Future Computing Platform Design: A Cross-Layer Design Approach

Hsiang-Yun Cheng*, Chun-Feng Wu†§, Christian Hakert‡, Kuan-Hsun Chen‡
Yuan-Hao Chang†, Jian-Jia Chen‡, Chia-Lin Yang§, Tei-Wei Kuo§¶
*Research Center for Information Technology Innovation, Academia Sinica, Taiwan
†Institute of Information Science, Academia Sinica, Taiwan
‡Department of Computer Science, Technische Universität Dortmund, Germany
§Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
¶College of Engineering, City University of Hong Kong, Hong Kong
E-mail: johnson@iis.sinica.edu.tw, jian-jia.chen@cs.uni-dortmund.de, {yangc, ktw}@csie.ntu.edu.tw
(Special Session)

***Abstract*—Future computing platforms are facing a paradigm shift with the emerging resistive memory technologies. First, they offer fast memory accesses and data persistence in a single large-capacity device deployed on the memory bus, blurring the boundary between memory and storage. Second, they enable computing-in-memory for neuromorphic computing to mitigate costly data movements. Due to the non-ideality of these resistive memory devices at the moment, we envision that cross-layer design is essential to bring such a system into practice. In this paper, we showcase a few examples to demonstrate how cross-layer design can be developed to fully exploit the potential of resistive memories and accelerate its adoption for future computing platforms.**

I. Introduction

In recent years, data analytics applications that must process increasingly large volumes of data, such as deep learning, graph analytics, etc, have become more and more popular. These big data applications demand large memory capacity and efficient data accesses. Unfortunately, mainstream computing systems are not designed to meet their needs, as DRAM has limited scalability [1] and the intensive data movements between CPU and memory incur serious performance and energy penalties [2]. This forces system architects to fundamentally rethink how to design future computing platforms.

To meet the demands of big data, we are seeing a disruptive change in the computing platform design: First, we need a new tier of memory (i.e., storage class memory) to blur the boundary between memory and storage to provide large-capacity persistent memory directly on the memory bus. Second, instead of moving data across the memory channels to distant compute units, bringing computation closer to data to build a memory-centric design can eliminate costly data movements. As emerging resistive memories, such as phase change memory (PCM) and resistive RAM (ReRAM), offer superior density, non-volatile property, and computing-in-memory capability, they are suitable to serve as storage class memory and enable the opportunity for a paradigm shift from the contemporary processor-centric design towards the revolutionary memory-centric design.

Despite the solution enabled by emerging memory technologies is promising, bringing such a system into practice remains challenging due to the drawbacks of current resistive memory devices. The limited write endurance and asymmetric read-write latency/energy constraint the practicality of storage class memory, while the inherent resistance variation in resistive memory cells induces unreliable computation and hinders the realization of computing-in-memory. One promising solution to overcome these challenges is cross-layer design. Different from device/circuit oriented solutions that rely on the advancement of technologies, cross-layer approaches can better deal with the drawbacks coming from the imperfect devices through the assistance of application, system software, and architecture designs. For instance, for some specific types of application data, such as the data that do not need non-volatile guarantee [3] or the frequently updated training data of deep neural networks (DNNs) [4], we can relax the retention time of resistive memory devices to reduce write latency of storage class memory. Regarding the computation reliability issue of computing-in-memory, we can avoid serious accuracy degradation by jointly considering the underlying device property and the error-tolerance characteristics of the target DNN to design software data encoding scheme [5] or adapt architecture-level configurations [5], [6].

In this paper, we present the device features of resistive memories, discuss their impacts on system architecture, and introduce several examples to illustrate how cross-layer approaches can be leveraged to tackle the design challenges incurred by resistive memory devices. For storage class memory, we elaborate how to facilitate wear-leveling through the joint assistance from common existing hardware and different levels of system software. We also show how to exploit the special characteristics of DNNs and co-design the underlying hardware mechanisms to suppress write hot-spots and improve performance. To put computing-in-memory into practice, we demonstrate how to conduct cross-layer design space exploration through a simulation framework and how to enable reliable DNN inference through device-architecture and software-hardware co-design. Our purpose is to attract attention from relevant researchers to notice how cross-layer design could

978-3-9819263-5-4/DATE21/©2021 EDAA

be helpful to exploit the potential of resistive memory and accelerate its adoption for future computing platforms.

