Merged Logic and Memory Fabrics for AI Workloads

Brian Crafton
ECE Department, Georgia Tech
brian.crafton@gatech.edu

Samuel Spetalnick
ECE Department, Georgia Tech
sspetalnick3@gatech.edu

Arijit Raychowdhury
ECE Department, Georgia Tech
arijit.raychowdhury@ece.gatech.edu

ABSTRACT

As we approach the end of the silicon roadmap, we observe a steady increase in both the research effort toward and quality of embedded non-volatile memories (eNVM). Integrated in a dense array, eNVM such as resistive random access memory (RRAM), spin transfer torque based random access memory, or phase change random access memory (PCRAM) can perform compute in-memory (CIM) using the physical properties of the device. The combination of eNVM and CIM seeks to minimize both data transport and leakage power while offering density up to 10× that of traditional 6T SRAM. Despite these exciting new properties, these devices introduce problems that were not faced by traditional CMOS and SRAM based designs. While some of these problems will be solved by further research and development, properties such as significant cell-to-cell variance and high write power will persist due to the physical limitations of the devices. As a result, circuit and system level designs must account for and mitigate the problems that arise. In this work we introduce these problems from the system level and propose solutions that improve performance while mitigating the impact of the non-ideal properties of eNVM. Using statistics from the application and known properties of the eNVM, we can configure a CIM accelerator to minimize error from cell-to-cell variance and maximize throughput while minimizing write energy.

ACM Reference Format:
Brian Crafton, Samuel Spetalnick, and Arijit Raychowdhury. 2021. Merged Logic and Memory Fabrics for AI Workloads. In *26th Asia and South Pacific Design Automation Conference (ASPDAC '21), January 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3394885.3431615

1 INTRODUCTION

Over the last decade, tremendous progress towards accelerating machine learning workloads has been made at all levels of the computing hierarchy, enabling orders of magnitude improvement in energy efficiency. At the software level, models are compressed, pruned, and quantized to minimize the total storage size and the energy cost of a single inference [9]. At the hardware level, prior work focuses on maximizing the reuse of all data such that expensive memory accesses and total data movement is minimized [3]. Both of these strategies focus on minimizing the cost of data movement and memory accesses, while maximizing the utility of available on-chip memory capacity. While these techniques yield strong results, they still face the fundamental technological limitations of CMOS. In particular, the large size of the SRAM bitcell ($\approx 150F^2$) results in limited on-die capacity, which necessitates movement of data from an external DRAM to the on-die SRAM at more energy per bit.

Fortunately, there has been an increasing research effort toward the design and fabrication of novel memory technologies that are logic process- and voltage-compatible, while providing non-volatility and high density with manageable read and write performance. These new devices have important new properties that have been long absent in traditional charge-based memory technologies. They are all embedded non-volatile memory (eNVM) solutions, meaning they can be completely powered down without loss of data, and hence consume virtually no leakage power. Furthermore, these technologies store information through change of resistance. This enables us to perform compute in-memory (CIM) on the bit-line (BL) with breakthrough improvements in throughput and energy-efficiency. These properties have the potential to realize the long awaited benefits of in-memory computing.

If successful, CIM with eNVM promises to solve many of the engineering challenges that the modern memory hierarchy faces with regards to memory bandwidth and density. In recent years, new proposed devices have reached huge milestones on their way to commercial viability. However, these emerging technologies present a few novel challenges which have so far precluded them from widespread commercial use. In this paper, we will examine some of these challenges and provide an overview of the recent developments from both the technology and circuits and systems perspectives. Specifically, the two main challenges we address in this paper are the high write energy of eNVM and cell-to-cell variation that impacts CIM.

All eNVMs feature significantly higher write energy than traditional CMOS memories [5]. Therefore, unlike traditional CMOS architectures, recent CIM architectures [15] do not re-program the arrays after initialization. This constraint requires several fundamental changes to data flow and placement of weights in the eNVM arrays that we discuss in detail in Section 4. The second key obstacle to mainstream CIM with all eNVMs is the inherent cell-to-cell variation in the device's resistive state. These variations are not specific to eNVM, and occur due to process and temperature or write-to-write (cycle-to-cycle) variations. When reading multiple memory cells at the same time with an analog-to-digital converter (ADC), high variation among resistive states results in sum-of-products errors accumulated on the BL. To overcome cell-to-cell variations, recent work has approached the problem from all levels of the computing hierarchy, ranging from circuits to algorithms. By solving these new challenges at various levels of the computing hierarchy, we can enable the use of new memories which will help break the long withstanding memory bottleneck and enable more efficient systems for machine learning.

