Digital-based Processing In-Memory for High Precision Deep Neural Network Acceleration

Mohsen Imani, Saransh Gupta, Yeseong Kim, Minxuan Zhou, and Tajana Rosing
Department of Computer Science and Engineering, UC San Diego, La Jolla, CA 92093, USA

1. INTRODUCTION

Artificial neural networks, in particular deep learning, have wide range of applications in diverse areas including: object detection, self driving car, and translation. Recently, in some specific tasks such as AlphaGo and ImageNet Recognition [1], deep learning algorithms presented human-level performance. Convolutional neural networks (CNN) are the most commonly used deep learning models. Processing CNNs in conventional von Neumann architectures is inefficient as these architectures have separate memory and computing units. The on-chip caches do not have enough capacity to store all data for large size CNNs with hundreds of layers and millions of weights. This consequently creates a large amount of data movement between the processing cores and memory units which significantly slows down the computation.

Processing in-memory (PIM) is a promising solution to address the data movement issue [2, 3, 4, 5]. ISAAC [2] and PRIME [4] exploit analog characteristics of non-volatile memory to support matrix multiplication in memory. These architectures transfer the digital input data into an analog domain and pass the analog signal through a crossbar ReRAM to compute matrix multiplication. The matrix values are stored as multi-bit memristors in a crossbar memory. Although these PIM-based designs presented superior efficiency, there are several limitations when using PIM for CNN training. First, the precision of the design is bounded to fixed-point precision as determined by the number of multi-bit memristors used to represent a value. However, CNN models often need to be trained with floating point precision to achieve high classification accuracy [6, 7]. For example, GoogleNet, trained with 32-bit fixed point values, achieves 3% lower classification accuracy than the one trained with 32-bit floating points. In addition, earlier work showed that, without enough precision, the model training is likely to diverge or provide low accuracy [6, 8, 7]. Most commercial CNN accelerators train their models using floating point precision, e.g., bfloat16 [9]. The bfloat16 is a half precision floating point format utilized in Intel AI processors, such as Nervana NNP-L1000, Xeon processors, and Intel FPGAs, Google Cloud TPUs, and TensorFlow.

Another limitation is that the state-of-the-art PIM-based designs utilize costly digital-to-analog (DAC) and analog-to-digital converter (ADC) blocks. In addition, the mixed-signal ADC/DAC blocks take the majority of the chip area and power, e.g., 98% of the total area and 89% of the total power, and do not scale as fast as the CMOS technology does [2]. In addition, prior PIM designs use multi-bit memristor devices that are not sufficiently reliable for commercialization unlike commonly-used single-level NVMs, e.g., Intel 3D Xpoint. Their very expensive write operations frequently occur during the training. For example, work in [10, 11] extend the application of analog crossbar memory to accelerate training, but they still have expensive converter units and multi-bit devices. PipeLayer [12] modified the ISAAC [2] pipeline architecture and use spike-based input to eliminate ADC and DAC blocks. However, the computation of PipeLayer still happens on the converted data and its precision limits to fixed-point operations.

In this paper, we propose FloatPIM, a novel high precision PIM architecture, which significantly accelerates CNNs in both training and testing with the floating-point representation. This paper presents the following main contributions:

- **FloatPIM directly supports floating-point representations, thus enabling high precision CNN training and testing.** To the best of our knowledge, FloatPIM is the first PIM-based CNN training architecture that exploits analog properties of the memory without explicitly converting data into the analog domain. FloatPIM is flexible in that it works with floating-point as well as fixed-point precision.

- **FloatPIM implements the PIM operations directly on a digital data stored in memory using a scalable architecture.** All computations in FloatPIM are done with bitwise `NOR` operation on a single bit bipolar resistive devices [13, 14]. This eliminates the overhead of ADC and DAC blocks to transfer data between the analog and digital domain. It also completely eliminates the necessity of the multi-bit memristors, thus simplifying manufacturing.

- **We introduce several key design features that optimize the CNN computations in PIM designs.** FloatPIM breaks the computation into computing and data transfer phases. In the computing mode, all blocks are working in parallel to compute the matrix multiplication and convolution tasks. During the data transfer mode, FloatPIM enables a pipelined, row-parallel data transfer between neighboring memory blocks. This significantly reduces the cost of internal data movement.

- **We evaluate the efficiency of FloatPIM on popular large-scale networks with comparisons to the state-of-the-art solutions.** In this paper, we show how FloatPIM accelerates computations of AlexNet, VGGNet, GoogleNet, and SqueezeNet for ImageNet dataset [15]. In terms of accuracy, FloatPIM supporting floating point precision can achieve up to 5.1% higher classification accuracy than the one using fixed point representation. In terms of efficiency, our evaluation shows that FloatPIM in training can achieve 303.2× and 48.6× (4.3× and 15.8×) speedup and energy efficiency as compared to the state-of-the-art GPU (PipeLayer PIM accelerator [12]). In training, FloatPIM provides 324.8×