II. Resistive Memory Devices

Resistive memories generally refer to any memory technology that stores and represents data using varying cell resistance. Unlike traditional DRAM whose states are determined by the amount of charge on a capacitor, resistive memories are amenable to scaling, non-volatile, and there is nearly no cell leakage. Among various resistive memory technologies that adopt different materials and switching physics, Phase Change Memory (PCM) and Resistive RAM (ReRAM) have superior density and are being actively pursued as DRAM replacements.

A. Phase Change Memory (PCM)

A PCM storage element consists of two electrodes separated by a resistive heater and a chalcogenide material (e.g., $Ge_2Sb_2Te_5$) [7], as shown in Figure 1(a). Through injecting current into the resistor junction and heating the chalcogenide, a PCM cell can be switched between a crystalline phase, representing low resistance state (LRS), and an amorphous phase, representing high resistance state (HRS). The programmed state is determined by the amplitude and width of the injected current pulse. Applying a high-power but short-duration current pulse RESET the chalcogenide to HRS, while injecting a moderate-power but long-duration current pulse gradually cooling the chalcogenide and SET the cell to LRS. Write performance is determined by SET latency, whereas write power is dictated by RESET energy. Iterative write-and-verify scheme [8] can be applied to program PCM cells to multiple intermediate levels, as the resistance contrast between HRS and LRS is large.

B. Resistive RAM (ReRAM)

A typical ReRAM cell is composed of a metal-oxide layer (e.g., HfO_x) sandwiched between two metal layers of electrodes, as shown in Figure 1(b). In order to switch the resistance state, an external voltage is applied across the ReRAM cell. A conductive filament (CF) made out of oxygen vacancies is either formed (SET) or ruptured (RESET) depending on the voltage polarity [9]. Due to the stochastic nature of the generation and rupture of oxygen vacancies, recent studies show that the resistance distributions of ReRAM cells follow the lognormal distribution [10], [11]. The size of the CFs is directly related to the amount of cell current. By using the iterative write-and-verify scheme [12] to change the strength of the CFs, a ReRAM cell can be programmed to intermediate levels between HRS and LRS. Generally, a ReRAM cell is called a single-level-cell (SLC) if it can only be programmed to two resistence levels to represent the situation of single data bit. A multi-level-cell (MLC) ReRAM can be programmed to more resistence levels for representing multiple data bits.

III. Impacts on System Architecture

Resistive memories are likely to bring revolutionary changes in future computing paradigm. In the following, we show how the promising features of resistive memories can be leveraged to enable the deployment of storage class memory (SCM) and computing-in-memory (CIM). For each individual application domain, we discuss the design challenges introduced by the non-idealities in current resistive memory devices.

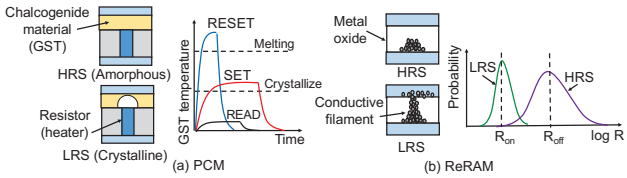

Fig. 1. The schematic view of (a) PCM cell and its temperature-time relationship; (b) ReRAM cell and its resistance distribution.

A. Storage Class Memory (SCM)

Resistive memories have been shown to provide higher density and comparable read performance as conventional DRAM. Thus, many recent studies placed resistive memories on the memory bus to serve as a large working memory [7], [13], and scalable server-grade resistive memory DIMMs have become available with the release of Intel Optane DIMM [14]. In addition to offering large capacity, the non-volatile feature of resistive memories makes it possible to persistently retain data structures in memory without writing them to traditional storage. With its persistent feature and DRAM-comparable performance, resistive memories blur the boundary between memory and storage, and thus envision a new tier of memory (i.e., storage class memory) that can offer significant performance advantages for data-intensive applications.

