Special Session - Non-Volatile Memories: Challenges and Opportunities for Embedded System Architectures with Focus on Machine Learning Applications

Jörg Henkel[1], Lokesh Siddhu[1], Lars Bauer[1], Jürgen Teich[2], Stefan Wildermann[2], Mehdi Tahoori[1], Mahta Mayahinia[1], Jeronimo Castrillon[3], Asif Ali Khan[3], Hamid Farzaneh[3], João Paulo C. de Lima[3], Jian-Jia Chen[4,5], Christian Hakert[4], Kuan-Hsun Chen[6], Chia-Lin Yang[7], Hsiang-Yun Cheng[8]

[1] Karlsruhe Institute of Technology (KIT), Germany [2] Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
[3] TU Dresden, Germany [4] TU Dortmund University, Germany [5] Lamarr Inst. for ML and AI, Germany
[6] University of Twente, the Netherlands [7] National Taiwan University, Taiwan [8] Academia Sinica, Taiwan

ABSTRACT

This paper explores the challenges and opportunities of integrating non-volatile memories (NVMs) into embedded systems for machine learning. NVMs offer advantages such as increased memory density, lower power consumption, non-volatility, and compute-in-memory capabilities. The paper focuses on integrating NVMs into embedded systems, particularly in intermittent computing, where systems operate during periods of available energy. NVM technologies bring persistence closer to the CPU core, enabling efficient designs for energy-constrained scenarios. Next, computation in resistive NVMs is explored, highlighting its potential for accelerating machine learning algorithms. However, challenges related to reliability and device non-idealities need to be addressed. The paper also discusses memory-centric machine learning, leveraging NVMs to overcome the memory wall challenge. By optimizing memory layouts and utilizing probabilistic decision tree execution and neural network sparsity, NVM-based systems can improve cache behavior and reduce unnecessary computations. In conclusion, the paper emphasizes the need for further research and optimization for the widespread adoption of NVMs in embedded systems presenting relevant challenges, especially for machine learning applications.

KEYWORDS

Non Volatile Memories, Machine Learning, Compute In Memory, Design Space Exploration

1 INTRODUCTION

Due to advancements in manufacturing, the utilization of non-volatile memories (NVMs) in embedded systems has gained traction [32, 56]. Many NVMs offer the advantage of storing multiple bits in a single memory cell (called a multi-level cell, MLC), increasing memory density. In addition, NVMs provide scalability, lower power consumption, the ability to compute-in-memory (CIM), and non-volatility [40]. However, its adoption has been limited due to reduced write performance and endurance.

Both academia and industry have explored many NVM technologies, such as Phase Change Memory (PCM), Spin-Transfer Torque RAM (STT-RAM), and Ferroelectric RAM (FRAM). Furthermore, NVM technologies are gaining relevance across multiple application domains of embedded systems that are battery-powered or rely on energy harvesting, including wireless sensor nodes, Internet of Things (IoT) devices, and wearable electronics. In conventional computing, it is assumed that sufficient energy is available to perform and finish the computations, which is, however, not guaranteed with batteries and energy harvesting. In this context, *intermittent computing* is an emerging paradigm where systems are designed such that they operate during periods where energy is available, interrupted by periods of power shortages to regain energy, e.g., by means of energy harvesting (see, e.g., [48, 61]).

One of the critical challenges of intermittent computing is that power availability is unpredictable: Power shortages can appear at any point in time. Conventional computing systems (as depicted on the left in Fig. 1) operate with a non-volatile disk and volatile main memory, caches, and registers. They require file systems and the support of operating systems, on top of which mechanisms and data structures for providing persistence and consistency must be implemented. However, these mechanisms generally induce high latency and energy overheads that are not tolerable in many embedded system domains. NVM technologies open a new perspective for the design of intermittent computing systems, as they raise the point of persistence closer to the CPU core.

Another prominent capability of NVM technologies is Computation in Memory. In current von Neumann architectures, computing efficiency is fundamentally bottlenecked by the data movement between physically separated processing elements and memory. Therefore, specialized hardware that offers processing capabilities where the data resides without the need to move it becomes a key to accelerate machine learning algorithms that demand a high volume of data.

Although NVM-CIM (Computation in resistive non-volatile memory) systems can be orders of magnitude more efficient compared to conventional computing systems, their reliability is challenging. This challenge originates from the analog computation capability

CASES '23 Companion, September 17–22, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0290-7/23/09…$15.00
https://doi.org/10.1145/3607889.3609088

as well as the device non-idealities, which need to be precisely considered. Moreover, their overall efficiency depends on the efficient mapping of the input kernel and the utilization of the underlying system. Presently, low-level device function calls are used to manually program these devices, which makes the operability of these devices restricted. This is especially challenging for more complex applications and heterogeneous CIM systems, as mapping compute primitives onto various CIM targets becomes a non-trivial task.

Recently, multi-level intermediate representation (MLIR) [29] based compilation frameworks have been developed that handle program representations in high-level domain-specific languages and transparently lower them to various CIM accelerators [26, 46]. These frameworks consist of a hierarchy of abstractions that implement domain- and device-specific analysis and optimization passes. Although these frameworks have demonstrated impressive performance for matrix multiplication and other machine learning applications and, in some cases, have generated codes comparable to the hand-optimized code, the acceleration of stencil applications and bulk bit-wise logic operation on CIM systems remains an open challenge. The latter requires a more rigorous analysis of the input application and a thorough exploration and characterization of the underlying system.

Such compute-in-memory paradigms also benefit various machine-learning applications. Machine learning has been widely involved in our contemporary society and has become integral to everyday life in different learning principles and models. Many of the advances and applications of machine learning have been possible due to increases in computation power [23]. In contrast, memory heavily required by machine learning has become a bottleneck, typically referred to as the memory wall. Although it is relatively easy today to use hundreds of processors in parallel on a GPU or application-specific processors designated for specific applications, the challenge of data movement still needs to be addressed [44]. Since data movement occurs through a narrow bus with high latency and limited bandwidth, it is not surprising to see that recent innovations have changed from processor-centric to memory-centric designs [19].

Several advanced techniques have demonstrated the great potential of NVMs, which can effectively mitigate or even eliminate such issues incurred by conventional architectures. For instance, CIM, as mentioned above, is of particular interest to the machine learning domain, since such accelerators can perform vector-matrix and matrix-matrix multiplication in constant time, which are heavily utilized in the execution of deep learning models. However, from the machine learning domain, it is common to study from a high-level perspective with limited consideration of the underlying hardware. In fact, there is also useful meta-information that can be abstracted and utilized during the system design.

One prominent example is the probabilistic view of the execution of a decision tree [6]. During the training phase, the comparison at each node is a Bernoulli experiment. Due to the nature of supervised learning, the probabilities collected over the training phase shall likely reflect the distribution of the inference phase, which can be further leveraged for optimizing the memory layout to benefit the cache behaviors [5]. Similarly, the sparsity in common neural networks can be exploited to reduce ineffectual computations. NVM-based CIM's tightly coupled crossbar structure makes it challenging to skip the computations with all the zeros in filter weights and input activations. However, for a practical design with reduced computation parallelism to alleviate errors induced by device non-ideality [35], it is possible to jointly exploit weight and activation sparsity at fine granularity to achieve satisfactory inference accuracy while delivering comparable performance and energy efficiency [64].

Table 1: Latency-retention time tradeoff in MLC PCM [67].