ASPDAC '21, January 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431615

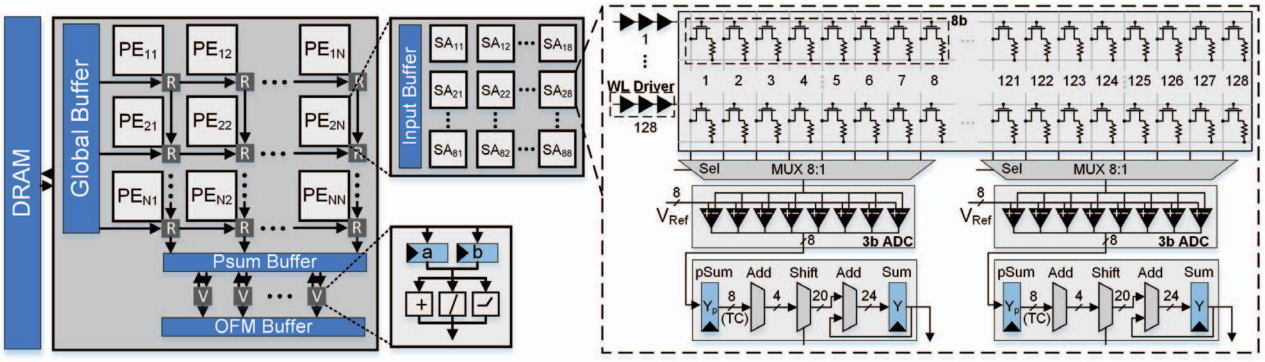

Figure 1: Prototypical compute in-memory hierarchy including processing engine (PE) and sub-array (SA) architecture.

2 OVERVIEW OF NON-VOLATILE COMPUTE IN-MEMORY DEVICES

Although most "emerging" non-volatile devices have been around for quite some time, recent advances in deep learning have heightened the importance of large on-chip memories. These devices offer a new solution to matrix multiplication by performing multiply and accumulate on the BL, reading multiple rows at the same time and using analog-to-digital converters (ADC) to read the output. While variations of traditional SRAM can perform this technique as well, eNVM enable virtually zero leakage power and much higher densities. In this section we overview three of the more popular and mature devices currently being explored today.

RRAM (often called ReRAM) [18] consists of filamentary devices that switch between a high resistance state (HRS) and low resistance state (LRS) based on the direction of current applied across the two terminals. The HRS and LRS in RRAM are achieved by forming and destroying a filament inside the insulator material of the device. By creating and destroying this filament we can lower and raise the resistance of the device by orders of magnitude. The transition from HRS to LRS is called the *set* process where the device allows more current to flow emulating a digital '1'. The transition from LRS to HRS is called the *reset* process where the device is less conductive and results in less current across the terminals.

A more mature technology for resistive memories is the *STT-MRAM* [2]. The *STT-MRAM* bitcell consists of one access transistor and one Magnetic Tunnel Junction (MTJ) where a single bit of information is stored. An MTJ is formed with two ferromagnetic CoFeB based layers and one insulating layer (MgO) in between. One ferromagnetic layer is called a fixed layer because its magnetic moment is fixed to one direction. The other ferromagnetic layer is called a free layer since the direction of magnetic moment can be changed based on the direction of current flowing across the MTJ.

PCRAM [19] devices enjoy some maturity and have a history of real-world use in optical storage media. PCRAM cells consist of a layer of glass chalcogenide phase-change material, typically GeSbTe (GST), sandwiched between a pair of electrodes. This layer can be fully or partially crystallized, or completely amorphous. In the amorphous phase, the conductivity of the phase-change material is much lower than that in the crystalline phase, allowing resistance to be controlled. Crystallization occurs rapidly when the material is heated, via Joule heating, to below the melting point. The amorphous phase may be formed by melting the cell then allowing it to rapidly cool; this is the so-called "quenching" procedure.