Despite the fact that exploiting resistive memories as SCM is promising, several design challenges need to be addressed:

Limited write endurance. The physical properties of resistive memories limit the number of reliable writes to each cell. For example, a PCM cell can only endure about 10^6 to 10^9 writes before an error occurs [15], [16], because the thermal expansion and contraction during current injection degrade the electrode storage contact. Similar to PCM, ReRAM also has an endurance issue due to the gradual resistance change over the write cycles [17]. Each ReRAM cell normally tolerates about 10^{10} writes [9], better than PCM, but some weak cells last for only 10^5 to 10^6 writes. The limited write endurance of resistive memories hurts the lifetime of SCM, especially if the writes of the running applications are not uniformly distributed to different cells and the heavily written cells fail earlier than expected. Thus, write reduction [7], [18], wear-leveling [7], [19], and error correction techniques [20] are needed to prolong the lifetime of SCM.

Asymmetric read-write latency and energy. In resistive memories, the transition between HRS and LRS is triggered by applying high-power electrical voltage/current pulses with long duration. Thus, compared to conventional DRAM, writes in resistive memories are slower and consume higher energy. For example, PCM writes require energy-intensive current injection, so the write latency/energy of PCM is an order of magnitude higher than its read latency/energy [7]. The resistance drift of PCM cells [3] and the iterative write-and-verify scheme [8] used to program multi-level cells further

exacerbate the problem. To tackle the challenge of asymmetric read-write latency/energy, prior studies have proposed some write reduction [7], [18], data encoding [8], [13], and scheduling techniques [13], [21]. Another possible solution is to relax the retention time to reduce write latency when SCM is serving working memory requests that do not need non-volatility guarantee [3].

B. Computing-in-Memory (CIM)

Resistive memories are able to perform arithmetic operations beyond data storage. Recent studies have demonstrated that resistive memory crossbar arrays can be utilized to efficiently perform matrix-vector multiplications [22]–[24]. As shown in Figure 2(a), if input voltage V_i is applied to each wordline, based on Kirchoff's Law, the accumulated current on bitline j is equal to the sum-of-products of the input voltage and cell conductance ($I_j = \sum_i V_i/R_{ij} = \sum_i V_i \times G_{ij}$). Ideally, with the crossbar structure, matrix-vector multiplication can be done in one constant time step. Such computing-in-memory capability provides a promising solution to accelerate deep learning inference [23], [24], since the convolution and fully-connected layers of DNNs involve a significant number of matrix-vector multiplications. For example, as shown in Figure 2(a), by storing filter weights as the conductance of resistive memory cells and converting input feature maps into input voltage signals via digital-to-analog converters (DACs), the output feature maps can be obtained by reading out the accumulated currents on the bitlines through the analog-to-digital converters (ADCs).

Reliability issue. While it is promising to exploit the CIM capability of resistive memories to accelerate deep learning, the design challenges introduced by the non-ideal devices and peripheral circuits need to be carefully examined to achieve robust computations. For example, in ReRAM cells, the stochastic nature of the conductive filament generation causes resistance variations. This stochastic variation of ReRAM resistance is likely to degrade the sensing accuracy of output currents, as shown in Figure 2(b). The accuracy degradation is further exacerbated when a large number of wordlines are activated concurrently, as more per-cell current deviations are accumulated and it becomes harder to differentiate between neighboring states with a large overlapped region in the output current distribution. The design of ADC, such as its bit-resolution and sensing method, also affects the error rate.

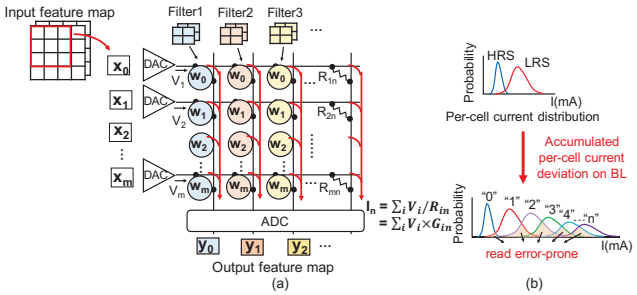

Fig. 2. (a) Resistive memory based matrix-vector multiplication for DNN inference; (b) Reliability issue caused by non-ideal ReRAM cells.

IV. Cross-Layer Design

The unique features of resistive memories enable an innovative change in computing systems, yet the non-ideal device properties hinder its realization. One promising solution to tackle this challenge is cross-layer design. Through leveraging the distinctive characteristics of various applications and the assistance from system software and architecture, it is possible to fully exploit the benefits of resistive memories while mitigating its weakness. In this section, we use several examples to demonstrate how cross-layer approaches can be employed to improve the reliability and performance of SCM and CIM.

