XBAROPT - Enabling ultra-pipelined, novel STT MRAM based processing-in-memory DNN accelerator

Aqeel Anwar†, Arijit Raychowdhury†, Ryan Hatcher‡, Titash Rakshit‡
Georgia Institute of Technology, Atlanta, GA, USA†, Samsung Semiconductor Inc. ‡

***Abstract*—An explosion in big data driven machine learning (ML) applications in conjunction with a severe slowdown of Moore's Law are prompting the search for alternative application-specific hardware fabrics. With its focus on bringing the compute inside memory bitcells, processing-in-memory (PIM) has been proposed to accelerate ML inference applications. In this paper, we present a modular, end-to-end simulation framework that is required to find a power-performance optimized solution for PIM based architectures for a given application. Our simulation framework encompasses multiple levels of hierarchies including device bitcell, array, memory hierarchy, dataflow, data re-use and algorithm-to-system mapping. Novel concepts at two levels of the hierarchy are introduced and evaluated: 1. Logic embeddable, high Ion/Ioff Magnetic Tunnel Junction (MTJ) bitcell and 2. Cycle accurate inter and intra layer pipelined operation for high performance and low power operations. Results are compared to pure digital custom ASIC implementation showing orders of magnitude improvements in power-performance on widely accepted MLPerf benchmarks.**

I. INTRODUCTION

Deep neural networks (DNN) are currently being widely used in a myriad of real world ML applications. The high compute and memory demands of the DNNs make them hard to fit in power constrained edge devices. Severe slowdown of Moore's Law has exacerbated the burgeoning gap between the application demand and the hardware compute/memory supply. Memory-centric PIM has been proposed to accelerate machine learning applications for inference with its focus on bringing computing inside memory bitcells. This addresses the logic-to-memory bottleneck and the reduced technology improvements that currently plague general purpose compute like CPUs and GPUs when applied to ML. However, to analyze, benchmark and optimize a PIM based architecture, a full-stack end-to-end simulator and optimizer is needed that can encompass different levels of hierarchies. Optimizations and innovations at all levels of this end-to-end hierarchy are needed to produce a system that can provide very high performance within an acceptable power budget. In this paper we outline such a hierarchical and modular simulator that is used 1. to evaluate system impacts of two novel concepts at two different layers of hierarchy, a novel logic embeddable 2T2MTJ bitcell and an ultra-pipelined mapping scheme 2. to modularly analyze critical parameters across the stack for highest power-performance and finally 3. to compare the PIM system to a digital custom ASIC framework and quantify improvements in power-performance layer-by-layer on widely accepted MLPerf benchamrks.

II. HIGH ION/IOFF 2T2MTJ BITCELL AND PIM ARRAY

PIMs accelerate the most ubiquitous mathematical operation in machine learning applications, the matrix-vector multiplication. Many novel memories, e.g. resistive RAM (RRAM) and phase change memory (PCM) have been proposed as the bitcell solutions due to their non-volatile, analog nature. However, none of these memories have been shown to be embeddable in a logic manufacturing process (both in terms of the process as well as the operating voltages), which is critical to build a machine learning processor with its requirement of myriad high performance digital parts. Magnetic Tunnel Junction (MTJ) based Spin Transfer Torque (STT) Magnetic RAM (MRAM) bitcell is a logic embeddable non-volatile memory [1] that has hithertho been ignored for PIM applications due to its binary storage and low Ion/Ioff ratio. To overcome the digital nature of MTJs, bitsliced digital voltage signals are used to represent the input and output feature maps. Multiple input activations are fed and weighted by the bitcell conductances thereby performing MVMs simultaneously in parallel. Resultant currents on the bitline are accumulated as partial sums for an entire subarray in one clock cycle. To add the generated currents from multiple MVMs in parallel, the requirements for the bitcells are stringent. The bitcells require high Ion/Ioff ratio as well as very low variation in conductances. RRAM and PCM memories have modestly high Ion/Ioff ratios but very high conductance variability [2]. STT MRAM is embeddable in a logic process and exhibits low conductance variation but also very low Ion/Ioff ratios. To solve the problem of the low Ion/Ioff ratio, we propose a novel cross-coupled 2T2MTJ STT MRAM bitcell in which the Ioff is determined by the leakage of the transistor rather than the low conductance level of the MTJ, leading to on/off ratios of $> 10^4$ instead of < 3. The 2T-2MTJ bitcell is is shown in Fig 1. A logical 1(0) is implemented by setting the MTJ to high resistance state Rap(Rp) and $\overline{MTJ}$ to the low resistance state Rp(Rap). The cross-coupled 2T-2MTJ bitcell enables significant advantages when the inputs are a logical 1 due to current to BL being limited by the off state resistance of the transistor, which is typically several orders of magnitude larger than Rap as shown in Fig 2. Therefore, an array of 2T-2MTJ bitcells generates an output current on the BL that is significantly smaller than the equivalent 1T-1MTJ array enabling significant improvements in area, power and performance as compared to an 1T-1MTJ solution [3]. The write scheme for an array comprised of 2T-2MTJ bit cells is described in Fig. 2. The 2T-2MTJ structure is indeed 2x in area at the bitcell level. However, the total array size (bitcell + periphery) is reduced as compared to the 1T1M implementation. This counter-intuitive fact transpires due to a significantly smaller MUX in the periphery of the 2T-2MTJ