3 COMPUTE IN-MEMORY

CIM seeks to perform matrix multiplication ($\vec{y} = W\vec{x}$) in a crossbar structure in the analog domain using Ohm's law, exploiting the non-volatile conductance state(s) provided by the eNVM. Using this technique, each weight of the matrix (W_{ij}) is programmed as the conductance of a bit-cell and each value of the vector ($\vec{x_i}$) is converted to a corresponding voltage and applied to the rows of the memory crossbar. The current through each cell is proportional to the product of the programmed conductance (W_{ij}) and applied voltage ($\vec{x_i}$) (Ohm's Law). By Kirchhoff's current law (KCL), the resulting currents that are summed along the columns of the crossbar are proportional to the product of the matrix and vector, ($\vec{y}$).

Recent designs have encoded various numbers of bits in each cell ranging from 8-bit [4], down to 2-bit [15] or 1-bit [16]. Furthermore, these designs use various ADC configurations, such as SAR [15] and flash [21] ADCs using various precision. However, recent works have gravitated towards lower precision eNVM cells and Flash ADCs to manage accumulated variance from cells [6, 14, 21]. Consistent with state of the art CMOS accelerators [10] and low-precision neural networks [9] 8 (or less) bits are typically used for inference. To implement this with binary cells, 8 adjacent cells to form a single 8-bit weight, like those shown in the columns of Figure 1. The 8-bit vector inputs to this array are represented as voltages (GND and VDD), and are input one at a time over 8 cycles. Depending on the size of the ADC and assumed cell-to-cell variance, prior works read the entire column at once [15] or break it down into several cycles [21]. For example, if 3-bit ADCs are used then 128 rows can be read in 16 cycles. This can be done more optimally if zero skipping [21] is used. For an N-bit ADC, zero-skipping enables the first 2^N wordlines containing '1's, thus ensuring the ADC will not sustain any quantization error. Because the input data contains more '0's than '1's, we expect more than 2× speedup [6, 21]. In Figure 2, we provide an example case for zero-skipping where 8 total rows are read using a 2-bit ADC. The first technique, we call *baseline*, involves simply reading as many rows as the ADC precision allows (e.g. for a 3-bit ADC, we read 8 rows simultaneously). This (2B) requires 2 cycles since it targets four consecutive rows at a time. Zero-skipping (2A) is able to finish

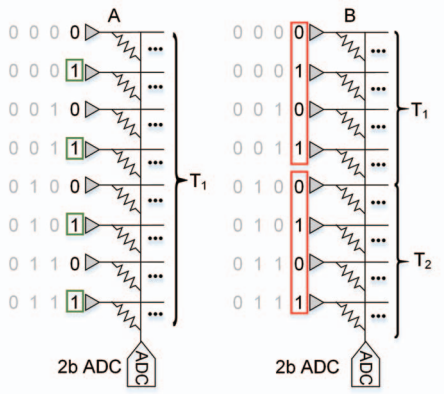

Figure 2: Performance advantages of zero skipping (A) over baseline (B) CIM technique.

all 8 rows in a single cycle because we only consider the '1's in the input vector. Zero skipping performs faster than the baseline technique because for most cases it will process more total rows per cycle.

The result of the binary dot product is the current passing through each cell, and the sum-of-products is accumulated along the bitline. The result is then collected at the ADCs. If there are more inputs than states that the ADC can distinguish, we must divide the binary dot product over multiple cycles and sum the partial products together with CMOS. For each input-bit and weight-bit binary product, we perform this binary dot product, and then multiply (shift) the result by the sum of the magnitudes of each bit. Thus in order to perform an 8b multiplication, we perform binary multiplication between each bit in the input data and each bit in the weight data, and then shift by combined magnitude. In this way, we can perform 8-bit matrix multiplication by performing 64 binary multiplications with shift and add operations.

4 ARCHITECTURE

Prior works encapsulate the array, ADCs, and shift and add logic to create a matrix multiplication engine. For example, if a 128×128 array is used, a 128×16 8b matrix multiplication can be implemented since 16 8b words can be fit across the 128 bitlines. Using these sub-arrays as building blocks, we can store larger matrices and perform distributed matrix multiplication where each sub-array operates as a partial matrix. Given that matrix multiplication makes up the majority of the workload in deep learning models, prior works have implemented Convolutional Neural Networks (CNN) [15] and Recurrent Neural Networks (RNN) [11].