A. Cross-Layer Design for SCM

To mitigate the endurance problem and reduce the write performance penalty for SCM, we propose several cross-layer mechanisms. First, we illustrate how to extend the lifetime of SCM through the joint assistance from common existing hardware and different levels of system software. Considering the inherently limited endurance of resistive memories, one can adopt the memory management unit (MMU) together with system software to redirect the memory accesses and achieve wear-leveling at the virtual page level. In addition, a finer wear-leveling within each page can be achieved by system software, which offsets the stack via the application binary interface (ABI) level. Second, we demonstrate how to leverage the characteristics of different neural network (NN) layers and co-design the underlying hardware mechanisms to alleviate write overheads. One example is to design CPU cache pinning strategy based on the write-intensity of different NN layers, so that write-hot data can be locked in the cache to reduce SCM writes. Another example is to improve SCM performance through data-aware programming scheme based on the error-tolerance characteristics of different NN layers. To be specific, for the NN training data that are updated frequently, we can relax the retention time to reduce write latency. The details of these cross-layer design examples are explained in the following.

1) Software Based Wear-leveling Across Layers: Since the limited endurance is crucial in the deployment of resistive memories, several wear-leveling strategies have been proposed in the literature. Despite many techniques proposing to tackle the issue within the memory hardware, a few approaches intend to maintain the memory wear-out in software by, either running wear-leveling algorithms as part of the system software, or directly as part of the application. These approaches either utilize information (e.g. the current memory aging) which are provided by special hardware or they even overcome the need for this information by software-based approximations. In order to modify the memory write accesses, software algorithms can take action at various layers. On the device driver level (i.e. MMU and virtual memory), fully transparent access redirection can be employed. On the application level, recompilation and automatic code rewriting can redirect memory accesses specific for single applications. On the application binary interface (ABI) level, memory contents can be replaced before application functions are called. In the following, we present a

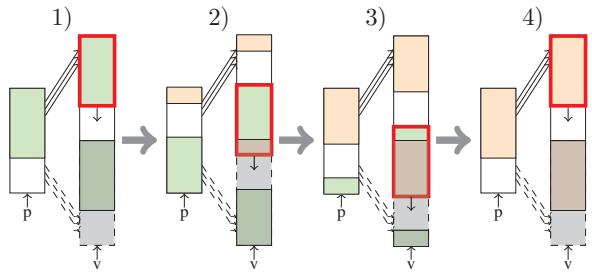

Fig. 3. Shadow stack maintenance. The currently used stack is marked in red.

wear-leveling approach, utilizing the device driver and the ABI levels.

For aging-aware coarse-grained wear-leveling we provide a operating system service, which interacts with common existing hardware [25]. This solution utilizes the MMU and modifies the mapping of virtual to physical memory pages. By doing so, the physical location of memory contents can be exchanged during runtime so that write accesses can be redirected. Provided with the current memory aging from another subsystem, the wear-leveling algorithm keeps an estimated age for every physical memory page. On an user-defined frequency, the algorithm identifies the "hottest" and the "coldest" page and exchanges the mapped virtual pages of both of them.

Since the above strategy only operates on the granularity of virtual memory pages (usually 4 kB), it might happen that only a few bytes within a page are intensively written and therefore a finer wear-leveling within pages is required. To resolve this, we propose a maintenance algorithm, which relocates the stack, which is the main cause for not properly wear-leveled memory pages, by offsets of a few bytes and keeps the view of applications on the stack consistent [26]. In order to achieve this, we copy stack contents and adjust software and hardware stack pointers in order to maintain the ABI semantics and require no application cooperation. We further utilize the hardware level and employ a *shadow* stack.

Figure 3 illustrates the working principle of the shadow stack maintenance algorithm. The physical pages of the stack (p) are mapped two times to consecutive pages in the virtual address space (v). We call the upper virtual mapping the real and the lower virtual mapping the shadow mapping. When the application uses the stack in the virtual address space, the maintenance algorithm relocates the stack by adjusting stack pointers by small positives offsets and copying the entire stack to the new location. At a certain moment, the used stack crosses the boundary of the the virtual memory pages from step 1) to 2). Due to the shadow mapping, the physical layout of the stack performs an automatic wraparound. The movement can be repeated until the original physical layout from 1) is established in 4) again. Repeating this procedure on a fixed frequency leads to a circular movement of the entire program stack, so that heavily written elements are moved through a larger memory space and the write accesses are distributed equally.