978-1-7281-4922-620$31.00

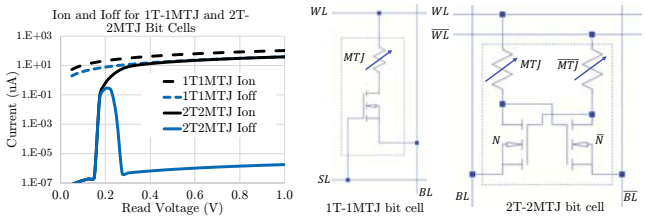

Fig. 1. (left) Transient I_{on} and I_{off} for 1T1MTJ and 2T2MTJ bitcells clearly shows the On/Off Ratio improvement as a function of V_{read}. (right) Bitcell diagram

			Selected Column		Other Columns	
Write Up	WL	$\overline{WL}$	BL	$\overline{BL}$	BL	$\overline{BL}$
MTJ	0	V_w	V_w	V_w	0	0
$\overline{MTJ}$	V_w	0	V_w	V_w	0	0

	Selected Row		Other Rows			
Write Down	WL	$\overline{WL}$	WL	$\overline{WL}$	BL	$\overline{BL}$
MTJ	V_w	V_w	0	0	0	V_w
$\overline{MTJ}$	V_w	V_w	0	0	V_w	0

Fig. 2. Write scheme for the 2T-2MTJ bit cell. The direction of the write (up or down) corresponds to whether the current is running from the word lines to the bit lines (down) or from the bit lines to the word lines (up). The write must take place in two separate steps – row-by-row and column-by-column

structure. The 3 to 4 orders of magnitude lower currents produced by the 2T-2MTJ structure Fig. 1 reduces the MUX dominated total area significantly Fig. 3. A more detailed version of the comparison w.r.t. 1T-1MTJ and other structures has been submitted for publication at ISCAS 2020 [3]. [4] describes a bitcell that is for TCAM arrays. However, the difference with our proposal is that the reference bitcell have the MTJ resistors connected to the drain of the transistors only. The key to our proposal is the cross-coupling at the gate which leads to the 3 to 4 orders of magnitude current reduction as the off state is governed by the transistor instead of the very high MTJ R_{off}. The peripheral circuitry especially the analog-to-digital (ADC) converter plays a big role in PIM based architectures. The ADC consumes much of the area and power and depends on the ADC bit precision which in turn affects the inference accuracy of the problem at hand eg CIFAR or ImageNet. Our modular optimizer can be used to choose the bit precision of ADCs to explore the PPA design space. As a baseline case, we use 6-bit ADC precision in conjunction with hardware aware retraining, that has been shown to be sufficient for most demanding ML problems [5]. The energy and area of the bitcell and peripherals are tabulated in Fig 3 based on 32nm node [6], [7] for consistent benchmarking wrt ScaleSim. Both negative and positive weights are encoded in our approach with multiple columns, typically 8, sharing one ADC.