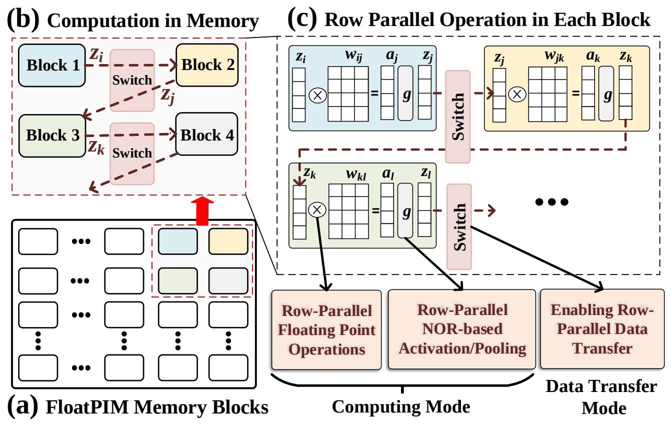
(b) Computation in Memory
(c) Row Parallel Operation in Each Block
Block 1
Block 2
Block 3
Block 4
Switch
Switch
z_i
z_j
z_k
w_{ij}
a_j
w_{jk}
a_k
w_{kl}
a_l
z_l
g
Switch
Row-Parallel Floating Point Operations
Row-Parallel NOR-based Activation/Pooling
Enabling Row-Parallel Data Transfer
Computing Mode
Data Transfer Mode
(a) FloatPIM Memory Blocks

Figure 1: Overview of FloatPIM.

and 297.9× (6.3× and 21.6×) speedup and energy efficiency as compared to GPU (ISAAC PIM accelerator [2]) respectively.

2. FloatPIM OVERVIEW

In this paper, we propose a digital and scalable processing in-memory architecture (FloatPIM), which accelerates CNNs in both training and testing phases with precise floating-point computations. Figure 1a shows the overview of the FloatPIM architecture consisting of multiple crossbar memory blocks. As an example, Figure 1b shows how three adjacent layers are mapped to the FloatPIM memory blocks to perform the feed-forward computation. Each memory block represents a layer, and stores the data used in either testing (i.e., weights) or training (i.e., weights, the output of each neuron before activation, and the derivative of the activation function (g')), as shown in Figure 1c. With the stored data, the FloatPIM performs with two phases: (i) computing phase and (ii) data transfer phase. During the computing phase, all memory blocks work in parallel, where each block processes an individual layer using PIM operations. Then, in the data transfer phase, the memory blocks transfer their outputs to the blocks corresponding to the next layers, i.e., to proceed either the feed-forward or back-propagation. The switches are shown in Figure 1b control the data transfer flows.

The block supports in-memory operations for key CNN computations, including vector-matrix multiplication, convolution, and pooling. We also support the activation functions like ReLU and Sigmoid in memory. MIN/MAX pooling operations are implemented using in-memory search operations. Our proposed design optimizes each of the basic operations to provide high performance. For example, for the convolution which requires shifting convolution kernels across different parts of an input matrix, we design shifter circuits that allow accessing weight vectors across different rows of the input matrix. The feed-forward step is performed entirely inside memory by executing the basic PIM operations. FloatPIM also performs all the computations of the back-propagation with the same key operations and hardware to the one used in the feed-forward.

FloatPIM further accelerates the feed-forward and back-propagation by fully utilizing the parallelism provided in the PIM architecture, e.g., row/block-parallel PIM operations. We show how these tasks can be parallelized for both feed-forward and back-propagation across a batch, i.e., multiple inputs at a time. It uses multiple data copies pre-stored in different blocks in memory.

For more information, please refer to our paper published in IEEE International Symposium on Computer Architecture (ISCA) 2019 [16]

3. REFERENCES

[1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[2] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, “Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” in *Proceedings of the 43rd International Symposium on Computer Architecture*, pp. 14–26, IEEE Press, 2016.

[3] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, “Rapidnn: In-memory deep neural network acceleration framework,” *arXiv preprint arXiv:1806.05794*, 2018.

[4] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, “Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory,” in *Proceedings of the 43rd International Symposium on Computer Architecture*, pp. 27–39, IEEE Press, 2016.

[5] M. Imani, D. Peroni, Y. Kim, A. Rahimi, and T. Rosing, “Efficient neural network acceleration on gpgpu using content addressable memory,” in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1026–1031, IEEE, 2017.

[6] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[7] M. Courbariaux, Y. Bengio, and J.-P. David, “Training deep neural networks with low precision multiplications,” *arXiv preprint arXiv:1412.7024*, 2014.

[8] M. Imani *et al.*, “Deep learning acceleration with neuron-to-memory transformation,” in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2020.

[9] “Bfloat16 floating point format..” https://en.wikipedia.org/wiki/Bfloat16$_floating-point_format$.

[10] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, and H. Yang, “Time: A training-in-memory architecture for memristor-based deep neural networks,” in *Proceedings of the 54th Annual Design Automation Conference 2017*, p. 26, ACM, 2017.

[11] Y. Cai, T. Tang, L. Xia, M. Cheng, Z. Zhu, Y. Wang, and H. Yang, “Training low bitwidth convolutional neural network on rram,” in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, pp. 117–122, IEEE Press, 2018.

[12] L. Song, X. Qian, H. Li, and Y. Chen, “Pipelayer: A pipelined reram-based accelerator for deep learning,” in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, IEEE, 2017.

[13] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, “MagicâĂŤmemristor-aided logic,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[14] A. Haj-Ali *et al.*, “Efficient algorithms for in-memory fixed point multiplication using magic,” in *IEEE ISCAS*, IEEE, 2018.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[16] M. Imani, S. Gupta, Y. Kim, and T. Rosing, “Floatpim: In-memory acceleration of deep neural network training with high precision,” in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 802–815, ACM, 2019.