Write Mode	Write Latency [ns]	Normalized Memory Energy	Retention time [s]
Fast	550	0.84	2.01
Medium	700	0.869	24.05
Slow	1150	1	3054.9

The design and optimization of neural network architectures and machine learning algorithms that are most suitable for utilizing NVM architecture remain an open challenge. The limitation and improvement of beyond-von Neumann architectures w.r.t. performance and energy efficiency have yet to be fully revealed. The lack of fundamental exploration of machine learning algorithms on beyond-von Neumann architectures hinders the possibility of tuning the precision of the machine learning algorithms (i.e., at the software level) and in the underlying NVM (i.e., at the HW level).

In this paper, we focus on the challenges and opportunities that arise from using NVM technologies for machine learning in embedded systems. We look at three important themes: (i) the integration of NVMs into embedded systems, (ii) leveraging the benefit from computation in resistive NVMs, and (iii) memory-centric machine learning. The rest of the paper continues as follows. Section 2 discusses the three themes in more detail. In Section 3, we explore the challenges in adopting NVM memories. The current state of research is discussed in Section 4. Finally, section 5 summarizes the challenges and opportunities of using NVM for machine learning in embedded systems.

2 UTILIZING NON-VOLATILE MEMORIES FOR EMBEDDED MACHINE LEARNING

In this section, we study the three important themes in detail. We focus on integrating NVMs into embedded systems, notably in intermittent computing, when systems operate during periods of available energy. Following that, we explore computation in resistive NVMs to overcome the memory wall barrier. Furthermore, we discuss memory-centric machine learning.

2.1 Embedded Systems with NVMs

Utilizing fast/slow write modes: Previous studies have investigated strategies to mitigate reduced write performance. Researchers have explored the utilization of fast/slow write modes in MLC-PCM memories [32, 40, 56, 67]. The fast mode provides low latency and write energy(Table 1 [67]). However, data have a short retention time, necessitating periodic refresh [10]. On the other hand, slow writes provide extended retention time, but come with timing and energy consumption drawbacks. PCM memory also offers additional write modes of medium speed, enabling a trade-off between write latency and retention time.

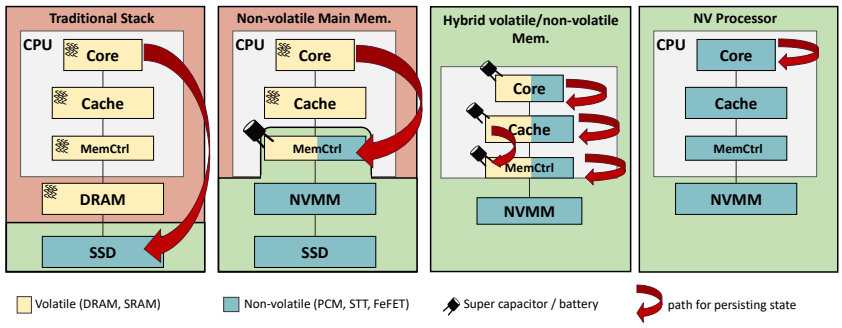

Figure 1: Overview of the design space of embedded CPU systems with NVMs in contrast to traditional memory organization, ranging from NVMM, over mixed volatile/non-volatile memory hierarchies to NVPs.

These different write modes can be utilized by carefully selecting which memory writes demand a high retention time, and thus slow writes need to be used, and which memory writes will be overwritten soon (short lifetime) and thus can benefit from the energy and latency savings of short writes. For instance, stack accesses are often timewise local accesses, while executing the code of a function, i.e., short writes may be suitable often. However, the oldest functions on the stack frame require a longer retention time. Sophisticated compile-time static data-flow analysis could be used to precisely determine the appropriate write mode. But that is challenging because the analysis needs to make assumptions about memory access latencies, which is difficult before deciding the memory write latencies for variables.

Design space of NVM in embedded systems: There are different design options to leverage non-volatile memory for embedded CPU systems as illustrated in Figure 1. An option is to provide embedded systems with *nonvolatile main memories (NVMM)*, which would even make disks obsolete. However, despite offering byte-addressability and persistence, programming efficient data structures that survive power shortages is extremely difficult. To achieve recoverability, it is not enough to simply store an in-memory data structure in NVMM. Instead, one has to carefully think about ordering every single write and explicitly insert write-back and fence operations, as else information about modified data in volatile caches would get lost with a power shortage. Architectures like x86 and ARM therefore support persistent main memory by instructions for cache flushes (explicitly writing back dirty cache lines to memory) and barriers/fences (enforce the given order of instruction execution in out-of-order pipelines). However, there exist several architecture-specific pitfalls which prevent a generic, cross-architectural solution. As an example, while x86 ensures total store order, no such guarantees exist for ARM architectures (this means that ARM processors are allowed to reorder store operations that do not have dependencies). Figure 2 illustrates the architecture-specific differences that need to be taken care of when implementing a persistent linked list.

In contrast to NVMM, *non-volatile processors (NVPs)* make use of NVM throughout the memory hierarchy. As registers and caches are non-volatile, no explicit cache flushes and barriers/fences and complex data structures are required for providing persistence in case of power shortages. However, NVM suffers from energy-

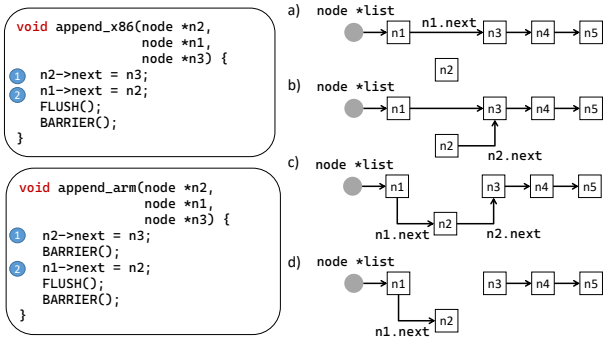

Figure 2: Effect and differences w.r.t. guarantees of total store order (TSO) as in x86 versus no such guarantees as in ARM by means of a simple example of implementing a persistent linked list. Executing `append_x86()` on an x86 architecture guarantees store order, so that (1) is executed before (2) resulting in always consistent internal state b) and then c). When executed with no-TSO, (2) could be executed before (1) resulting in inconsistent state d): After a power shortage, all nodes starting from `n3` would be unreachable from `node *list`. The correct ordering of (1) and (2) can only be guaranteed by adding a barrier as illustrated in `append_arm()`.

and time-expensive writes. As a consequence, NVPs can only be operated with low clock frequencies. Another issue is that NVM technologies suffer from write endurance being several orders of magnitude lower than for their volatile counterparts. Techniques such as wear-leveling for caches or main memory would be too expensive at the level of registers or register files or lower-level caches.

Architectures with a hybrid *volatile/non-volatile memory hierarchy* combine both volatile and non-volatile caches in a multi-level cache hierarchy. They aim to bridge the gap between the low write latencies and high endurance of volatile caches and the high density and persistence of non-volatile memory, offering a balance between performance, energy efficiency, endurance, capacity, and data persistence. There is a huge design space, since the volatility boundary does not necessarily have to be set between two cache levels. Rather, a mixture of volatile and non-volatile caches within

the same level can also be used. Upon a power shortage, a processor has to automatically persist in the state of computation such that after the power is back, the program is able to continue its execution. Particularly, all non-persisted dirty lines of non-volatile caches have to be written to non-volatile components of the hierarchy following a selective writeback mechanism. This checkpointing has to be backed up by a supercapacitor that provides sufficient energy for this process.