III. XbarOpt Schematic and Mapping

The schematic block diagram of XbarOpt accelerator can be seen in Fig. 4. eMRAM is used to stored the layer outputs before they can be fed into the next layer as inputs for processing. Input register is responsible of storing intermediate input activations to be fed into the Xbar arrays as inputs. Each xbar array unit (XA) has a DAC and Sample and Hold unit in it. ADC is used the convert the analog outputs from the xbar arrays and the partial sums for each

Area (mm2)		Energy (pJ)	
array:	0.00010485	readDynamicEnergyArray:	14.1138
wlSwitchMatrix:	0.00265971	wlSwitchMatrix:	209.938
mux:	2.33E-03	mux:	28.9239
muxDecoder:	3.26E-05	muxDecoder:	44.7752
blSwitchMatrix:	2.50E-04	multilevelSenseAmp:	140.412
multilevelSenseAmp:	0.0035144	multilevelSAEncoder:	45.4243
multilevelSAEncoder:	0.0046242	Leakege	6.54E-05
Total	**0.014517**	**Total**	**438.587**

Fig. 3. Energy and area breakdown for bitcell array and peripherals

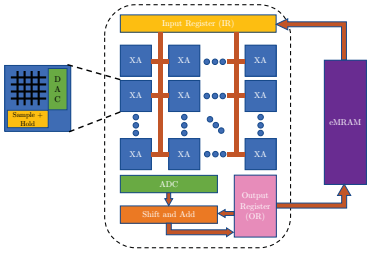

Fig. 4. XbarOpt Schematic Diagram

bit-sliced input activation is shifted and added and stored in the Output registers (OR). The idea behind XbarOpt is to allocate resources when it comes to Xbar arrays, hence it can be seen that xbar arrays are clustered together as opposed to tile-based division used in [8]. Ongoing work includes modelling these tile-based clusters into XbarOpt. A set of xbar arrays are assigned to each layer in the DNN topology. The number of rows is determined by the filter size of the layer, while the number of columns are determined by the number of filters of the layer, number of bits mapped per cell of the STT MRAM and the precision of the filter weights. The input activation are fed as rows to these xbar arrays and the output are collected from the bottom computing the partial sum, which is then shifted and added to the partial sum of the next input activation bit. The partial sum of each output value is stored in an Output Buffer. Once the final output is generated it is stored back in the eMRAM, which can then be used as inputs to the next layer. Based on how and when this output data is used for processing in the next layer, different pipelining schemes exist. In unpipelined (unpipe) scheme, the processing of the next layer isn't started until the entire output of the previous layer has been generated. The purpose of introducing this scheme is to have a fair comparison with a digitally implemented accelerator in the results section. In inter layer pipelined scheme (Pipe), the processing of the next layer begins as soon as we have enough output values generated from the previous layer [8]. This depends on the filter size and the stride length of the next layer. This scheme, however, is prone to delays when the stride length of the previous layer is greater than 1. This introduces delays in the pipeline, which trickles down to the last layer. This delay can be overcome by making use of intra-layer pipelining generating more than one output value per cycle (Pipe+ scheme). For example if the stride length of next layer is 2, the current layer needs to generate 2 outputs per time instant to overcome the pipeline delays caused by the lack of sufficient data. This means that we need to double the xbar arrays assigned to current layer, which results in and increased requirement of the xbar array (and hence the

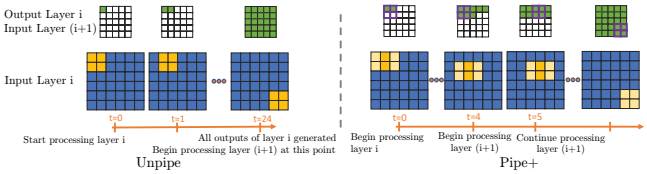

Fig. 5. Intra layer pipelining (pipe+)

clk	status	address				
0	miss	0	1	2	3	4
16	hit	12	13	14	15	16
32	hit	24	25	26	27	28
48	hit	36	37	38	39	40
64	hit	48	49	50	51	52
80	hit	60	61	62	63	64
96	hit	72	73	74	75	76
112	hit	84	85	86	87	88
128	hit	96	97	98	99	100
144	hit	108	109	110	111	112
160	hit	120	121	122	123	124

clk	status	num_writes	address		
2	busy	96	20000000	20000001	20000002
3	busy	96	20000000	20000001	20000002
4	busy	96	20000000	20000001	20000002
5	busy	96	20000000	20000001	20000002
6	busy	96	20000000	20000001	20000002
7	busy	96	20000000	20000001	20000002
8	busy	96	20000000	20000001	20000002
9	done	96	20000000	20000001	20000002
10	busy	96	20000096	20000097	20000098
11	busy	96	20000096	20000097	20000098
12	busy	96	20000096	20000097	20000098

Fig. 6. Example CSV trace file generated by XbarOpt

requirement of a larger compute unit area). This increase in the number of xbar arrays depends on the network topology and generally is only a fraction of the total number of xbar arrays assigned to the network. Fig 5 visualizes these three pipelining schemes.

