POD-RACING: Bulk-Bitwise to Floating-Point Compute in Racetrack Memory for Machine Learning at the Edge

Sébastien Ollivier, Xinyi Zhang, Yue Tang, Chayanika Choudhuri, Jingtong Hu, and Alex K. Jones,
University of Pittsburgh, Pittsburgh, PA, 15260, USA

Convolutional neural networks (CNNs) have become a ubiquitous algorithm with growing applications in mobile and edge settings. We describe a compute-in-memory (CIM) technique called POD-RACING using Racetrack memory (RM) to accelerate CNNs for edge systems. Using transverse read, a technique that can determine the number of "1"s in multiple adjacent domains, POD-RACING can efficiently implement multioperand bulk-bitwise and addition computations, and two-operand multiplication. We discuss how POD-RACING can implement both variable precision integer and floating point arithmetic using digital CIM. This allows both CNN inference and on-device training without expensive data movement to the cloud. Based on these functions we demonstrate the implementation of several CNNs with backpropagation using RM CIM and compare these to the state-of-the-art implementations of CNN inference and training. During training, POD-RACING improves efficiency by 2×, energy consumption by ≥27%, and increases throughput by ≥18% versus a state-of-the-art FPGA accelerator.

Edge computing has become increasingly attractive for accelerating machine learning algorithms, such as convolutional neural networks (CNNs), to support the needs of mobile applications. However, edge systems must adhere to constraints often referred to as SWaP (size, weight, and power). For CNN acceleration, field programmable gate arrays (FPGA) are studied as the best possible acceleration engines for low latency while meeting the energy requirements of these edge systems but, are limited by the need for intensive off-chip memory accesses. Compute-in-memory (CIM) is proposed to alleviate this bottleneck. Unfortunately, crossbar-based solutions require high-energy digital/analog conversion, which is inappropriate for mobile devices and DRAM-based solutions have not demonstrated sufficient precision to implement training. Spintronic *Racetrack memory*[1] (RM) is attractive for edge systems as it has the necessary density, i.e., between 1-4F^2 per cell, while not suffering from endurance concerns of other tiered memory candidates, such as phase-change and resistive memories. It also has a low-energy consumption of circa 0.1 pJ[2] per write and a low access latency of circa 1-ns generating excitement for use as main memory,[3,4] particularly for SWaP constrained systems.

We present *precision optimized deep-learning* (POD) using *Racetrack arithmetic computed in-memory for native gradient-descent* (RACING). POD-RACING is the first digital CIM implementation of floating-point multiply-accumulate designed to implement full CNN algorithms under SWaP constraints.

With POD-RACING CIM acceleration of deep learning, we achieve as much as 5× higher performance than the state-of-the-art DRAM CIM, which leverages ternary (bulk-bitwise and summation) weight calculations[5,6] with a nearly 50% reduction in power. POD-RACING is 2.8× faster and more than 3× more energy efficient for integer precision (multiplication and addition) than the state-of-the-art RM CIM. We also achieve 18%–74% performance improvement and 26%–81% reduction in power compared to a low-energy FPGA for 32-bit floating-point precision online training targeting small to moderate CNNs. In particular, POD-RACING makes the following contributions.

0272-1732 © 2022 IEEE
Digital Object Identifier 10.1109/MM.2022.3195761

This article has been accepted for publication in IEEE Micro. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/MM.2022.3195761

- POD-RACING is, to the best of our knowledge, the first RM CIM approach to implement floating-point addition and multiplication.
- We propose floating-point CIM designed to conduct multioperand floating-point addition.
- We show that POD-RACING outperforms and provides better efficiency for both CIM (inference) and FPGA (training) targeting edge systems.

The remainder of this article is organized as follows. First, we provide the necessary background on CNNs and RM. Next, we describe the basic concepts of POD-RACING, starting with its architecture and how to perform integer operations. We then explain how to perform floating-point multiplication with RM CIM followed by an algorithmic-level explanation on how to extend similar approaches to perform FP multioperand addition. We provide experimental results comparing improvements of POD-RACING with DRAM CIM for inference and an FPGA accelerator, followed by conclusions.