2.2 Computation in Resistive NVMs

Computation in resistive non-volatile memory (NVM-CiM) is a promising approach to overcoming the memory wall problem. On one side, by empowering the memory subsystem to perform certain computations, the back-and-forth data transfer between processor and memory units is reduced, improving the overall performance and energy efficiency. On the other hand, the resistive nature of the NVM devices enables *analog computation* to natively perform operations on large vectors (memory rows) in parallel, further improving performance and energy efficiency.

Despite the aforementioned advantages of the NVM-CiM, however, there are some reliability issues stemming from the NVM device non-idealities and analog computing, which can even overshadow its performance benefits. Therefore, the reliability aspects of the NVM-CiM concept need to be carefully investigated. The reliability aspects of the NVM-CiM are dependent on the multiple parameters which we will discuss in the following. Here, we focus on the reliability aspects of the NVM-CiM which can perform Boolean operations.

The Boolean operation can be executed on the NVM-CiM hardware with the concept of *scouting logic* [60]. In this non-stateful scheme, the resistive states of the NVMs, being in the low or high resistive state (LRS or HRS), encode the operands of the Boolean operation. On the basis of the resistive states of the operands, the output signal can have a unique analog value. The binary result of the Boolean operation, however, is determined based on a comparison operation. Using a sense amplifier, the output signal is compared with a reference signal. If the analog output signal is less than the reference, the result of the Boolean operation is '0' otherwise is '1'.

A critical reliability-threatening factor in the scouting-based NVM-CiM is the effect of process variation (PV). Due to the PV on the NVM devices, the distinct resistive levels of the NVM devices do not have a fixed value, instead, they follow the statistical distribution. Due to the existence of such a statistical distribution, *decision failure* can occur at the tails of the distribution. Decision failure is a scenario in which though the output is supposed to be less (greater) than the reference, because of the PV, it becomes the other way around, changing the outcome of the Boolean operation. Figure 3 (a) and (b) show the concept of NVM-CiM based on scouting and its susceptibility to decision failure, respectively.

To reduce the probability of failure of the decision, the overlap region (see Figure 3) should be small. The larger HRS-LRS ratio of NVM technology and better control of photovoltaic energy during the fabrication process can reduce the probability of decision failure.

The *NVM technology* and the *number of the scouting-based NVM-CiM inputs* are two impactful parameters on the decision failure

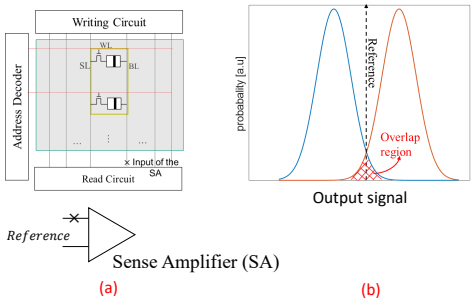

Figure 3: (a) Performing scouting-based NVM-CiM, (b) Decision failure in the overlapping area between two signal distributions.

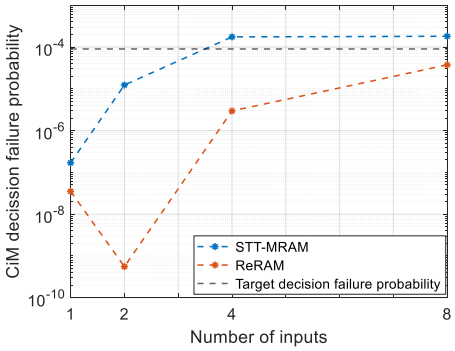

Figure 4: CiM decision failure vs. the number of inputs in STT-MRAM, ReRAM during scouting-based NVM-CiM.

probability. From the NVM technology point of view, the larger the HRS-LRS ratio, the lower the decision failure. Among different NVM technologies, spin transfer torque RAM (STT-MRAM) and redox-based RAM (ReRAM) are two representative technologies that can provide small and large HRS-LRS ratio, respectively.

The *number of the scouting-based NVM-CiM inputs* is the other impactful factor on the decision failure. The higher number of CiM inputs indicates a smaller sense margin (harmful for the CiM decision failure) as well as smaller standard deviations (σ) (helpful for the CiM decision failure). In fact, these two effects counter each other and the ultimate effect is technology-dependent. Figure 4 shows the decision failure probability with respect to the number of scouting-based NVM-CiM inputs. According to Figure 4, the decision failure probability has an increasing trend with the number of inputs. However, as discussed, the countering impact of higher input numbers on the overlapping region of the output distributions, for ReRAM, the minimum decision failure happens for two inputs.

Table 2 compares the latency and energy overhead of scouting-based NVM-CiM operation for different numbers of inputs in STT-MRAM and ReRAM technologies. While increasing the number of inputs for performing more complex operations is beneficial in terms of energy and latency, however, as shown in Figure 4, due to reliability limitations, it cannot be extended beyond a certain point, which is also technology-dependent.

Table 2: The latency and energy of the scouting-based NVM-CiM for different numbers of inputs in STT-MRAM and ReRAM technologies, the array size is 256 kB.

Number of CiM inputs	STT-MRAM		ReRAM	
	Latency	Energy	Latency	Energy
1 (Read)	4.00 nS	77.4 pJ	45.9 nS	973 pJ
2	4.02 nS	78.2 pJ	45.8 nS	970 pJ
4	4.09 nS	81.1 pJ	45.9 nS	973 pJ
8	4.22 nS	85.9 pJ	46 ns	978 pJ

Although CIM systems have demonstrated superior performance compared to conventional CPU-centric systems in various application domains [44, 47], translating applications written in a high-level programming language and its efficient mapping onto a CIM accelerator remains an open challenge. Presently, most available CIM and CNM systems, including the commercially available UPMEM system [53], mainly provide low-level device libraries and leave the mapping problem and optimizations to the programmer. As a result, the accessibility of these innovative architectures is limited to device experts, impeding their widespread adoption. Even for these experts, the process of manually rewriting large applications and simultaneously making intelligent mapping decisions to meet specific constraints is a tedious and non-trivial task. These programmability challenges can be overcome by developing high-level compilation flows that leverage the high-level program description and architectural specifications, and by conducting extensive analysis can generate code that effectively harnesses the capabilities of the underlying architecture.

Compilation for CIM systems: The CIM paradigm leverages the unique physical attributes of memory devices to directly implement compute and/or logic operations within memory. Different devices exhibit varying levels of efficiency in implementing logic or compute operations, for which we expect future general-purpose CIM architectures to be highly heterogeneous. To effectively program such systems, there is a need for novel abstractions and compiler frameworks that support both device-agnostic and device-specific optimizations. In recent research, the multi-level intermediate representation (MLIR) [29] has emerged as a powerful toolkit for developing hierarchical flows and targeting CIM systems. MLIR enables the progressive lowering of abstractions, facilitating the analysis and transformations at each abstraction level to reason about the computational primitives and memory behavior of an input application.