Given the high density of these PEs, hundreds or thousands of them can be tiled in the same area used by modern ICs, enabling smaller footprints and greater memory capacity. Despite high density and zero leakage power, eNVM suffer from high write energy and high write latency. As a result, existing work [15] avoids writing the eNVM once programmed. While this is advantageous for data transport and energy efficiency, it means each CIM processing element (PE) can only perform operations it has the weights for. This implies that if there is an unbalanced workload where some PEs operations take longer than others, we cannot simply re-allocate these operations to other PEs. Therefore, we must use

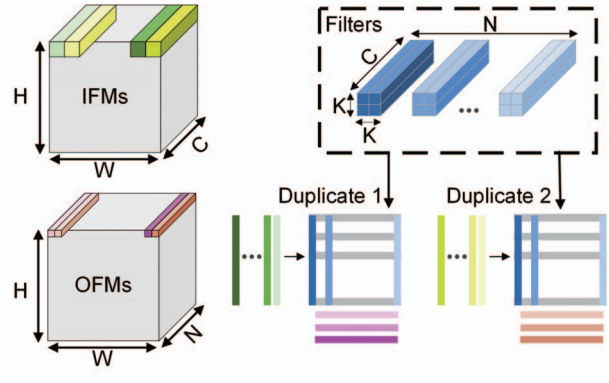

Figure 3: CNN layer using CIM with weight duplication.

synchronization barriers for all PEs so distributed matrix multiplication completes before another is started. In contrast, every CMOS and SRAM based PE are computationally identical and can perform any operation in the DNN graph.

Since we cannot simply reallocate operations to CIM PEs for an unbalanced workload, it is challenging to ensure that all PEs remain fully utilized. Due to this constraint, a fundamental problem in CIM based designs is array utilization, that is, the percent of time an array is in use. Recent large scale CIM designs [15], use two techniques called weight duplication and layer pipelining to maximize array utilization under new constraints. To generalize these techniques and optimize for any network, [6, 14] partition weights such that throughput is maximized.

4.1 Weight Duplication

Weight duplication [15] is used to maximize throughput in large scale CIM accelerators where the amount of on-chip memory exceeds the number of weights in the model. In [14], 24,960 arrays are used for a total on-chip memory capacity of nearly 104 MB (2b cells), while only using an area of $250mm^2$. Using this enormous on-chip memory capacity, they not only fit ResNet [8] but duplicate shallow layers up to 32×. When weights are duplicated, the input data is divided equally amongst each duplicate array so they can process in parallel. We illustrate this idea for a convolutional layer in Figure 3. The input patches from the input feature maps (IFMs) are divided into groups based on the number of duplicates, and then mapped to each duplicate.

In Figure 3 we further depict how these arrays can be pieced together to form a larger matrix. In this example, both input feature maps and filters are vectorized with the filters forming the columns of a matrix. The vectorized feature maps are input to the crossbar to perform matrix multiplication, where the results are output feature maps for this layer in a CNN.

4.2 Layer Pipelining

Layer pipelining [15] is used to maximize throughput in eNVM CIM accelerator, where arrays are not re-programmed due to large amounts of on-chip memory and high write energy. At the same time, most modern neural networks contain 20 or more layers that must be processed sequentially. Given that most designs use 128×128 arrays, it becomes infeasible to partition arrays such that

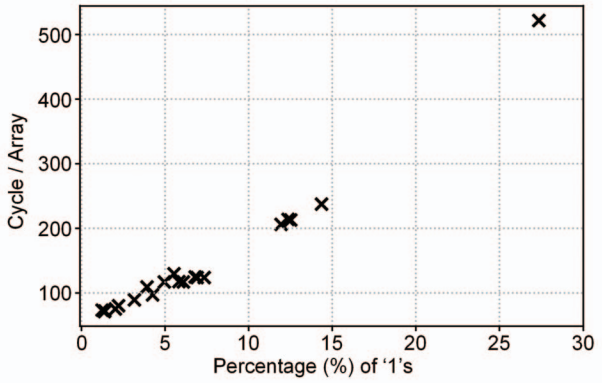

Figure 4: Average performance by layer (ResNet18) versus percentage of '1's in input data.

they can be used for each layer without being re-programmed. This implies that the majority of PEs would sit idle waiting for their layer to be processed. To solve this problem, images are pipelined through the network to keep all arrays utilized. Although this compromises single example latency, it maintains maximum throughput.