In order to evaluate the proposed strategies without limiting on any specific resistive memory, an online runtime system can be used, which adopts performance counters and configurable memory permissions (hardware level) to approximate the amount of write accesses to certain memory locations [25]. The performance counters are used to count the total amount of memory write accesses in the entire system and are configured to trigger an interrupt when exceeding a certain threshold. The evaluation highlights that in the best case, the above software-based wear-leveling approaches can achieve a 78.43% wear-leveled memory. This makes an improvement of $\approx 900\times$ in the memory lifetime compared to a basic setup without any wear-leveling mechanisms.

2) DNN characteristics aware hardware design: Neural networks are one of the most popular applications in recent years. To accommodate its large memory demands with lower unit cost and standby power, SCM technologies (e.g., PCM in the following cases) are a promising candidate to replace the role of DRAM. However, compared with DRAM, SCM devices usually suffer from a limited lifetime and lower access performance. As reported by several previous works [4], [27], instead of adopting a general management approach (e.g., start-gap [19] or age-based wear leveling [28]), a more effective way to tackle both lifetime and performance issues is to manage the SCM devices by being aware of the behaviors of NNs.

There are two major phases during performing CNN inferences, that is convolutional and fully-connected phases. Due to different design objectives, the convolutional phases, which aim to extract low-level to high-level features, may cause more intensive memory write accesses on same specific memory locations than that of the fully-connected phases which are designed to make the inference decision. This phenomenon is called write hot-spot effect [27]. To suppress the write hot-spot effect, a self-bouncing CPU cache pinning strategy is proposed to capture and pin (or lock) the write-hot data in the CPU cache by being aware of CNN access behaviors. It is worth noted that, without increasing programmer overheads or overhauling software tools, this work does not need any programmer hints provided by the user library or even the compiler. This strategy periodically monitors the numbers of CPU write cache misses and dynamically adjusts the reserved amounts of CPU cache for cache line pinning. With this strategy, more CPU cache space will be used for pinning the write-hot cache line during running convolutional phases so as to suppress the write hot-spot effect. If the strategy judges that the system enters the fully-connected phases, the CPU cache space will be released for general-purposed usage so as to avoid performance degradation.

Compared with the inference operation, training neural network is more time-consuming, because a training operation iterates both forward and backward NN processes several times until reaching the saturation point. To accelerate the performance of NN training on PCM devices, data-aware programming scheme [4] is proposed to co-design the two observed NN training behaviors (i.e., the bit-change rates and data-update duration), and the special write commands provided by the PCM devices. To make the training result smoothly converge to the saturation point, model weights and biases will be updated by using the manner of gradient updates, which finely tune the model weights and biases. In this way, the bit

change rates of the positions close to the most significant bit (MSB) are much slower than that close to the least significant bit (LSB). The reason is that, model weights and biases are encoded by IEEE Standard for Floating-Point Arithmetic (IEEE-754) where bit positions close to MSB represent the exponent part which is hardly changed especially when the data value is updated slightly. On the other hand, the update duration of model weights and biases in different NN layers is different. Generally, model weights and biases belonging to the rearmost NN layers have a smaller update duration compared with those belonging to the foremost NN layers because a backward process is always executed right after the completion of a forward process. The data-aware programming scheme introduced Lossy-SET and Precise-SET operations to program the PCM cells by considering the trade-off between programming performance and data endurance. Specifically, Precise-SET is applied to program data bits with low bit-change rates for ensuring data correctness and Lossy-SET is used to program data bits with high bit-change rates so as to speed up the performance. Moreover, data bits programmed by Lossy-SET shall be re-programmed by being aware of the data-update duration and the error-tolerance characteristics of the target DNN so as to avoid serious data corruptions.