For instance, in the Open CIM Compiler (OCC) [46], designed to accelerate operations like matrix multiplication (matmul) on PCM-based crossbars, the `CIM` abstraction not only performs PCM-specific optimizations but also replaces various instructions with their corresponding device library calls. CINM, a more recent compilation framework, abstracts over various CIM and CNM devices and provides an end-to-end compilation framework for heterogeneous systems [26]. For each supported CIM and CNM system, it introduces a device-aware abstraction that implements unique device-specific optimizations and library function calls. The hierarchical lowering in CINM allows for the identification of the most appropriate CIM target for each compute primitive in the input application and its mapping onto it.

```
def forward(self, input, weight, bias, stride, padding, dialation, groups):
    return torch.ops.aten.conv2d(input, weight, bias, stride, padding, dialation
        ↪ , groups)
```

(a) PyTorchScript code for Conv2D

```
...
linalg.conv_2d ins (%Img, %Filt: memref<?x?xf32>, memref<?x?xf32>) outs (%Out:
    ↪ memref<?x?xf32>)
...
```

(b) MLIR (linalg) representation of Conv2d

```
...
%0 = linalg.im2Col(%Img)
%1 = linalg.im2Col(%Filt)
linalg.matmul(%0, %1, %OutTmp)
linalg.Col2Im(%OutTmp, %Out)
...
```

(c) MLIR (linalg) representation of Conv2D after rewriting

```
%tile0 = cim.copyTile(%0, %i, %k)
%tile1 = cim.copyTile(%1, %k, %j)
cim.write(%id, %tile1)
cim.matmul(%id, %tile0, %tempTile)
cim.barrier(%id)
cim.accumulate(%tileOut, %tempTile)
cim.storeTile(%tileOut, %OutTmp, %i, %j)
```

(d) MLIR (CIM) version of Conv2D

Figure 5: Progressive lowering of a 2D convolution for CIM

Analysis and optimizations: These high-level compilation flows not only facilitate the programming of CIM systems but also optimize them. In addition to conventional compiler transformations such as parallelism or locality optimizations, these flows can be leveraged to rewrite non-CIM-friendly compute primitives into CIM-friendly primitives, minimize the number of writes for NVM-based CIM devices, and fuse operators to reduce the overall computational complexity of a kernel.

To illustrate, let's consider a CIM-accelerator for matrix multiplication (matmul) kernels and a two-dimensional convolution kernel written in PyTorch (see Figure 5a). In both the OCC and CINM frameworks, the front-end first lowers the PyTorch code to a high-level domain-specific abstraction (shown in Fig. 5b). The analysis passes at this abstraction determine that the convolution operation can be rewritten as matmul operation, shown in Fig. 5c. Subsequently, the matmul primitive in the rewritten code is offloaded to the CIM accelerator using the CIM abstraction, shown in Figure 5d.

Various loop and layout transformations can be applied to improve the overall performance and lifetime of the CIM systems and reduce their energy consumption. Tiling is typically used to adapt large-size input kernels to fit onto the fixed-size CIM arrays or enable parallel execution. On the other hand, loop reordering is applied to reduce the number of writes on NVM-based CIM systems.

2.3 Memory-Centric Machine Learning

Architectural-level Optimization Despite the NVM-based DNN accelerators can mitigate costly data movements encountered in conventional von Neumann machines, bringing such a system into practice remains challenging due to hardware constraints, such as the imperfect device and high-overhead analog-digital conversion. The impact of the hardware constraints varies as different NVM devices and peripheral circuits are leveraged and highly depends on the characteristics (e.g., error-tolerance ability, model sparsity, etc.) of the target DNN model. One promising way to tackle the

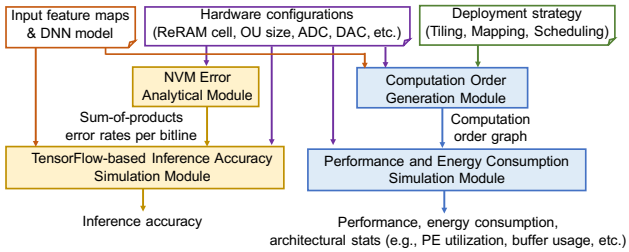

Figure 6: Overview of DL-RSIM [35] simulation framework.

challenge is to optimize the design at the architecture level, considering both hardware constraints and software features. In the following, we illustrate two example approaches that can be used to improve the reliability and energy efficiency of in-NVM DNN inference: (a) a simulation framework facilitating device-architecture co-design [35] and (b) an innovative architecture enabling joint exploitation of weight and activation sparsity in DNN models [64].

In recent years, several researchers have developed simulation frameworks for NVM-based DNN accelerators to facilitate design space exploration [8, 28, 30, 59]. These simulation frameworks enable comprehensive analysis on reliability, performance, and energy efficiency. Nevertheless, most of these prior studies are oblivious to the necessary architectural changes for maintaining inference accuracy when conducting performance and energy efficiency analysis. Specifically, due to the accumulated effect of device variation, only a limited number of wordlines and biltines, namely Operation Unit (OU), can be activated per cycle in a practical design [35, 64]. This OU constraint affects computation parallelism and should be considered when analyzing performance and energy efficiency instead of assuming the entire crossbar array can be operated concurrently. DL-RSIM [35] is the rare one that takes the OU constraints into consideration to facilitate device-architecture co-design. It realizes inference accuracy and performance/energy efficiency analysis using four modules, as shown in Figure 6, and can be incorporated with any DNN model implemented by TensorFlow. Here we explain one example of using DL-RSIM for device-architecture co-design. Through the inference analysis provided by DL-RSIM, we can find a good OU size for the selected NVM device and the target DNN model to achieve satisfactory inference accuracy [11]. Then, based on the selected OU size, we can leverage the performance/energy efficiency analysis portion of DL-RSIM to explore different model deployment strategies (i.e., weight mapping and operation scheduling) to optimize performance and energy efficiency. This enables cross-layer joint optimization taking both reliability and energy efficiency into consideration.

Considering the OU constraint, a practical design is likely to deliver lower performance compared to an over-idealized design [45] that neglects the accuracy loss caused by the accumulated device-induced errors since less computation is done in one cycle. The tradeoff between performance and accuracy can be resolved if we could exploit finer-grained sparsity offered by the OU-based design [64]. In an over-idealized design, weights stored in the same worldline need to multiply to the same input, and the accumulated current flowing through the same bitline contribute to the same output. Thus, weight sparsity in a DNN model can be exploited to skip redundant computations only when the entire worldline or

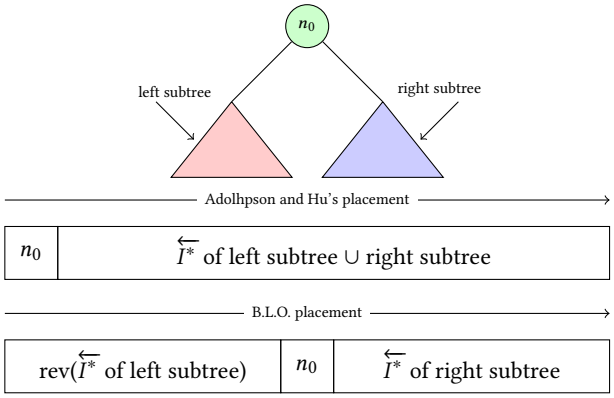

Figure 7: Suboptimal placement correction, derived from [21]

bitline contains zeros. Unlike the over-idealized design, each OU in a crossbar is operated independently in practice. This creates more sparsity exploitation opportunities as computations can be skipped as long as a row or column within an OU contains zeros. Similarly, finer-grained activation sparsity can be leveraged in OU-based design than the over-idealized counterpart. In addition, more redundant computations can be saved by adding a simple circuit module to dynamically activate only the wordlines with non-zero inputs. Combining row-wise weight compression with dynamic worldline activation makes it possible to achieve satisfactory inference accuracy considering the limitation of ReRAM cell reliability while delivering comparable performance and energy efficiency with the over-idealized counterpart [64]. This indicates the prominence of architectural-level optimization.