BACKGROUND

In this section, we first introduce the elements that compose the CNN inference and the additional operations required for training. Next, we provide a brief introduction to RM.

Convolutional Neural Network

CNN's are primarily based on the convolution operation, which is a windowed pointwise multiplication accumulation of multiple channels of input features with a set of weights to generate output features. As an example, for the input features $\mathbf{I}$ and weights $\mathbf{K}$ of size $N \times R_{\text{in}} \times C_{\text{in}}$ and $M \times N \times 3 \times 3$, respectively, the convolution operation for the window at m (output channel index), r (row), and c (column) is

$$\text{Conv}(\mathbf{I}, \mathbf{K})(m, r, c) = \sum_{n=0}^{N-1} \sum_{j=0}^{2} \sum_{t=0}^{2} \mathbf{K}_{m,n,j,t} \times \mathbf{I}_{n,r+j,c+t}$$

where M is the number of output channels, N is the number of input channels, $R_{\text{in}} \times C_{\text{in}}$ is the size of an input feature map. The inference operation requires convolution steps broken up with activation layers composed of *pooling layers* to reduce the dimensionality of input matrices through average or maximum value operations and *ReLU function*, a linear function that will output the input if positive and zero otherwise. Once these convolution layers are completed, *fully connected layers* are used to provide the classification result. The fully connected layers can be mathematically written as $\text{ReLU}(\mathbf{W}\mathbf{x} + \mathbf{b})$.

Training of the CNN includes forward propagation, loss backpropagation, and weight update. During the forward propagation, which is the same as in inference, the values at each activation layer are stored for the weight update. The loss is calculated by a loss function, such as cross-entropy loss.[7] After calculating the loss of the last layer, the loss is propagated layer by layer until reaching the first layer of the CNN, by a process that includes weight rotation, convolution, and channelwise accumulation. Based on the loss backpropagation, the weights are updated in each layer individually. The operations in weight updates are depthwise convolution, elementwise multiplication, and elementwise subtraction.

While deep learning with CNNs presumes calculations with floating-point values, CNN inference calculations can often be reduced to integer computation with as few as 8-bits achieving reasonable accuracy. Recent DRAM CIM work has shown that in many cases this can be further reduced to ternary $w \in \{-1, 0, 1\}$[5] or even binary $w \in \{0, 1\}$ computations[8] operations to replace the multiplications. However, online training for all but the simplest CNNs still requires full 32-bit floating-point computations to work properly. Without this accuracy, the weight updates can be ineffective and possibly even detrimental.

Next, we describe the basics of RM that provides the foundation for CIM acceleration of CNN functions.

RM Fundamentals

Spintronic RM is made of ferromagnetic nanowires consisting of many magnetic domains separated by domain walls (DWs), as shown in Figure 1. Each domain has its own magnetization direction such that binary values are represented by the magnetization direction of each domain, either parallel/antiparallel to a fixed reference. For a planar nanowire, several domains share one/few access point(s) (APs) for read and write operations.[9] DW motion is controlled by applying a short current pulse laterally along the nanowire governed by SL. An access requires *shifting* the

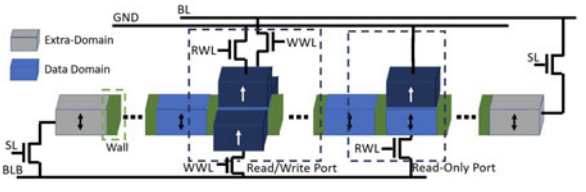

FIGURE 1. Anatomy of a DW memory nanowire.

TABLE I. POD-RACING compared to accelerators.