B. Cross-Layer Design for CIM

Cross-layer design approaches can also be leveraged to tackle the computation reliability issue when exploiting the CIM capability of resistive memories to accelerate deep learning. In this section, we illustrate two examples. First, we show how a simulation framework can be used to facilitate device-architecture co-design for reliable DNN inference. As the inference accuracy of a ReRAM-based DNN accelerator is jointly affected by impact factors across different system levels (i.e., from device properties, architecture design, to DNN model characteristics), we can conduct cross-layer design space exploration via a reliability simulation framework to find the best setting that guarantees satisfactory inference accuracy. Second, to enhance DNN reliability, a software-hardware co-design strategy (named as adaptive data manipulation strategy) is introduced to encode and place DNN parameters on a ReRAM-based DNN accelerator by being ware of the IEEE-754 data representation properties and the accelerator architecture. In the following, we explain these cross-layer design approaches in more details.

1) Simulation framework to facilitate device-architecture co-design for DNN: Considering the device characteristics of resistive memories, it is important to comprehensively explore the design space from device to architecture level to guarantee reliable CIM computations for DNN inference. For example, due to the intrinsic resistance variation in ReRAM cells, it is hard for ADCs to read out the correct values when a large number of wordlines are turned on concurrently. As a result, a practical ReRAM-based DNN accelerator only activates a smaller section (OU) of a crossbar array in a single cycle [29]. The setting of the OU size depends on the cell properties of the resistive memory device, the DNN model characteristics, and the target inference accuracy. Thus, a simulation framework, DL-RSIM [6], is proposed to facilitate design space exploration for developing reliable resistive memory based DNN accelerators.

DL-RSIM is composed of two modules, as shown in Figure 4, and can be incorporated with any DNN models implemented by TensorFlow. The Resistive Memory Error Analytical Module models the sensing error rates of the accumulated current per bitline. It first takes a set of device configurations, such as the resistance mean and deviation of each cell state, as inputs and uses Monte Carlo sampling method to model the accumulated current distribution on a bitline. It then estimates the error rates of each sum-of-products results based on the user-specified ADC bit-resolution and sensing method. The estimated sum-of-products error rates are passed to the TensorFlow-based Inference Accuracy Simulation Module, which models the impact of sum-of-products sensing errors on the inference accuracy of the target DNN.

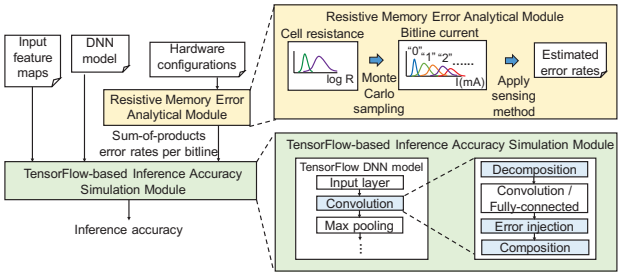

Fig. 4. Simulation framework of DL-RSIM.

Here we use an example to illustrate how to use DL-RSIM to perform device-architecture co-design: finding a good OU size for the selected resistive memory device and the target DNN model to achieve satisfactory inference accuracy. As shown in Figure 5, the accuracy is degraded when OU height (i.e., number of concurrently activated wordlines) increases. Advances in device technology (i.e., increasing *R-ratio* and reducing resistance deviation) can help to improve the inference accuracy by shrinking the overlap with neighboring states in the accumulated current distribution to reduce ADC sensing errors. With 3x improvement in *R-ratio* and resistance deviation, a simple three-layer NN model can achieve satisfactory accuracy on MNIST dataset even when the height of an OU is 128, while the height of an OU needs to be less than 16 to avoid accuracy degradation for the complex CaffeNet testing on ImageNet dataset.

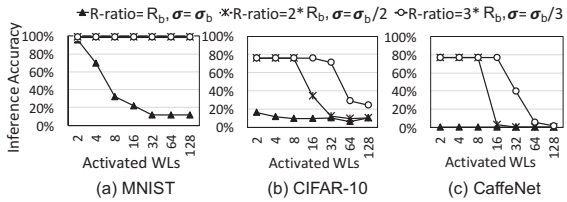

Fig. 5. Inference accuracy of (a) MNIST, (b) CIFAR-10, and (c) CaffeNet when various number of wordlines (WLs) are activated concurrently, with three different types of ReRAM cells [30]. R_b and σ_b are the *R-ratio* and resistance deviation of WO_x ReRAM [10].