IV. Results

A python based simulator of the proposed system architecture is designed and used in conjunction with hardware generated numbers. Prior work in the xbar arrays [9]–[13] lack a complete system architecture design and characterization of CNNs. The idea behind this simulation is to provide a full-stack understanding of the 2T2MTJ crossbar accelerator by exploring the design space for the required performance metrics. This simulation takes in the network topology, crossbar configuration file and a mapping file, while outputting layer-wise and system level performance metrics. The simulation goes through different phases and in each phase generates one or more comma-separated (CSV) files. Sample CSV trace files generated by XbarOpt can be seen in Fig. 6. The design space includes various modelling variables such as crossbar array dimensions, 2T2MTJ modelling parameters (such as bits-per-cell, area etc), memory sizes, access bandwidths, selection from the various available logical-to-physical array mappings, activation precision bits, digital-to-analog (DAC) resolution of the input, analog-to-digital (ADC) precision, inter-layer-pipelining etc.

A. XbarOpt vs Digitally-implemented Accelerator

We compare the performance of XbarOpt with a digitally-implemented accelerator simulator. A systolic array accelerator based on ScaleSim [14] was used to draw the comparison which is more optimized than CPU and GPU for these MLPerf tasks. In order to have a fair comparison, the dimension of the systolic array used for each of the network was determined individually such that the area occupied by the compute units for XbarOpt and ScaleSim was same. Four different neural networks (AlphaGoZero, AlexNet, ResNet50 and Neural Collaborative Filtering) were used as workloads to both these accelerator simulators. ScaleSim is compared to both unpipe and pipe version of XbarOpt. Fig. 7 summarizes the improvement results achieved by XbarOpt over ScaleSim for the following improvement performance metrics:

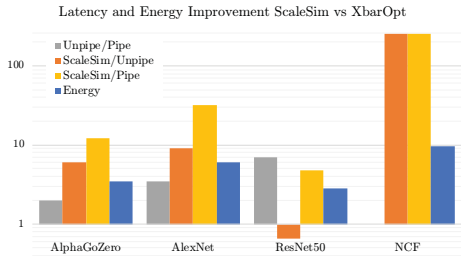

Fig. 7. Average improvement in using XbarOpt over ScaleSim for different performance metrics (log scale)

TABLE I
Performance parameters improvement for XbarOpt w.r.t ScaleSim

Network Name	Energy	Latency	Power	GOPS/Watt
AlphaGoZero	3.4231	12.249	0.346	2.88
AlexNet	6.1197	31.720	0.242	4.12
ResNet50	2.8628	4.8	1.379	0.72
NCF	9.6595	255.948	0.037	26.4

- Total energy per inference
- Latency between *ScaleSim* and $XbarOpt_{unpipe}$
- Latency between *ScaleSim* and $XbarOpt_{pipe}$
- Latency between $XbarOpt_{pipe}$ and $XbarOpt_{unpipe}$

The more the number of layers in a network, the greater the advantage from pipelining. Moreover, since there is no data re-use for fully connected layers as there is for convolutional layer, fully connected layers can't be pipelined in the manner discussed above. It can be seen from the figure that the latency improvement between $XbarOpt_{pipe}$ and $XbarOpt_{unpipe}$ is maximum for ResNet50, while it is minimum (=1) for NCF which consists of all fully connected layers and hence no room for pipelining improvement. The larger the percentage of fully connected layers (or more number of filters, in general) in the network, the better the resource allocation. Hence the latency improvement between *ScaleSim* and $XbarOpt_{unpipe}$ is largest for NCF, and smallest for ResNet50. The improvement between the latency of *ScaleSim* and $XbarOpt_{pipe}$ depends both on the number of layers in the network and ratio of fully connected layer weights to the total weights of the network. Hence it is largest for NCF. The energy improvement depends on various factors, the most significant one being the xbar array utilization. Using a larger xbar array for a smaller layer will result in unnecessary energy loss. AlexNet has the best xbar array utilization and hence the best energy improvement. Table I reports the improvement in per inference energy, time, consumed power and required GOPS per watt for the workloads compared to ScaleSim.

B. Exploring the design space

In this section we present the result of varying XbarOpt parameters and analyzing the effects on the AlexNet workload. These parameters include Xbar Array dimensions, eMRAM read/write bandwidth, pipeline techniques, ADC bit precision and 2T2MTJ vs ReRAM based xbar arrays

The results have been plotted in Fig 10. Increasing the Xbar array dimensions in general help in reducing the total inference energy for medium sized filters. For workloads where the filter size is small, using larger xbar arrays will yield in poor array utilization and unnecessary energy loss.