4.3 Array Allocation

CIM with weight duplication and layer pipelining offers an elegant solution to the memory wall observed by CMOS accelerators. Due to the massive number of fixed-function PEs available, how these PEs are duplicated and assigned to specific layers becomes an important challenge. Hence, the challenge CIM faces is not with weight transport, but rather with weight placement and allocation. To maximize throughput, weights must be distributed in a way that allows each CIM PE to be operating at all times to maximize throughput. One example of an optimal weight mapping and data flow was demonstrated in [14]. Using redundant weights and clever mapping strategies, they maximize throughput of a large scale CIM accelerator. However, this work assumes deterministic computation time for each array.

Circuit level techniques like zero-skipping greatly increase performance, but create non-deterministic workloads such that each array takes a different amount of time to complete its task. This is because the number of ones in the input vector of the CIM operation is randomly distributed, and thus the amount of time to finish a dot product is non-deterministic. In Figure 4, we plot the average time for an array to perform a 128×16 matrix multiplication versus the percentage of '1's in all the 8-bit input features for the 20 convolutional layers in ResNet18. Naturally, this information can be used to better allocate arrays in our design. Rather than allocating arrays just based on the total number of MACs per layer, we can allocate arrays based on both workload and performance.

5 OVERCOMING DEVICE VARIANCE

Upon moving multiply-and-accumulate operations to the analog domain, we observe numerous advantages, but face device variation challenges that digital logic avoids by design. These variations are not specific to eNVM, and occur due to process and temperature

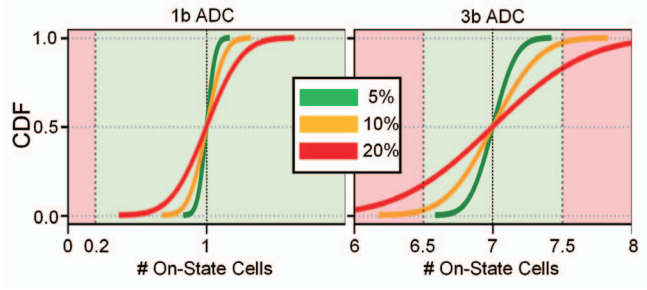

Figure 5: Impact of ADC precision and variance on CIM.

variations or write-to-write (cycle-to-cycle) variations. Conventional digital memory such as SRAM overcomes this challenge using differential sensing and a large ratio between the '0' and '1' states. However, when reading multiple memory cells at the same time with an analog-to-digital converter (ADC), high variance between resistive states results in sum-of-products errors accumulated on the bitline. Simultaneously, state-to-state separation is reduced as more states are placed into the same dynamic range. Given that these operations are used to implement matrix multiplication, and thus neural networks, we find that device level variance results in erroneous computation. While neural networks can tolerate these errors to some extent, accuracy degrades as a function of the error rate. To better understand the impact of variance on CIM, we plot the CDF of error for a 1-bit ADC and a 3-bit ADC in Figure 5. For the 1-bit ADC, the single reference voltage can be set such that even high variance in LRS states do not cause an error. In this case, the reference voltage is set to 0.2 on-state (LRS) cells and the entire CDF is inside the green region which denotes correct operation. However, for the 3-bit ADC the reference voltages cannot be set in such an optimal way because most output states are flanked on either side by other states. In Figure 5B, we observe how both 10% and 20% variance yield errors frequently.

To translate this to accuracy, we evaluate a trained VGG11 network on CIFAR10 in Figure 6. We sweep between 1% and 20% variance using 1-bit, 2-bit, and 3-bit ADCs. For each ADC, the maximum number of distinguishable wordlines are enabled. Hence, for the 1-bit ADC 1 wordline is enabled, for the 2-bit ADC 4 wordlines are enabled, and for the 3-bit ADC 8 wordlines are enabled. To emulate variance-induced errors, we add noise at the output of each matrix multiplication in the network. Like prior work [12], we find that the accuracy falls based on the number of wordlines enabled and the variance of the devices used. From this experiment we infer that devices with higher variance cannot be used for even 3-bit ADCs. Thus in order to increase the space of ADC designs that can be used, low device variation is required motivating the following subsections.