Inference improvement compared to CIM					
Benchmark	**Target**	**Throughput**	**Power**	**Efficiency**	
		FPS	**W**	**FPS/W**	
LeNet-5	DRAM[6]	8330	–	–	
Ternary[5]	POD-R	32075	0.028	$1.1{\cdot}10^6$	
POD-R improvement		3.85×	*Oslash;*	*Oslash;*	
AlexNet	DRAM[5]	84.8	2	42.4	
Ternary[5]	POD-R	490	0.93	526	
POD-R improvement		5.78×	1.94×	12.4×	
LeNet-5	RM[11]	59	0.017	13291	
Integer	POD-R	163	0.006	44169	
POD-R improvement		2.76×	2.33×	3.32×	
AlexNet	RM[11]	32.1	5.89	5.45	
Integer	POD-R	90.5	4.99	18.13	
POD-R improvement		2.81×	1.18×	3.33×	
Training improvement compared to FPGA					
Benchmark	Target	Throughput	Power	Efficiency	
		GFLOPS	W	GFLOPS/W	
LeNet-10	FPGA[17]	86.12	14.23	6.05	
		POD-R	101.5	2.76	36.77
POD-R improvement		1.18×	5.16×	6.08×	
AlexNet	FPGA[18]	34.52	7.74	4.46	
		POD-R	50.72	5.65	8.97
POD-R improvement		1.47×	1.36×	2.01×	
VGG-16	FPGA[18]	46.99	7.71	6.09	
		POD-R	81.95	5.7	14.37
POD-R improvement		1.74×	1.35×	2.36×	

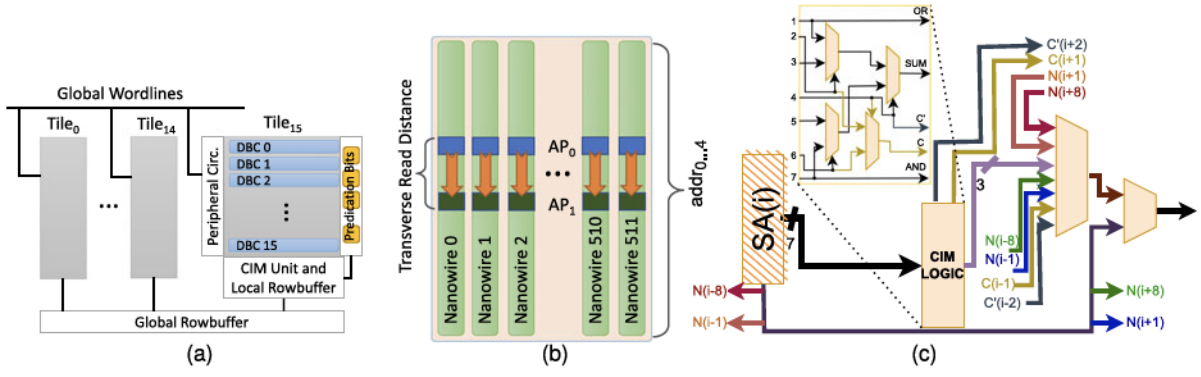

FIGURE 2. RM architecture following rank, bank, subarray, and tile conventions. All tiles are decomposed into DBCs with some tiles (e.g., one per subarray) augmented for transverse access and CIM. (a) Subarray built from tiles and DBCs. (b) DBC design: 2 APs for TR. (c) CIM unit for logic, arithmetic, and shifting.

target domain to *align* it with an AP (dark blue) and apply a current to *read* or *write* the target bit. To avoid data loss when shifting, overhead domains, represented in grey, are required.

RM, like many other novel memories, including resistive memory CIM crossbars,[10] has also received significant attention for CIM, particularly for deep learning.[2,11,12] In the next section, we describe the POD-RACING RM CIM approach that can operate at multiple levels of precision from binary/ternary weight inference to full floating-point precision online training.

POD-RACING

The memory architecture concept behind POD-RACING is shown in Figure 2. We follow a DRAM-inspired hierarchical organization consisting of ranks and banks constructed from subarrays built with tiles [see Figure 2(a)]. Each tile is constructed from bundles of RM nanowires shifted together and referred to as a domain-block cluster (DBC).[4,13] A DBC can accommodate D rows with parallel access to all bits belonging to the same row through the parallel APs [see Figure 2 (b)]. $D \in \{16, 32, 64\}$ is the number of data domains per nanowire. Each tile maintains a 512×512 shape, akin to DRAM. To enable CIM, a tile may be extended in two ways. A second AP is added to the DBCs in that tile to allow a current to traverse all the domains between the two APs indicated by the orange arrow [see Figure 2(b)]. If spaced within a prescribed *transverse read distance* (TRD), transverse read (TR) can distinguish between resistance levels based on the number of "1"s (ν) between the APs much like a multi-level cell.[14] Using TR, the local row buffer is extended with a *CIM unit* that retains a fast (bypass) path for a standard read, but can also convert the "1"s count from a TR into multioperand logic and the building blocks for arithmetic [shown for $\mathrm{TRD} = 7$ in Figure 2 (c)].