Inference Optimization on NVMs To serve the machine learning models, NVMs can be used in various forms, such as main memory, scratchpad memory, or CIM accelerator. Since each NVM features its unique execution properties, the corresponding cost model should be first figured out to argue the decision-making. Afterward, the execution pattern of trained models has to be analyzed to match the architecture behavior. Suitable optimizations thus can be introduced to rearrange the deployment. Two interesting cases have been studied and optimized, i.e., decision tree ensembles on racetrack [21, 22], and convolution neural network (CNN) on the memristor-based CIM accelerator [52], presented below.

Considering a low-power embedded systems for machine learning inference, the target system equips racetrack memory (RTM) as integrated scratchpad memory to store the trained decision trees. Note that data on RTM cannot be randomly accessed. It needs to be shifted until the targeted data aligning with an access port to be read out [3]. Several works have proposed data placement heuristics to minimize shift latency for general applications [9, 27]. With decision trees, domain-specific placement approaches can be developed to optimize the memory layout on RTM. A Bidirectional Linear Ordering (B.L.O.) algorithm has been presented (see Figure 7), which resembles an existing solution for constrained rooted trees and eliminates the major cause for long shift distances between two inferences [21, 22]. The results show that B.L.O. outperforms general heuristics in terms of shifts, runtime, and energy consumption.

Although the memristor-based CIM accelerator has demonstrated its promising potential, recent work has shown that, due to the imperfect circuits and devices, the accumulated effect of per-cell current deviation and ADC overhead might degrade inference accuracy when operating the entire crossbar array directly [34]. To achieve satisfactory inference accuracy, only limited wordlines and bitlines in a crossbar should be operated at once, namely Operation Unit (OU) [34, 64]. For example, when mapping trained CNN models to the CIM accelerator under such a hardware constraint, the design space can be divided into spatial (ie, mapping of weight values and multiplication results onto crossbar cells) and temporal developments (ie, the execution order of matrix multiplication) [52]. With an estimation model for end-to- end inference latency, an optimized scheduling strategy can be effectively derived.

Training against Reliability Issues Utilizing NVM for inference under CIM requires the machine learning algorithm to be trained to be aware of accuracy and error tolerance under given resource constraints, e.g., area, energy, and cost. Recent results by Buschjäger et al. [4] and Yayla et al. [66] have demonstrated that the accuracy and error-tolerance optimization of binarized neural networks (BNNs) can be achieved by error injection during training in combination with cross-entropy loss or by maximizing margins in the BNN. Results for joint optimization of accuracy and bit-error tolerance for quantized neural networks have been demonstrated by Stutz et al. [49].

3 CHALLENGES

Several challenges are faced in the design and use of NVM technologies. This includes the mixed/hybrid design space of non-volatile memory architectures, the need for accurate modeling of NVM parameters, and the challenges of computation in Resistive NVMs. These challenges are discussed in detail below.

3.1 NVM Parameters

Many different NVM technologies were developed over the last years, partially vastly different in the underlying concepts, how they work, etc. And even within one NVM type, let us say STT-RAM, the technology changed significantly over the generations and years, which means that established planar STT-RAM has very different properties than recent perpendicular STT-RAMs. We can conclude that investment in research in hardware and/or software architectures that may be very specific to a certain technology and also change easily with advances in these technologies. Targeting NVMs in general is very challenging, and even obtaining reasonable model parameters to simulate NVM technologies is difficult [25].

3.2 Complexity of the Design Space for Computation in Memory

Designing mixed/hybrid volatile/non-volatile memory architectures poses challenges due to the speed discrepancy between NVM and volatile memory technologies. Careful consideration of various design parameters is required to optimize performance, energy efficiency, endurance, and data persistence. The design space involves selecting appropriate memory technologies, organizing the memory hierarchy, designing memory controllers, and developing strategies for data placement and migration. Simulating design options and considering NVM parameters are essential. NVM-CiM systems introduce reliability concerns, complicating the exploration of design space. Making intelligent offloading decisions for compute primitives, considering performance, energy gains, and bitwise logic operations, remains challenging. Target selection in heterogeneous systems requires complex cost models, while co-optimizing performance, energy, and reliability adds complexity from the application to the technology level.

3.3 WCET Analysis and Optimization for NVM

Providing safe upper bounds on the worst-case execution time with the presence of NVMs requires modeling and assessment of timing properties to capture the characteristics of disruptive embedded NVM technologies. WCET analysis has to account for features introduced by NVMs, such as data retention and wear leveling. As timing properties of NVMs may vary over time due to activities related to data retention or wear leveling, it is crucial to develop parametric worst-case timing analysis to include such variations, which will be utilized to devised optimized memory management to reduce the WCET of a program. Deep explorations on the suitable WCET-aware memory controllers and near-memory accelerators are needed to better utilize the NVM. Hardware/software co-design approaches can be investigated to look for suitable hardware/software configurations to reduce the WCET of a program.

3.4 Cross-layer optimization considering hardware constraints

Even though in-NVM DNN inference is promising, the unreliable computation induced by imperfect NVM devices and the large analog-to-digital converter (ADC) overheads hinder its realization [35]. Tackling this challenge through device/circuit-oriented solutions relies on technological advancement. Cross-layer optimization methods, with the assistance of application and architecture design, can better deal with the challenge. For example, jointly considering the error-tolerance ability of the target DNN model and the underlying device property could help to adapt architecture-level configurations (e.g., OU size, data encoding method, model mapping strategy, etc.) to maximize computation parallelism while maintaining satisfactory inference accuracy. Mutually optimizing the DNN training method and architecture-level configurations could also help to better exploit the sparsity feature of the target DNN model to skip ineffectual computations. Although cross-layer optimization is challenging as multiple design points need to be jointly considered, it is envisioned to be essential to bring in-NVM DNN inference into practice

3.5 Model-specific mapping strategies

Given trained models, different mapping strategies are needed to effectively match different types of NVM. Each strategy has to tailor its placement policy with suitable granularity and takes corresponding cost metrics and model-specific access pattern into account. For example, when using a memristor-based CIM accelerator, the accumulated effect of per-cell current deviation degrades inference accuracy when operating the entire crossbar array within a single cycle. To obtain the satisfactory results, the weights of neural networks should be mapped on to the crossbar with the awareness of

available wordlines and bitlines at once, so-called Operation Unit (OU) [34, 64]. In addition, tree-based learning models can also be mapped to such NVM accelerators, e.g., memristive analog content-addressable memory (CAM) [41] and 3D digital ternary CAM [69], or racetrack scratchpad memory [22]. The effectiveness of such mapping strategies often depends on understanding access patterns. Board explorations on the cost metrics and the execution models are thus needed.

3.6 NVM Reliability

In the context of ML, non-volatile memories, including ReRAM, face significant challenges related to write endurance, faults, and noise. Write endurance refers to the limited number of write cycles before memory degradation occurs, posing a concern for ML systems with frequent write operations. Faults can originate from sources such as stuck-at, sneak path, and write disturbance, leading to data integrity issues. Noise in non-volatile memories can corrupt stored information, impacting the reliability of ML algorithms. Various mitigation methods have been proposed that involve error correction codes, adaptive algorithms, and architectural techniques that distribute writes, employ wear leveling, and implement error detection and correction mechanisms. However, such methods have significant space/time overheads. Designing low-overhead approaches to enhance non-volatile memories' longevity, reliability, and performance in ML applications is an open challenge.