5.1 Write-Verify Protocols

Write-verify methods take an iterative approach to minimizing write error (*i.e.* state variance) using feedback. During each iteration, devices are programmed to the desired state and then read back to determine the error. After each iteration, the error from the desired state is used to determine the parameters defining next write. Over several iterations, the device's conductance converges to the desired

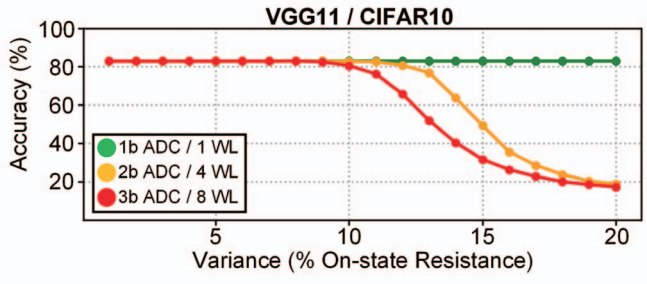

Figure 6: Impact of ADC precision and variance on CIFAR10 classification accuracy using VGG11.

value. Once the device's conductance is within an accepted error range, the write-verify protocol terminates. In this way, several iterations of feedback can be used to greatly reduce cell-to-cell variance.

Several demonstrations and various protocols for write-verify on eNVM have been proposed [1, 20, 22, 23]. In [22], RRAM cell-to-cell variation is minimized with a 6-step write-verify technique. They implement a binary neural network using two RRAM cells to represent '+1' and '-1' and 3-bit ADCs. For SET operations (LRS), 2.3V is applied to the transistor's gate and 2.1V is applied on the bitline. For RESET operations (HRS), 3.8V is applied to the gate of the transistor and 4V is applied on the bitline. In [20], PCRAM devices are programmed through write-verify and various write voltages to achieve 3.5% cell-to-cell variation. At this distribution, one can expect less than 3σ error when using 3-bit ADCs enabling more efficient CIM.

Recently [23], a write-verify scheme was implemented on a 64KB RRAM macro that was profiled over a broad range of programming voltages. In this work, the advantages of using higher write voltages and write-verify is demonstrated. In Figure 7, we show this data along with a die image and specifications. In Figure 7 we plot the cumulative distribution function (CDF) of device resistance for the various write voltage experiments. Over the 5 different write voltage experiments, significant reduction in cell-to-cell variation is observed. With a write voltage of 1.1V, the standard deviation in resistance across all devices (σ_R) is 1046. However, increasing the write voltage to 1.9V a σ_R of only 84 Ω is achieved. Measured as a percentage of the average LRS (μ), the cell-to-cell variation (σ/μ) for 1.1V is 18.5% and for 1.9V is only 3.5%. Although a higher write voltage and more iterations yields a higher resistance ratio and tighter HRS/LRS distribution, it greatly reduces the endurance in these devices [13]. Therefore, extensive characterization should be done [23] to determine the best trade-off between endurance and precision.

5.2 Variance-Aware Training Methods

Variance-aware training can be categorized into two types: offline and online. Offline training methods take existing DNN models and attempt to make them robust to noise by retraining the networks with noisy input data and computation [11]. This can be done by acquiring LRS and HRS distributions and running simulations to understand the impact on the DNN workload, then training the DNN models during simulation or just by simply injecting noise

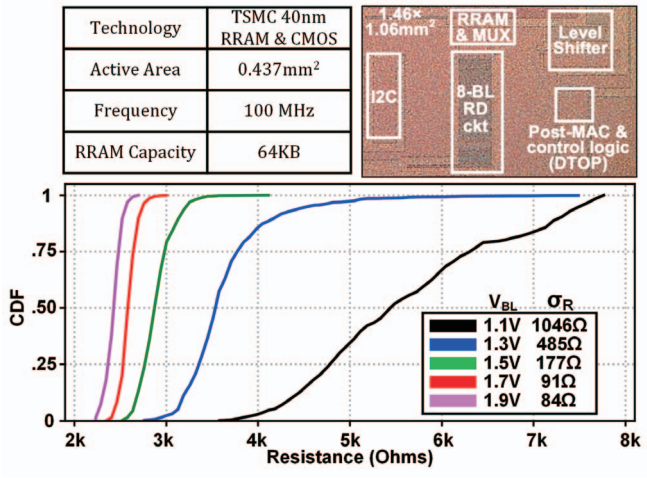

Figure 7: Write variation on commercial RRAM process

to emulate cell-to-cell variation after each layer. Like approximate computing, this technique relies on the idea that neural networks do not need exact computation for high accuracy, and that high accuracy can be obtained with erroneous computation if networks are trained to do so.