Multioperand AND and OR are naturally determined by sensing the highest (all ones) or lowest (all zeros) resistance levels. Operations of fewer operands can be accomplished by padding with ones or zeros as appropriate. Unlike prior processing using memory approaches POD-RACING includes logic to directly compute XOR from the 1's count, which also serves as the sum S for addition. All of these *bulk-bitwise* operations may be computed in parallel across the entire memory row. To support arithmetic we also can compute a carry $C = \frac{\nu}{2} \bmod 2$ and *super carry* $C' = \frac{\nu}{4}$ with minimal additional logic, which are needed to generate a sum of up to seven "1"s. Addition of TRD-2 operands may be computed directly by activating each nanowire in sequence. For nanowire N_i, S_i is written to AP_0 of N_i ($S_i \rightarrow N_{i,0}$), while similarly $C_i \rightarrow N_{i+1,1}$ and $C'_i \rightarrow N_{i+2,0}$ in parallel enabling a carry chain using navy and yellow connections in the CIM.[12]

The CIM block allows *logical* left and right shifts by both ones (orange and blue) and eight (red and green) positions. These *logical* shifts are different from RM nanowire shifting, which aligns different domains with APs (up and down). The shift by one position along with a small number of predication bits[15] to support multiplication using partial product addition. For more

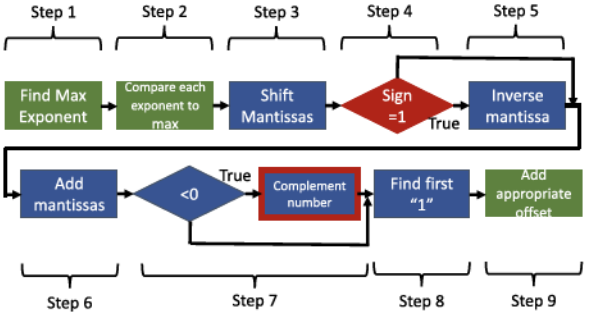

FIGURE 4. FP addition flow. Steps in blue, green, and red are operating on the mantissa, exponent, and sign bits, respectively.

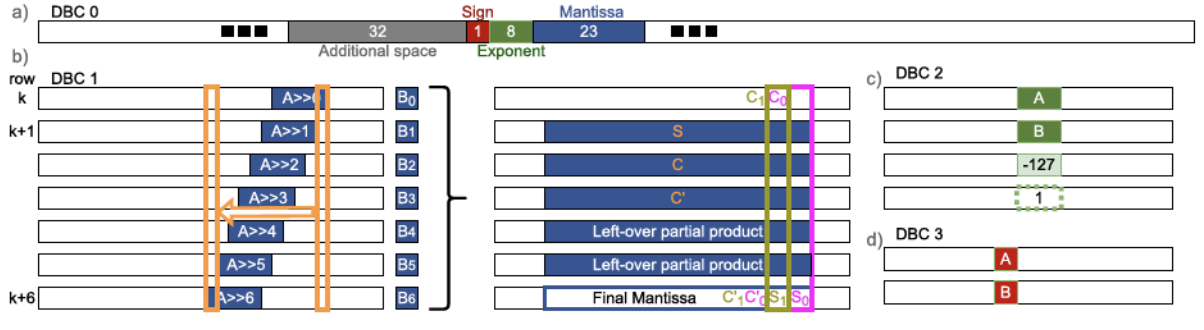

FIGURE 3. FP multiplication in memory.

than TRD-2 operands, computing the S, C, and C' bits in parallel reduces seven operands to three, allowing multiplication and reduction over addition to being computed in $O(n)$ time where n is the operand width. To clarify these capabilities from prior work,[12] detailed algorithms for these arithmetic operations are presented in Algorithm 1 of the supplementary material, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/MM.2022.3195761.