4 LITERATURE SURVEY

Several works have explored techniques to optimize and enhance NVM structures and system architectures. They discuss optimizations for read and write operations, mitigate endurance issues, and design methods for efficient NVM architectures. This includes fast/slow write modes, wear-leveling techniques, design space exploration for embedded systems, programming frameworks for resistive NVMs, and architectural optimizations for memristor-based machine learning accelerators.

4.1 Optimizing and Leveraging NVM Characteristics

Pan et al. [40] used the on-chip scratch pad RAM to minimize the slow writes to NVM memories. Chen et al. [10] considered data encoding schemes to enable efficient recovery for retention time violations. Li et al. [32] suggest estimating the retention time during compilation to add fast write instructions. However, such a framework has limited modeling of various hardware structures and assumes no caches. In [67], Zhang et al. suggested QuicknDirty (*QnD*), which uses the memory busy and (relatively) idle phases. During busy phases, the processor utilizes the fast write mode, and when memory is idle, data is refreshed using slow writes. *QnD* is implemented in hardware (memory controller) and does not consider a variable's lifetime.

To mitigate the relatively limited endurance issue, several techniques have been proposed in the literature. One strategy is to reduce actual write accesses to NVM, for example, data migration [13], caching [43], and write reduction [12]. Another popular strategy is to amortize the wear-out of cells over all pages or with finer granularity. Specifically, effective granularity can vary from single bits [12, 15], over cache lines [42] for fine-grained approaches to memory pages [7, 16, 18, 20] or even larger memory segments [70]. Depending on the size of effective regions and how the wear-leveling approach is triggered. By taking into account the current age of the cell, some aging-aware approaches [7, 18, 20] operate on memory blocks, whereas some wear-leveling approaches [16, 20, 42, 70] operate in a circular or randomized manner inside memory blocks. Interestingly, several approaches like [16, 20, 70] periodically shuffle or rotate the targeted memory blocks, but their feasibility has never been investigated for real-time systems.

4.2 Design space of NVMs in embedded systems

NVMM has become commercially available in the desktop and server market with Intel Optane based on x86 architectures. In the domain of embedded systems, ARM is the predominant computer architecture. Due to architectural differences, architecture-specific persistent data structures have to be provided, as presented in [55]. Programming such data structures can be tedious and imposes overheads in terms of latency and energy consumption during normal operation. A further challenge is the asymmetric read/write latency and power consumption of NVMM, which requires novel cache management strategies [58, 63]. Various architectural options for providing NVP have been discussed. NVPs are currently restricted to niche applications with rather low performance requirements and ultra-low power consumption, for example, using energy harvesting [36, 37]. Likewise, various design options based on mixed/hybrid volatile/ non-volatile memory hierarchies have been discussed. Examples of such design variants include hierarchies containing a non-volatile last-level cache [39], a hybrid L2 cache consisting of a volatile SRAM and a non-volatile STT-RAM [57], or a hybrid volatile/non-volatile L1 cache [61]. However, there exists neither a comparative study of these design options nor any support of design automation to automatically explore the complex design space.

4.3 Programming frameworks for CIM systems

For memristor accelerators, Ambrosi et al. proposed a software stack with an ONNX backend that generates ISA code from a graph representation of a neural network model [1]. Ankit et al. built upon this work and developed a run-time compiler as a C++ library, requiring application rewriting with the proposed API [2]. Similarly, Fujiki et al. proposed a compilation flow that lowers Google's TensorFlow data flow graph into simpler instructions that are supported by the in-memory accelerator [17].

In contrast to these manual rewriting of the applications, automatic compilation flows that detect CIM-amenable primitives and lower them to device API calls without user intervention are also proposed [14, 54]. These frameworks use the low-level LLVM intermediate representation (IR) and the schedule trees in the polyhedral compiler to transparently detect and offload CIM-amenable patterns to the CIM accelerators. Although motif detection has been proven to be robust compared to prior approaches, these tools can still fail and miss offloading opportunities due to the lower abstraction of the LLVM IR.

4.4 Architectural-level optimizations for memristor-based DNN accelerators

Many prior studies have proposed various methods to optimize the design of memristor-based DNN accelerators from the architectural perspective. Since different hardware configurations (e.g., type of memristor devices, peripheral circuits, array architecture, interconnect, etc.) affect reliability, performance, and energy efficiency, several simulation frameworks have been developed to facilitate design space exploration [8, 28, 30, 59]. These works provide comprehensive analysis, but the impact of different model mapping and operation scheduling strategies is not explored. Some researchers have proposed various mapping strategies to achieve efficient DNN inference [31, 45]. These works map all weights of a layer onto the same crossbar and replicate weights to exploit the parallel computing ability of multiple crossbars. However, considering the OU constraint, these strategies cannot fully utilize available crossbar cells to maximize the parallel degree. In addition to mapping strategies, several prior studies have proposed to exploit model sparsity to reduce ineffectual computations for better energy efficiency. Considering the tightly coupled crossbar structure, these works leverage structurally compression methods [24] or clustering techniques [33] to skip the computations with irregularly distributed zeros in weight and activations. These works overlook the inference accuracy loss caused by non-ideal memristor devices and assume an over-idealized architecture. Some recent work considers the OU constraint and propose methods to reuse the output results derived from the same repetitive pattern in weights [68] and activations [51] to reduce redundant computations.

4.5 Shift reduction approaches for racetrack memories

To read out the data, the racetrack must be shifted until the targeted data aligns with the access points [3, 65] and the amount of the shifts reflects the required access latency (so as the energy consumption). Hence, minimizing the shifts is vital to the resource efficiency when utilizing racetrack. Several approaches have been proposed to reduce the number of shift operations in racetrack memories, such as runtime data swapping [50], preshifting [50], intelligent instruction [38], and SRAM buffering [62]. Chen et al. in [9] present a heuristic that appends data objects based on adjacency information for data placement. Khan et al. formulate the data placement problem with an ILP and enhance the previous heuristic [27]. However, as generalized solutions, both works do not exploit the specification of access patterns, which can be featured by ML models, e.g., dependencies between tree nodes [22].

5 SUMMARY

Several techniques have been already developed to address endurance issues, emulations, WCET analyses etc. for emerging non-volatile memories. Although several real-time non-volatile storage techniques have been proposed, their insights cannot exploit the characteristics of byte-addressable NVM directly. Along with the emerging designs of using NVM as the main memory, the timing predictability under such disruptive characteristics is not fully explored. How to mitigate the stress of the NVM issues under specific timing predictability requirement(s) remains as an open problem.

From the CIM perspective, NVMs are particularly interesting due to their non-volatility and promising physical attributes that can enable analog computing. So, they can be leveraged to realize non-volatile processors. However, due to the fact that some technologies are particularly better for some operations than others, general purpose CIM systems have to be heterogeneous, offering a huge design space for exploration. To facilitate a rapid DSE, high-level evaluation frameworks are needed that can assess various design trade-offs by analyzing different system configurations and their performance characteristics. In addition to the slow write operations, analog nature of computation, and device non-idealities, reliability in some NVM technologies is also a major concern and has to be considered a first-class optimization metric.