Offline training methods use estimated distributions of cell-to-cell variation rather than the actual variations present on a specific chip. Online training exploits more exact distributions, by retraining DNN models for a given chip. Each chip will have its own unique variations and faults, that will be similar, but still different than the distributions used for offline training. Hence, it is advantageous to train each chip based on its own variations and faults such that performance is optimal for the given chip [17]. Naturally, higher accuracy can be obtained, but at the cost of requiring each chip to be retrained after fabrication. This requires significant time per chip during the test phase and also requires that the design has circuits to support on-chip re-training.

5.3 Statistical Readout Methods

Statistical readout methods like counting cards [7] control variance induced errors during inference time. Given that CIM-based vector-matrix multiplication (VMM) is implemented by a series of ADC reads followed by shift and add operations, the expected error for a VMM can be computed. Starting with an ADC read, error can be formulated as a function of the number of wordlines enabled and standard deviation of the resistance of the memory cells. Next, the error of the full VMM can be computed by summing together expected error for all ADC reads. However, because multi-bit VMM is converted into several binary VMMs (sub-operations) followed by shift and add operations, the applied magnitude of each sub-operation must be considered. Therefore the expected error for a VMM will be the sum of the expected error for each ADC read times the applied magnitude of the sub-operation it belongs to. Therefore, the majority of error from CIM operations comes from high-magnitude sub-operations.

The simplest way to control accumulated variance and error is to reduce the number of word lines enabled at one time, but this will compromise the performance and energy efficiency of using

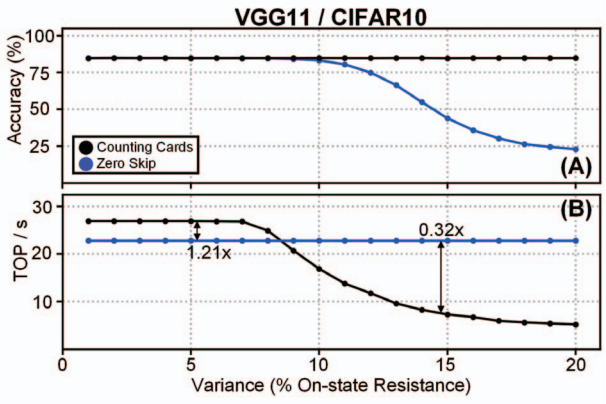

Figure 8: Classification accuracy and performance versus cell-to-cell variance for VGG11 on CIFAR10.

CIM. However, we have little choice in this trade-off because target applications require certain accuracy. In this way, the variance of the device should ultimately dictate the precision of the ADC used. Therefore, our objective function becomes achieving a target accuracy while maximizing performance. To find the optimal solution, counting cards enables more wordlines for low magnitude sub-operations and less wordlines for high magnitude sub-operations.

To set a target accuracy, an error threshold is set based on the application. Then, by computing the expected error for sub-operations, the threshold can be satisfied while enabling optimal performance. In Figure 8, we show the performance and accuracy of this technique compared to traditional zero skipping. This simulation was performed in [7], and used a cycle-accurate simulator to simulate CIFAR10 with variance cell-to-cell variances on a standard CIM accelerator. We observe that for low variance devices (1%-5%) this method yields higher performance than zero skipping, but in order to compensate for high variance memory (<10%), performance degrades while maintaining accuracy (as intended).

6 CONCLUSION

In this paper we introduced and analyzed some of the current challenges facing CIM with eNVM. Through our analysis, we identified several shortcomings of CIM as we work towards widespread adoption. Additionally, we reviewed recent solutions for these challenges and revealed some future research directions that have the potential to bring CIM closer to deployment in machine learning systems.

7 ACKNOWLEDGEMENT

This work was funded by the U.S. Department of Defense's Multidisciplinary University Research Initiatives (MURI) Program under grant number FOA: N00014-16-R-FO05 and the Semiconductor Research Corporation under the Center for Brain Inspired Computing (C-BRIC) and Qualcomm.

REFERENCES

[1] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. 2012. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology* 23, 7 (2012), 075201.