This CIM approach forms the basis to conduct binary, ternary, and integer/fixed point CNN inference. In the next sections, we describe floating-point multiplication and reduction over addition required for accurate backpropagation used for CNN online training.

FLOATING-POINT TWO-OPERANDS MULTIPLICATION

POD RACING implements 32-bit floating-point multiplication on operands composed of packed 23-bit mantissas represented in blue, 8-bit exponents (biased by $2^8 - 1$) in green, and a sign bit in red, in rows with 64-bit alignment, as shown in Figure 3(a). Data from multiple input channels are packed in 512-bit rows. Each operation executes on eight 64-bit values allowing channelwise parallelism regardless of window size or stride. Convolution across multiple windows in parallel is possible across different subarrays.[12]

First, the mantissa M_i, exponent E_i, and sign S_i of each operand $i \in \{A, B\}$ are masked off with an AND operation, the leading "1" restored with an OR operation and stored in separate DBCs. In Figure 3(b), using integer operations we multiply the two 24-bit mantissas by using operand B and the predication registers to store a shifted copy of A or "0"s depending on each bit of B. We sum the partial products in two major steps. First, we create for all bits in parallel a sum $\vec{S}$, carry $\vec{C}$, and supercarry $\vec{C'}$ vector, using the S, C, and C' operators, respectively, for each of seven partial products. We continue to reduce the generated $\vec{S}$, $\vec{C}$, and $\vec{C'}$ until we have ≤ 5 operands, as shown in the right half of Figure 3(b). Here the final sum is created by summing each bit position carrying C and C' where the final sum that is the mantissa product expanded to 48-bits. Because $1.0 \leq M_i < 2.0$ then it follows that their product P is $1.0 \leq P < 4.0$.

To normalize $P \geq 2.0$ uses the top bit t to govern a predicated normalization right shift by one [see Figure 2(c)]. Then as shown in Figure 3(c), we add the exponents $E = E_A + E_B + -127 + t$ using multioperand integer arithmetic where -127 counteracts the exponent offset and t is from P normalization. Finally, to determine the resulting sign of the multiplication, as shown in Figure 3(d), we execute $S = S_A \,\text{XOR}\, S_B$. A detailed algorithm is presented in Algorithm 2 of the supplementary material, available online. We leave M, E, and S, decomposed to facilitate reduction over floating-point addition described in the next section.

FLOATING-POINT MULTIOPERANDS ADDITION

FP addition requires nine steps, as shown in Figure 4. Step 1 determines the maximum exponent within the convolution window. The maximum exponent is determined by the searching for "1"s using TR at each position from most to least significant, if a "1" is present from any exponent in that bit position, all exponents without that bit set are set to zero eliminating all but the maximum value, this process is also used for determining the maximum value during pooling.[12]

In step 2 each local exponent is subtracted from the maximum. In step 3 this difference is used to govern right shifts to normalize the corresponding mantissas. This is implemented in POD-RACING by copying each difference bit from the lowest to highest into the predication register and executing a series of predicated logical right shifts (read and shift using the CIM unit). Each subsequent bit requires increasing the shift distance by powers of two for which we can leverage the orange N(i-1) or/and red N(i-8) connections in Figure 2(c) as appropriate.

In step 4, the sign bit of each operand is used as a predication value to invert the mantissa using an XOR operation with all "1"s. In step 5, the same predication register governs storing one. These two rows are now necessary for representing the signed representation M_i. In step 6, $M = \sum_i M_i$, where each M_i is represented as two rows, is conducted as discussed in the prior section for adding partial products. In step 7, if the most significant bit of M is "1," we complement the number, using this bit as a predicated inversion operation and then add one.

