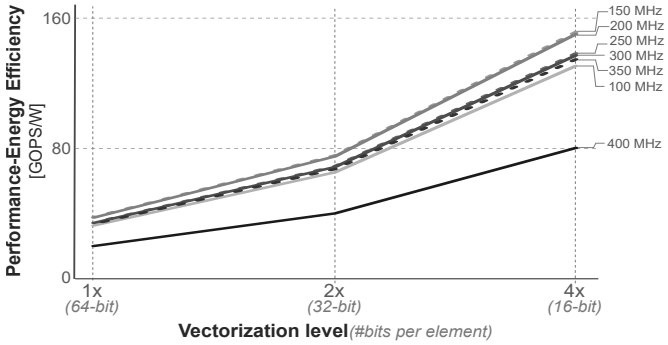

Fig. 6. Energy efficiency values for different operating frequencies.

precision information), decoding them to handle infinite, zero, NaN and subnormal values (similarly to the DCU). Conversion back to double-precision is done by zero padding the mantissas to the left and by adding the bias back to the exponents, depending on the format of the inputs.

III. IMPLEMENTATION AND EXPERIMENTAL EVALUATION

The proposed FPU design was described in System Verilog and synthesized to 28nm UMC technology process. The resulting implementation operates at a highest attainable frequency of 400 MHz under normal conditions (1.05V, 25 ºC), with the critical path being the multiplication stage. Power and area metrics were obtained by logical synthesis using Cadence Genus 21.15. The functional validation was done with AMD Xilinx Vivado 2022.1, by using a comprehensive set of test vectors generated with TestFloat [15].

A. Hardware Resources Analysis

The obtained synthesis results evidence a total area of $53580\mu m^2$ and a power consumption of $19.9mW$. When broken down into the FPU's stages, the vectorized FPU accounts for $45151\mu m^2$ (84%), divided into $4931\mu m^2$ (11%) for decode, $15944\mu m^2$ (35%) for multiplication, $6484\mu m^2$ (14%) for accumulation and $9340\mu m^2$ (21%) for normalization and encoding stages, the remaining $8452\mu m^2$ (19%) are occupied by collapsed multiplexers and other logic. The DCU and UCU account for $6507\mu m^2$ and $1170\mu m^2$, respectively, with other logic accounting for $752\mu m^2$, adding up to $8429\mu m^2$ (16%).

B. Comparison with Related Work

Table II highlights the hardware resources, power consumption and other characteristics of our proposed FPU with respect to other similar state-of-the-art units.

Despite using different technology processes, it can be observed that the overall silicon area is comparable to that of the other state-of-the-art units analyzed, particularly [8], [10], [11]. When compared with the dynamically configurable unit from [8] it shows a slight area increase of 20% due to the new features introduced by the proposed FPU, these include offering added support for 64 and 16-bit floating-point formats, which are not supported in [8]. Nonetheless, we still achieve a 2.7x reduction in power consumption when compared to [8] while offering further functionality.

Despite the added functionality of the proposed FPU, its power consumption is the lowest when compared to the other FPUs [10], [12]. Peak energy efficiency metrics were obtained at different frequencies and precisions. At 400MHz, our unit has a peak energy efficiency of 80.4 GOPS/W which is comparable to the other state-of-the-art works. It reaches a maximum of 152 GOPS/W at 150MHz (see Fig. 6), which is very close to the values obtained with the other dynamic unit [8]. The maximum area efficiency values were obtained at 350MHz, at 29.5×10^{-6} GOPS/μm^2. The obtained energy-delay product values are comparable to those of the other units.

IV. CONCLUSION

This paper presented the development of a novel transprecision Floating-Point Unit (FPU) that supports all IEEE 754 data types (double, single, and half-precision), in addition to bfloat16 and DLFloat formats. It features dynamic precision adjustment of operands and an efficient reallocation of the freed portions of the arithmetic datapath to enhance throughput through vector parallelism. Implemented using a 28nm UMC technology process, the unit demonstrated competitive characteristics in terms of area ($53.580\mu m^2$) and energy efficiency, while achieving a lower power consumption of 19.9 mW. This power consumption can be further reduced by adjusting the operating frequency, highlighting the FPU's potential for use in energy-efficient hardware accelerators.

REFERENCES

[1] Gustafson and Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomput. Front. Innov.: Int. J.*, vol. 4, no. 2, p. 71–86, jun 2017. [Online]. Available: https://doi.org/10.14529/jsfi170206

[2] A. Agrawal, S. M. Mueller, B. M. Fleischer, X. Sun, N. Wang, J. Choi, and K. Gopalakrishnan, "Dlfloat: A 16-b floating point format designed for deep learning training and inference," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, June 2019, pp. 92–95.

[3] B. Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner, A. Forin, H. Zhu, T. Na, P. Patel, S. Che, L. C. Koppaka, X. Song, S. Som, K. Das, S. Tiwary, S. Reinhardt, S. Lanka, E. Chung, and D. Burger, "Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[4] Shibo Wang, Pankaj Kanwar, "Bfloat16: The secret to high performance on cloud tpus," https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus, 2019, accessed: 2024-01-03.

[5] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. luc Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, and J. Ross, "In-datacenter performance analysis of a tensor processing unit," 2017. [Online]. Available: https://arxiv.org/pdf/1704.04760.pdf

[6] Z. Carmichael, S. H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson, and D. Kudithipudi, "Deep positron: A deep neural network using the posit number system," *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1421–1426, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:54459662

[7] IEEE SA Working Group P3109, "IEEE Working Group P3109 Interim Report on 8-bit Binary Floating-point Formats," IEEE, Tech. Rep., 2024.

[8] H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, F. Sheikh, R. Krishnamurthy, and S. Borkar, "A 1.45ghz 52-to-162gflops/w variable-precision floating-point fused multiply-add unit with certainty tracking in 32nm cmos," in *2012 IEEE International Solid-State Circuits Conference*, Feb 2012, pp. 182–184.

[9] A. Carvalho and R. Azevedo, "Towards a transprecision polymorphic floating-point unit for mixed-precision computing," in *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2019, pp. 56–63.

[10] S. Mach, F. Schuiki, F. Zaruba, and L. Benini, "Fpnew: An open-source multiformat floating-point unit architecture for energy-proportional transprecision computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 774–787, April 2021.

[11] L. Crespo, P. Tomás, N. Roma, and N. Neves, "Unified posit/ieee-754 vector mac unit for transprecision computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 5, pp. 2478–2482, 2022.

[12] H. Zhang, D. Chen, and S.-B. Ko, "Efficient multiple-precision floating-point fused multiply-add with mixed-precision support," *IEEE Transactions on Computers*, vol. 68, no. 7, pp. 1035–1048, July 2019.

[13] A. Waterman, K. Asanovic *et al.*, "RISC-V "V" Vector Extension," 2021.

[14] N. Neves, P. Tomás, and N. Roma, "A reconfigurable posit tensor unit with variable-precision arithmetic and automatic data streaming," *Journal of Signal Processing Systems*, vol. 93, 12 2021.

[15] John Hauser, "Berkeley testfloat," https://jhauser.us/arithmetic/TestFloat.html, 2018, accessed: 2024-09-11.