[2] Dmytro Apalkov, Alexey Khvalkovskiy, Steven Watts, Vladimir Nikitin, Xueti Tang, Daniel Lottis, Kiseok Moon, Xiao Luo, Eugene Chen, Adrian Ong, et al. 2013. Spin-transfer torque magnetic random access memory (STT-MRAM). *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 9, 2 (2013), 1–35.

[3] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[4] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 27–39.

[5] Brian Crafton, Sam Spetalnick, Yan Fang, and Arijit Raychowdhury. 2020. Merged Logic and Memory Fabrics for Accelerating Machine Learning Workloads. *IEEE Design & Test* (2020).

[6] Brian Crafton, Samuel Spetalnick, Gauthaman Murali, Tushar Krishna, Sung Kyu Lim, and Arijit Raychowdhury. 2020. Breaking Barriers: Maximizing Array Utilization for Compute In-Memory Fabrics. In *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE.

[7] Brian Crafton, Samuel Spetalnick, and Arijit Raychowdhury. 2020. Counting Cards: Exploiting Weight and Variance Distributions for Robust Compute In-Memory. *arXiv preprint arXiv:2006.03117* (2020).

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

[10] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–12.

[11] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. 2019. Design of reliable DNN accelerator with un-reliable ReRAM. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1769–1774.

[12] Yandong Luo, Xiaochen Peng, Ryan Hatcher, Titash Rakshit, Jorge Kittl, Mark S Rodder, Jae-Sun Seo, and Shimeng Yu. 2020. A Variation Robust Inference Engine Based on STT-MRAM with Parallel Read-Out. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[13] C Nail, G Molas, P Blaise, G Piccolboni, B Sklenard, C Cagli, M Bernard, A Roule, M Azzaz, E Vianello, et al. 2016. Understanding RRAM endurance, retention and window margin trade-off using experimental results and simulations. In *2016 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 4–5.

[14] Xiaochen Peng, Rui Liu, and Shimeng Yu. 2019. Optimizing Weight Mapping and Data Flow for Convolutional Neural Networks on Processing-In-Memory Architectures. *IEEE Transactions on Circuits and Systems I: Regular Papers* (2019).

[15] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.

[16] Xiaoyu Sun, Shihui Yin, Xiaochen Peng, Rui Liu, Jae-sun Seo, and Shimeng Yu. 2018. XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1423–1428.

[17] Xiaoyu Sun and Shimeng Yu. 2019. Impact of non-ideal characteristics of resistive synaptic devices on implementing convolutional neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 3 (2019), 570–579.

[18] H-S Philip Wong, Heng-Yuan Lee, Shimeng Yu, Yu-Sheng Chen, Yi Wu, Pang-Shiu Chen, Byoungil Lee, Frederick T Chen, and Ming-Jinn Tsai. 2012. Metal–oxide RRAM. *Proc. IEEE* 100, 6 (2012), 1951–1970.

[19] H-S Philip Wong, Simone Raoux, SangBum Kim, Jiale Liang, John P Reifenberg, Bipin Rajendran, Mehdi Asheghi, and Kenneth E Goodson. 2010. Phase change memory. *Proc. IEEE* 98, 12 (2010), 2201–2227.

[20] JY Wu, YS Chen, WS Khwa, SM Yu, TY Wang, JC Tseng, YD Chih, and Carlos H Diaz. 2018. A 40nm low-power logic compatible phase change memory technology. In *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 27–6.

[21] Tzu-Hsien Yang, Hsiang-Yun Cheng, Chia-Lin Yang, I-Ching Tseng, Han-Wen Hu, Hung-Sheng Chang, and Hsiang-Pang Li. 2019. Sparse ReRAM engine: joint exploration of activation and weight sparsity in compressed neural networks. In *Proceedings of the 46th International Symposium on Computer Architecture*. 236–249.

[22] Shihui Yin, Xiaoyu Sun, Shimeng Yu, and Jae-sun Seo. 2020. High-Throughput In-Memory Computing for Binary Deep Neural Networks With Monolithically Integrated RRAM and 90-nm CMOS. *IEEE Transactions on Electron Devices* 67, 10 (2020), 4185–4192.

[23] Jong-Hyeok Yoon, Muya Chang, Win-San Khwa, Yu-Der Chih, Meng-Fan Chang, and Arijit Raychowdhury. 2021. Reliable Write Operations and Multi-bit Encoding in high-endurance RRAM arrays. In *2021 IEEE International Reliability Physics Symposium*. IEEE, 1–2.