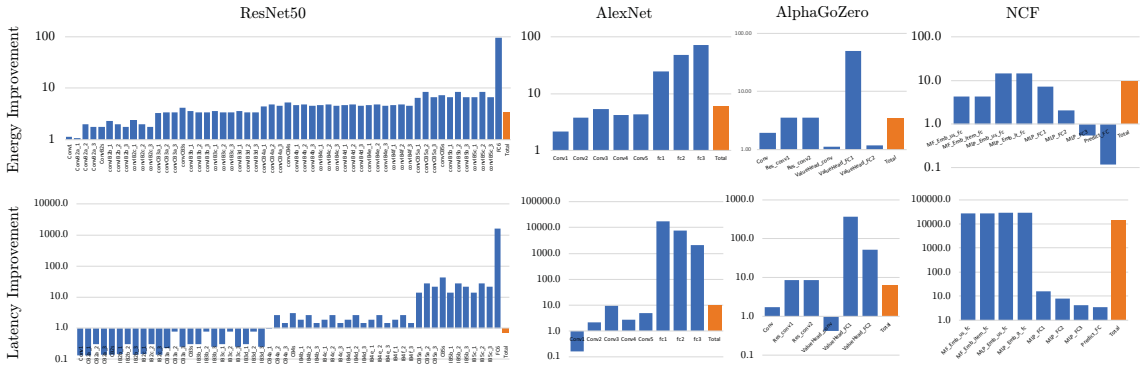

Fig. 8. Layer-wise energy and latency improvement across all four workloads (log scale)