Last but not least, machine learning is a trending killer application for NVMs. But it is commonly studied from a high-level perspective with limited consideration of the underlying hardware, even though it is well-known memory-hungry. To optimize the performance and energy consumption, the execution properties featured by NVM should be taken into account, such as access-dependent latency on racetrack memories, while deploying the machine learning models. Moreover, architecture-induced reliability issues shall also be addressed, even at the training phase. With increasing demand and emerging memory technologies, how to fully utilize the advantages of NVM is foreseen one key to make machine learning further resource-efficient.

ACKNOWLEDGMENTS

This work is supported in parts by the German Research Foundation (DFG) as part of the priority program "SPP 2377: Disruptive Memory Technologies" under projects: *Reconfigurable Architectures and Real-Time Systems Co-Design for Non-Volatile Memory (ARTS-NVM)* and *Co-Design of Persistent, Energy-efficient and High-speed Embedded Processor Systems with Hybrid Volatility Memory Organisation (HYPNOS, project number 502213043)* and *Balancing computations in in-memory nonvolatile heterogeneous systems (HetCIM, project number 502388442)*. We would like to thank DFG (Project Number: 405422836). We would like to acknowledge the Ministry of Science and Technology of Taiwan (MOST-111-2923-E-002-014-MY3, MOST-111-2218-E-002-026, MOST-109-2221-E-002-147-MY3, MOST-109-2221-E-001-012-MY3, NSTC-112-2218-E-002-025-MBK), Macronix Inc., Taiwan (111HT912003) for their support.

REFERENCES

[1] J. Ambrosi, et al. 2018. Hardware-Software Co-Design for an Analog-Digital Accelerator for Machine Learning. In *Int. Conf. on Rebooting Computing*. 1–13.
[2] A. Ankit, et al. 2019. PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference. In *Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 715–731.
[3] R. Bläsing, et al. 2020. Magnetic Racetrack Memory: From Physics to the Cusp of Applications Within a Decade. *Proc. IEEE* 108, 8 (2020), 1303–1321.
[4] S. Buschjäger, et al. 2021. Margin-Maximization in Binarized Neural Networks for Optimizing Bit Error Tolerance. In *Design, Automation & Test in Europe Conf. (DATE)*. 673–678.
[5] S. Buschjager, et al. 2018. Realization of Random Forest for Real-Time Evaluation through Tree Framing. In *Int. Conf. on Data Mining (ICDM)*. 19–28.
[6] S. Buschjäger. 2018. Decision Tree and Random Forest Implementations for Fast Filtering of Sensor Data. *IEEE Transactions on Circuits and Systems I: Regular Papers* 65, 1 (2018), 209–222.
[7] C.-H. Chen, et al. 2012. Age-based PCM wear leveling with nearly zero search cost. In *Design Automation Conf. (DAC)*. 453–458.

[8] P.-Y. Chen, et al. 2018. NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 37, 12 (2018), 3067–3080.

[9] X. Chen, et al. 2016. Efficient Data Placement for Improving Data Access Performance on Domain-Wall Memory. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 24, 10 (2016), 3094–3104.

[10] Y.-S. Chen, et al. 2023. DTC: A Drift-Tolerant Coding to Improve the Performance and Energy Efficiency of Multi-Level-Cell Phase-Change Memory. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2023).

[11] H.-Y. Cheng, et al. 2021. Future Computing Platform Design: A Cross-Layer Design Approach. In *Design, Automation & Test in Europe Conf. (DATE)*. 312–317.

[12] S. Cho. 2009. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Int. Symp. on Microarchitecture (MICRO)*. 347–357.

[13] G. Dhiman, et al. 2009. PDRAM: A hybrid PRAM and DRAM main memory system. In *Design Automation Conf. (DAC)*. 664–669.

[14] A. Drebes, et al. 2020. TC-CIM: Empowering Tensor Comprehensions for Computing-In-Memory. In *Int. Workshop on Polyhedral Compilation Techniques*.

[15] S. R. Dulloor, et al. 2014. System software for persistent memory. In *European Conf. on Computer Systems (EuroSys)*. 1–15.

[16] A. P. Ferreira, et al. 2010. Increasing PCM main memory lifetime. In *Design, Automation & Test in Europe Conf. (DATE)*. 914–919.

[17] D. Fujiki, et al. 2018. In-Memory Data Parallel Processor. In *Int. Conf. on Arch. Support for Programming Languages and Operating Systems (ASPLOS)*. 1–14.

[18] V. Gogte, et al. 2019. Software Wear Management for Persistent Memories. In *Conf. on File and Storage Technologies (FAST)*. 45–63.

[19] J. Gómez-Luna, et al. 2022. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access* 10 (2022), 52565–52608.

[20] C. Hakert, et al. 2022. Software-Managed Read and Write Wear-Leveling for Non-Volatile Main Memory. *ACM Trans. on Embedded Computing Systems (TECS)* 21, 1 (2022), 1–24.

[21] C. Hakert, et al. 2021. BLOwing Trees to the Ground: Layout Optimization of Decision Trees on Racetrack Memory. In *Design Automation Conf. (DAC)*. 1111–1116.

[22] C. Hakert, et al. 2023. ROLLED: Racetrack Memory Optimized Linear Layout and Efficient Decomposition of Decision Trees. *IEEE Trans. on Computers (TC)* 72, 5 (2023), 1488–1502.

[23] D. Hernandez. 2020. Measuring the Algorithmic Efficiency of Neural Networks. arXiv:2005.04305 [cs.LG]

[24] H. Ji, et al. 2018. ReCom: An efficient resistive accelerator for compressed deep neural networks. In *Design, Automation & Test in Europe Conf. (DATE)*. 237–240.

[25] S. Kargar. 2022. Challenges and future directions for energy, latency, and lifetime improvements in NVMs. *Distributed and Parallel Databases* (2022), 1–27.

[26] A. A. Khan, et al. 2023. CINM (Cinnamon): A Compilation Infrastructure for Heterogeneous Compute In-Memory and Compute Near-Memory Paradigms. *arXiv preprint arXiv:2301.07486* (2023).

[27] A. A. Khan, et al. 2019. ShiftsReduce: Minimizing Shifts in Racetrack Memory 4.0. *ACM Transactions on Architecture and Code Optimization (TACO)* 16, 4, Article 56 (2019), 23 pages.

[28] C. Lammie, et al. 2022. MemTorch: An Open-source Simulation Framework for Memristive Deep Learning Systems. *Neurocomputing* 485 (2022), 124–133.

[29] C. Lattner, et al. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *Intl. Symp. on Code Generation and Optimization (CGO)*. 2–14.

[30] M. K. F. Lee, et al. 2019. A System-Level Simulator for RRAM-Based Neuromorphic Computing Chips. *ACM Trans. on Architecture and Code Optimization (TACO)* 15, 4, Article 64 (2019), 24 pages.

[31] B. Li, et al. 2020. HitM: High-Throughput ReRAM-based PIM for Multi-Modal Neural Networks. In *Int. Conf. On Computer Aided Design (ICCAD)*. 1–7.

[32] Q. Li, et al. 2013. Compiler directed write-mode selection for high performance low power volatile PCM. In *Languages, Compilers and Tools for Embedded Systems (LCTES)*. 101–110.

[33] J. Lin, et al. 2019. Learning the Sparsity for ReRAM: Mapping and Pruning Sparse Neural Network for ReRAM Based Accelerator. In *Asia and South Pacific Design Automation Conf. (ASP-DAC)*. 639–644.