In Steps 8 and 9, we normalize the mantissa to 23 bits and adjust the exponent based on the normalization. There are three cases for M, the leading "1" is higher than bit position 48 requiring to increase the exponent and shift M right, it is lower than 48 but higher than 23 requiring decreasing the exponent and shift M right, or it is lower than 23 requiring decreasing the exponent and shifting M left. We accomplish this by creating a copy of M. Each cycle M is shifted left and predicated instructions are issued that are governed by seeing the first "1" to adjust the exponent

and after seeing the first "1" to shift the actual mantissa. Unfortunately, due to space limitations we were unable to provide precise details in the article, we describe the detailed algorithms for these operations in Algorithm 3 of the supplementary material, available online, which follows the basic flow and intuition laid out here.

ADDITIONAL OPERATIONS FOR BACKPROPAGATION

During backpropagation weight matrices must be rotated 180°, which is equivalent to swapping the values of these relatively small (3×3 up to 11×11) along the vertical and horizontal bisecting lines of the matrix. We use POD-RACING PIM to mask off the individual values of each row using AND, logically shift to the correct position, and recombine using OR. In addition, the weight update operation: $W' = W - L_R \times \Delta W$ where the new weight W' is a function of the previous weight W the learning weight L_R and the weight difference ΔW calculated via gradient descent method. We also use floating-point POD-RACING CIM to compute this function.

RESULTS

POD-RACING enables multiple precision modes from binary weight used for inference to floating-point required for effective training. FPGAs can also use multiple precision modes, however, it has been demonstrated by previous work that DRAM CNN-inference is faster and consumes less energy than FPGA CNN-inference.[16] Thus, we compare POD-RACING for inference against the state-of-the-art DRAM CIM using ternary weights[5,6] and RM using integer weights,[11] where the RM and DRAM (DDR3-1600) have a 1 and 1.6-GHz clock frequency, respectively. We compare POD-RACING for training using floating-point operations against energy-efficient FPGAs suitable for edge systems: Xilinx ZU19EG (LeNet-10)[17] and ZCU102 (AlexNet and VGG-16).[18] We used CIFAR-10 for LeNet-10 and ImageNet for AlexNet and VGG16 as in prior work.[18] The energy and latency parameters of accessing RM and TR in POD-RACING are provided by Yu et al.[2] and Roxy et al.[14] The latency and energy consumption for the CIM unit architecture extensions in Figure 2 were determined by implementing the design with the Cadence ASIC Flow targeting 45-nm technology. POD-RACING simplifies the CIM-unit but expands the shifting capability over prior work on integer CIM using RM requiring an area overhead of 10% creating one CIM-enabled tile per subarray.[12]

Like prior work, we presume the memory controller issues CIM operations from dedicated CIM instructions, which can be included directly in the software code as compiler directives. These instructions indicate the source, destination, the operation, and block size.[12,19] Either the virtual memory management system can be made aware of these instructions or they can be assigned to previously declared regions of CIM memory like in memory mapped I/O. In many cases, such as addition operations the memory controller must issues several instructions in sequence, governed by the block size, such as addition with the carry chain, finding the maximum, and predicated operations. Algorithms 1–3 in the supplementary document, available online, the detailed behavior of the memory controller operations required for these instructions.

CNN Inference

During the CNN inference phase, precision can be tuned based on the required accuracy. Reduced precision can provide a lower latency result *in situ*, which is particularly valuable for edge networks with small batch sizes. For instance, integer, ternary, or binary weight calculation reduces the complexity of addition and multiplication to simpler integer functional units while providing sufficient accuracy compared to more expensive floating-point computation. In fact, ternary and binary forms convert multiplication to *much* simpler bulk-bitwise (e.g., XOR) operations.

Using bulk-bitwise ternary weight CNN inference POD-RACING is more than 3× faster than the state-of-the-art DRAM CIM[5,6] with an approximately 2× power advantage leading to an order of magnitude efficiency advantage for AlexNet[a]. In fact, ternary weight CNN inference with POD-RACING is 2–3× faster than even simpler binary weight CNN inference using DRAM CIM.[12] Using integer operations, POD-RACING can outperform by nearly 3× and provides more than 3× the efficiency of the latest RM CIM.[11] The results are detailed in Table 1.