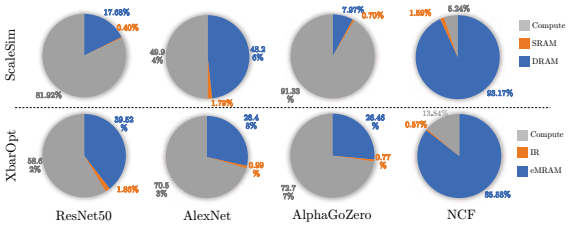

Fig. 9. Energy breakdown for different workloads on ScaleSim and XbarOpt

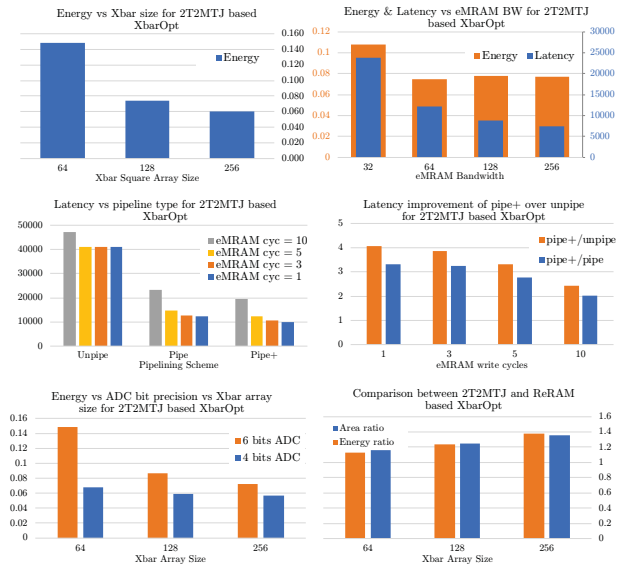

Fig. 10. Parameter sweep over AlexNet workload

Since AlexNet has comparatively larger filter size, increasing the xbar array dimensions yields in a lower total inference energy. eMRAM modelling has been taken into account in terms of its bandwidth and write cycles. We observed that increasing the eMRAM bandwidth overcomes the eMRAM access delays and results in a lower latency. The total energy per inference, however, depends on the per access energy of eMRAM for a given bandwidth. CACTI [15] based numbers were used to generate the per access read and write energies. It can be seen that the energy advantage by increasing the eMRAM BW tapers off after a certain point. eMRAM suffers with low write latency. XbarOpt models the write latency of eMRAM by dictating the number of cycles required to write. For lower write cycles, the increase in the total latency per inference is similar. The latency increases when the write cycles increases beyond a certain value. This value is determined by the input bit slicing and precision. In these results the input were decomposed into 8 slices of 1 bit each. Hence it takes 8 cycles for the xbar array to generate one partial sum. During that time eMRAM is not invoked and can consume cycles to write previous partial sum. As soon as the eMRAM write cycles increases beyond 8, the total inference latency starts increasing (Fig 10). The total inference latency for these three pipelining schemes w.r.t to the eMRAM write cycles can be seen in Fig 10. The figure also shows the latency improvement of pipe+ scheme over the unpipelined scheme as the eMRAM write cycles are varied. The pipe+ scheme yields about 2-4 times less latency as compared to other pipelining schemes with only an extra compute area overhead of 4%. With the increase of the eMRAM write cycles, the latency improvement decreases since pipe+ is already tightly coupled and takes full advantage of parallelism, increasing the number of write cycles have a greater impact on the latency of pipe+ scheme than that of unpipe scheme. The last row of the figure plots the results of varying the ADC bit precision and compares the performance of the proposed 2T2MTJ Xbar design with an ReRAM based design. Increasing the ADC bit precision has a larger impact for smaller xbar arrays as compared to larger ones. For smaller xbar array sizes, the ADC dominates the total energy of the xbar array and hence a larger impact. Comparing 2T2MTJ based xbar arrays with that of ReRAM, 2T2MTJ based xbar array only yields in a 1.1 to 1.4 times inference energy as that of an ideal ReRAM based xbar array design, with an added advantage that it can be fabricated.

V. CONCLUSION

Optimizations and innovations at all levels of hierarchy are needed to produce a PIM based system that can provide very high performance within an acceptable power budget especially at the edge. We outline such a hierarchical and modular simulator that is used 1. to evaluate system impacts of two novel concepts at two different layers of hierarchy, a novel logic embeddable 2T2MTJ bitcell and an ultra-pipelined mapping scheme 2. to modularly analyze critical parameters across the stack for highest power-performance and finally 3. to compare the PIM system to a digital custom ASIC framework and quantify improvements in power-performance.

REFERENCES

[1] Y. K. Lee, Y. Song, J. Kim, S. Oh, B.-J. Bae, S. Lee, J. Lee, U. Pi, B. Seo, H. Jung *et al.*, "Embedded stt-mram in 28-nm fdsoi logic process for industrial mcu/iot application," in *2018 IEEE Symposium on VLSI Technology*. IEEE, 2018, pp. 181–182.

[2] Z. Jiang, Y. Wu, S. Yu, L. Yang, K. Song, Z. Karim, and H.-S. P. Wong, "A compact model for metal–oxide resistive random access memory with experiment verification," *IEEE Transactions on Electron Devices*, vol. 63, no. 5, pp. 1884–1892, 2016.

[3] Y. L. et al, "A variation robust dnn inference engine based on stt-mram with parallel read-out."

[4] H. Noguchi, K. Kushida, K. Ikegami, K. Abe, E. Kitagawa, S. Kashiwada, C. Kamata, A. Kawasumi, H. Hara, and S. Fujita, "A 250-mhz 256b-i/o 1-mb stt-mram with advanced perpendicular mtj based dual cell for nonvolatile magnetic caches to reduce active power of processors," in *2013 Symposium on VLSI Technology*. IEEE, 2013, pp. C108–C109.

[5] K. R. A. R. Shubham Jain, Abhronil Sengupta, "Rxnn: A framework for evaluating deep neural networks on resistive crossbars," *arXiv preprint arXiv:1809.00072*, 2019.

[6] X. Peng, R. Liu, and S. Yu, "Optimizing weight mapping and data flow for convolutional neural networks on rram based processing-in-memory architecture," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[7] "https://github.com/neurosim/dnn_neurosim_v1.0."

[8] 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," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[9] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, p. 61, 2015.

[10] Y. Kim, Y. Zhang, and P. Li, "A digital neuromorphic vlsi architecture with memristor crossbar synaptic array for machine learning," in *2012 IEEE International SOC Conference*. IEEE, 2012, pp. 328–333.

[11] X. Liu, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Y. Wang, H. Jiang, M. Barnell, Q. Wu *et al.*, "Reno: A high-efficient reconfigurable neuromorphic computing accelerator design," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[12] C. Yakopcic and T. M. Taha, "Energy efficient perceptron pattern recognition using segmented memristor crossbar arrays," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2013, pp. 1–8.

[13] T. M. Taha, R. Hasan, C. Yakopcic, and M. R. McLean, "Exploring the design space of specialized multicore neural processors," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2013, pp. 1–8.

[14] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator," *arXiv preprint arXiv:1811.02883*, 2018.

[15] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2007, pp. 3–14.