[34] M.-Y. Lin, et al. 2018. DL-RSIM: A Simulation Framework to Enable Reliable ReRAM-based Accelerators for Deep Learning. In *Int. Conf. on Computer-Aided Design (ICCAD)*. 1–8.

[35] W.-T. Lin, et al. 2022. DL-RSIM: A Reliability and Deployment Strategy Simulation Framework for ReRAM-Based CNN Accelerators. *ACM Trans. on Embedded Computing Systems (TECS)* 21, 3, Article 24 (2022), 29 pages.

[36] Y. Liu, et al. 2015. Ambient energy harvesting nonvolatile processors: From circuit to system. In *Design Automation Conf. (DAC)*. 1–6.

[37] K. Ma, et al. 2015. Nonvolatile Processor Architecture Exploration for Energy-Harvesting Applications. *IEEE Micro* 35, 5 (2015), 32–40.

[38] J. Multanen, et al. 2019. SHRIMP: Efficient Instruction Delivery with Domain Wall Memory. In *Int. Symp. on Low Power Electronics and Design (ISLPED)*. 1–6.

[39] M. Ni, et al. 2020. A Novel Prefetching Scheme for Non-Volatile Cache in the AIoT Processor. In *Int. Conf. on Universal Village (UV)*. 1–7.

[40] C. Pan, et al. 2017. Exploiting multiple write modes of nonvolatile main memory in embedded systems. *ACM Trans. on Embedded Computing Systems (TECS)* 16, 4 (2017), 1–26.

[41] G. Pedretti, et al. 2021. Tree-based machine learning performed in-memory with memristive analog CAM. *Nature communications* 12, 1 (October 2021), 5806.

[42] M. K. Qureshi, et al. 2009. Enhancing Lifetime and Security of PCM-based Main Memory with Start-gap Wear Leveling. In *Int. Symp. on Microarch.* 14–23.

[43] M. K. Qureshi, et al. 2009. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Int. Symp. on Computer Arch.* 24–33.

[44] A. Sebastian, et al. 2020. Memory devices and applications for in-memory computing. *Nature nanotechnology* 15, 7 (2020), 529–544.

[45] A. Shafiee, et al. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *Int. Symp. on Computer Arch.* 14–26.

[46] A. Siemieniuk, et al. 2021. OCC: An Automated End-to-End Machine Learning Optimizing Compiler for Computing-In-Memory. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 41, 6 (2021), 1674–1686.

[47] G. Singh, et al. 2018. A review of near-memory computing architectures: Opportunities and challenges. In *Euromicro Conf. on Digital System Design*. 608–617.

[48] P.-L. A. E. Sixdenier, et al. 2023. Seque: Lean and Energy-aware Data Management for IoT Gateways. In *Int. Conf. on Edge Computing and Communications*.

[49] D. Stutz, et al. 2021. Bit Error Robustness for Energy-Efficient DNN Accelerators. In *Machine Learning and Systems (MLSys)*. 569–598.

[50] Z. Sun, et al. 2013. Cross-layer racetrack memory design for ultra high density and low power consumption. In *Design Automation Conf. (DAC)*. 1–6.

[51] C.-Y. Tsai, et al. 2021. RePIM: Joint Exploitation of Activation and Weight Repetitions for In-ReRAM DNN Acceleration. In *Design Automation Conf. (DAC)*. 589–594.

[52] Y.-T. Tsou, et al. 2022. This is SPATEM! A Spatial-Temporal Optimization Framework for Efficient Inference on ReRAM-based CNN Accelerator. In *Asia and South Pacific Design Automation Conf. (ASP-DAC)*. 702–707.

[53] Upmem. 2022. UPMEM Processing In-Memory (PIM): Ultra-efficient acceleration for data-intensive applications. In *2022 UPMEM PIM Tech paper v2.7*. 1–22.

[54] K. Vadivel, et al. 2020. TDO-CIM: transparent detection and offloading for computation in-memory. In *Design, Autom. & Test in Europe Conf.* 1602–1605.

[55] C. Wang, et al. 2020. Crab-Tree: A Crash Recoverable B+-Tree Variant for Persistent Memory with ARMv8 Architecture. *ACM Trans. on Embedded Computing Systems (TECS)* 19, 5, Article 35 (2020), 26 pages.

[56] W.-C. Wang, et al. 2019. Achieving lossless accuracy with lossy programming for efficient neural-network training on NVM-based systems. *ACM Trans. on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–22.

[57] Z. Wang, et al. 2014. Adaptive placement and migration policy for an STT-RAM-based hybrid cache. In *Int. Symp. on High Perf. Comp. Arch. (HPCA)*. 13–24.

[58] Z. Wang, et al. 2013. WADE: Writeback-Aware Dynamic Cache Management for NVM-Based Main Memory System. *ACM Trans. on Architecture and Code Optimization (TACO)* 10, 4, Article 51 (dec 2013), 21 pages.

[59] L. Xia, et al. 2016. MNSIM: Simulation platform for memristor-based neuromorphic computing system. In *Design, Automation & Test in Europe Conf. (DATE)*. 469–474.

[60] L. Xie, et al. 2017. Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing. In *Computer Society Symp. on VLSI (ISVLSI)*. 176–181.

[61] M. Xie, et al. 2019. A Novel STT-RAM-Based Hybrid Cache for Intermittently Powered Processors in IoT Devices. *IEEE Micro* 39, 1 (2019), 24–32.

[62] R. Xu, et al. 2023. Optimizing Data Placement for Hybrid SRAM+Racetrack Memory SPM in Embedded Systems. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 42, 3 (2023), 847–859.

[63] Y. Xu, et al. 2017. Energy-Efficient Cache Management for NVM-Based IoT Systems. In *Int. Symp. on Parallel and Distributed Processing with Applications and Int. Conf. on Ubiquitous Computing and Communications (ISPA/IUCC)*. 491–493.

[64] T.-H. Yang, et al. 2019. Sparse ReRAM Engine: Joint Exploration of Activation and Weight Sparsity in Compressed Neural Networks. In *Int. Symp. on Computer Architecture (ISCA)*. 236–249.

[65] Y.-H. Yang, et al. 2022. Evolving Skyrmion Racetrack Memory as Energy-Efficient Last-Level Cache Devices. In *Int. Symp. on Low Power Elect. and Design*. 6 pages.

[66] M. Yayla, et al. 2022. FeFET-Based Binarized Neural Networks Under Temperature-Dependent Bit Errors. *IEEE Trans. on Computers (TC)* 71, 7 (2022), 1681–1695.

[67] M. Zhang, et al. 2019. Quick-and-Dirty: An Architecture for High-Performance Temporary Short Writes in MLC PCM. *IEEE Trans. Comp.* 68, 9 (2019), 1365–1375.

[68] Y. Zhang, et al. 2020. PattPIM: A Practical ReRAM-Based DNN Accelerator by Reusing Weight Pattern Repetitions. In *Design Automation Conf. (DAC)*. 1–6.

[69] L. Zhao, et al. 2019. RFAcc: A 3D ReRAM Associative Array Based Random Forest Accelerator. In *Int. Conf. on Supercomputing*. 473–483.

[70] P. Zhou, et al. 2009. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. In *Int. Symp. on Computer Arch. (ISCA)*. 14–23.

Received June 2023