CNN Training

Given *in situ* training for low latency with small batch sizes and to maintain SWaP of edge systems GPUs may not be practical for their relatively high power. Sending these large datasets to the cloud for GPU acceleration is also impractical. Given CIM has yet to demonstrate CNN training with floating-point

[a]Power and energy data were not reported for the LeNet-5 DRAM CIM implementation[6] and is noted as a "–" in Table 1.

precision, we compare with FPGAs accelerators, which are emerging for *in situ* edge CNN training.[17,18]

POD-RACING is competitive, even outperforming FPGAs by 18%–74% with a significant improvement in power. We demonstrate a more than 2× improvement in efficiency even as the complexity of the CNN increases; POD-RACING for AlexNet is 2× more efficient, while VGG-16 is 2.36× more efficient. Thus, not only is POD-RACING demonstrating that CNN training is possible using CIM, it may even be more practical than FPGAs. When coupled with the high capacity and low-energy consumption of RM-based memory, the capabilities for SWaP constrained edge acceleration of deep learning and beyond are impressive and worthy of further exploration.

CONCLUSION

POD-RACING is the first, to the best of our knowledge, approach to enable full CNN architectures in memory, with multiple precision capabilities suitable for tuning both inference and training operations. While floating-point operations have always been a major roadblock for in-memory processing, POD-RACING can perform these operations efficiently at a speed and energy consumption improves over FPGA technology. In particular, POD-RACING is between 18% and 74% faster in terms of throughput, and at least 26% better in terms of energy, resulting in an efficiency improvement of more than 2× compared to the state-of-the-art FPGAs for small to moderate-sized CNNs. POD-RACING is the first CIM architecture that is sufficiently reconfigurable to provide capabilities and improvements over the state-of-the-art techniques for both *in situ* CNN inference and training for edge computing.

ACKNOWLEDGMENTS

This work was supported in part by the http://dx.doi.org/10.13039/100000001NSF under Grant CNS-1822085, Grant CNS-2133267, the http://dx.doi.org/10.13039/100009226National Security Agency , and Laboratory of Physical Sciences. The authors would like to thank Dr. Xulong Tang and Sheng Li for their consultation on this manuscript.

REFERENCES

1. S. S. P. Parkin, S. S. Parkin, M. Hayashi, and L. Thomas, "Magnetic domain-wall Racetrack memory," *Science*, vol. 320, no. 5873, pp. 190–194, 2008.
2. H. Yu *et al.*, "Energy efficient in-memory machine learning for data intensive image-processing by non-volatile domain-wall memory," in *Proc. 19th Asia, South Pacific Des. Automat. Conf.*, 2014, pp. 191–196.
3. D. Wang *et al.*, "Shift-optimized energy-efficient Racetrack-based main memory," *J. Circuits, Syst. Comput.*, vol. 27, no. 05, 2018, Art. no. 1850081.
4. A. A. Khan *et al.*, "Shiftsreduce: Minimizing shifts in Racetrack memory 4.0," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, pp. 1–23, 2019.
5. Q. Deng *et al.*, "DRACC: A dram based accelerator for accurate CNN inference," in *Proc. 55th Annu. Des. Automat. Conf.*, 2018, pp. 1–6.
6. X. Xin, Y. Zhang, and J. Yang, "Elp2im: Efficient and low power bitwise operation processing in dram," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2020, pp. 303–314.
7. P.-T. De Boer *et al.*, "A tutorial on the cross-entropy method," *Ann. Operations Res.*, vol. 134, no. 1, pp. 19–67, 2005.
8. J. Sim, H. Seol, and L. Kim, "NID: Processing binary convolutional neural network in commodity dram," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2018, pp. 1–8.
9. Y. Zhang *et al.*, "Perpendicular-magnetic-anisotropy CoFeB Racetrack memory," *J. Appl. Phys.*, vol. 111, no. 9, 2012, Art. no. 093925.
10. P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," *ACM SIGARCH Comput. Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.
11. B. Liu *et al.*, "An efficient Racetrack memory-based processing-in-memory architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl., IEEE Int. Conf. Ubiquitous Comput. Commun.*, 2017, pp. 383–390.
12. S. Ollivier, S. Longofono, P. Dutta, J. Hu, S. Bhanja, and A. K. Jones, "CORUSCANT: Fast efficient processing-in-Racetrack memories," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2022. Q3
13. R. Venkatesan *et al.*, "Tapecache: A high density, energy efficient cache based on domain wall memory," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Des.*, 2012, pp. 185–190.
14. K. Roxy, S. Ollivier, A. Hoque, S. Longofono, A. K. Jones, and S. Bhanja, "A novel transverse read technique for domain-wall 'Racetrack" memories," *IEEE Trans. Nanotechnol.*, vol. 19, pp. 648–652, 2020.
15. M. Lenjani *et al.*, "Fulcrum: A simplified control and access mechanism toward flexible and practical in-situ accelerators," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2020, pp. 556–569.

16. L. Jiang, M. Kim, W. Wen, and D. Wang, "XNOR-POP: A processing-in-memory architecture for binary convolutional neural networks in wide-Io2 drams," in *IEEE/ACM Int. Symp. Low Power Electron. Des.*, 2017, pp. 1–6.
17. Z. Liu, Y. Dou, J. Jiang, Q. Wang, and P. Chow, "An FPGA-based processor for training convolutional neural networks," in *Proc. Int. Conf. Field Programmable Technol.*, 2017, pp. 207–210.
18. Y. Tang *et al.*, "EF-Train: Enable efficient on-device CNN training on FPGA through data reshaping for online adaptation or personalization," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 27, no. 5, 2022, Art. no. 49.
19. V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2017, pp. 273–287.

SÉBASTIEN OLLIVIER research interests include novel memory reliability, computing units for processing in memory and application acceleration. He has authored or coauthored several articles focusing on domain-wall memory. Ollivier received his Ph.D. degree in electrical and computer engineering from the University of Pittsburgh, Pittsburgh, PA, USA. Contact him at sbo15@pitt.edu.

XINYI ZHANG research interests include software-hardware code-design for machine learning algorithms, deep-learning algorithm compression, and FPGA high-level synthesis. Zhang received his Ph.D. degree in electrical and computer engineering from the University of Pittsburgh, Pittsburgh, PA, USA. Contact him at xinyizhang@pitt.edu.

YUE TANG is currently a Ph.D. candidate with the University of Pittsburgh, Pittsburgh, PA, 15260, USA, in electrical and computer engineering. Her research interests include FPGA-based CNN training and on-device artificial intelligence. Tang received her M.S. degree from the School of Automation Science and Electrical Engineering, Beihang University, Beijing, China. Contact her at yut51@pitt.edu.

CHAYANIKA CHOUDHURI is a research volunteer with the University of Pittsburgh, Pittsburgh, PA, 15260, USA, in electrical and computer engineering under the supervision of Professor Alex K. Jones. Her research interests focusses on novel memories as domain-wall memory. Contact her at roc74@pitt.edu.

JINGTONG HU is currently an Associate Professor and William Kepler Whiteford Facultyfellow in the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA, 15260, USA. His current research interests include hardware/software co-design for machine learning algorithms, on-device AI, embedded systems.. Hu received his Ph.D. degree in computer science from the University of Texas at Dallas, Richardson, TX, USA. He is a senior member of IEEE. Contact him at jthu@pitt.edu.

ALEX K. JONES is a professor of ECE and CS with the University of Pittsburgh, Pittsburgh, PA, 15260, USA. He is currently serving as a program director with the U.S. NSF in the CNS Division of the CISE Directorate. His research interests include compilation for configurable systems and architectures, scaled and emerging memory, reliability, fault tolerance, and sustainable computing. He has authored or coauthored more than 200 publications in these areas. His research is funded by the NSF, DARPA, NSA, and industry. Jones received his Ph.D. degree in ECE from Northwestern University, Evanston, IL, USA. He is a senior member of IEEE and ACM. Contact him at akjones@pitt.edu.