2) Software-hardware co-design for DNN reliability enhancement: Due to the overlapping variation error, ReRAM-based DNN accelerators suffer from the reliability issue if more wordlines shall be activated simultaneously for achieving better performance. The overlapping variation error is that, because of the CF shape varies from from cycle to cycle [31], [32], the summed currents in the end of each bitline are difficult to be converted to the correct digital value especially when more wordlines are activated. To tackle the reliability issue, a software-hardware co-designing adaptive data manipulation strategy [5] is proposed to encode both inputs and weights for alleviating the overlapping variation error by exploiting the properties of IEEE-754 data representation and the accelerator architecture. The proposed strategy is mainly composed of a weight-rounding design (WRD) and an adaptive input sub-cycling design (AISD). To be specific, WRD aims to reduce the number of the ReRAM cells programmed in the low-resistance-state (LRS) by rounding the weight value to the neighbor value which can be represented by using fewer LRS. That is, due to the CF shape of ReRAM cells, more serious overlapping variation error is caused by the ReRAM cells programmed to LRS because these cells induce wider current values than the ReRAM cells programmed to high-resistance-state (HRS). Besides, AISD dynamically divides each input cycle into multiple cycles by being aware of the bit positions representing different degree of importance. For example, an input cycle near the MSB position will be divided into much more sub-cycles so as to involve fewer activated wordlines in each sub-cycle and thus relieve the reliability issue in the more critical computation cycles.

V. Conclusion

In this special session paper, we illustrate that emerging resistive memories enable a paradigm shift for future computing platforms, yet the non-ideality of current resistive memory devices might hinder its realization. A few examples of cross-layer designs are provided to demonstrate how the design challenges can be tackled. Such cross-layer designs can fully exploit the potential of resistive memories while mitigating its drawbacks. Our illustration and discussion show that cross-layer solutions are attractive and play an essential role in designing future computing systems.

Acknowledgment

This work has been supported by the Ministry of Science and Technology of Taiwan (MOST 109-2221-E-002-147-MY3, 109-2221-E-001-012-MY3, 109-2218-E-002-019, 107-2923-E-001-001-MY3, 108-2221-E-001-001-MY3, and 108-2221-E-001-004-MY3), National Taiwan University (NTU-CC-109L891803), Macronix Inc., Taiwan (109-S-C24), and Deutsche Forschungsgemeinshaft (DFG), as part of the project OneMemory (project number 405422836).

References

[1] O. Mutlu, "The rowhammer problem and other issues we may face as memory becomes denser," in *DATE*, 2017, pp. 1116–1121.

[2] A. Boroumand *et al.*, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *ASPLOS*, 2018, p. 316–331.

[3] R.-S. Liu *et al.*, "NVM Duet: Unified working memory and persistent store architecture," in *ASPLOS*, 2014, p. 455–470.

[4] W.-C. Wang *et al.*, "Achieving lossless accuracy with lossy programming for efficient neural-network training on NVM-based systems," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 5s, 2019.

[5] Y.-W. Kang *et al.*, "On minimizing analog variation errors to resolve the scalability issue of ReRAM-based crossbar accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3856–3867, 2020.

[6] M.-Y. Lin *et al.*, "DL-RSIM: A simulation framework to enable reliable ReRAM-based accelerators for deep learning," in *ICCAD*, 2018, pp. 1–8.

[7] B. C. Lee *et al.*, "Phase-change technology and the future of main memory," *IEEE Micro*, vol. 30, no. 1, p. 143, Jan. 2010.

[8] J. Wang *et al.*, "Energy-efficient multi-level cell phase-change memory system with data encoding," in *ICCD*, 2011, pp. 175–182.

[9] H. . P. Wong *et al.*, "Metal–Oxide RRAM," *Proc. IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[10] K. C. Hsu *et al.*, "A study of array resistance distribution and a novel operation algorithm for WOx ReRAM memory," in *SSDM*, 2015.

[11] M. Suri *et al.*, "Neuromorphic hybrid RRAM-CMOS RBM architecture," in *NVMTS*, 2015, pp. 1–6.

[12] C. Xu *et al.*, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *DAC*, 2013, pp. 1–6.

[13] C. Xu *et al.*, "Overcoming the challenges of crossbar resistive memory architectures," in *HPCA*, 2015, pp. 476–488.

[14] Intel Corp., "Intel Optane data center persistent memory," in *IEEE Hot Chips (HCS)*, 2019, pp. i–xxv.

[15] O. Zilberberg *et al.*, "Phase-Change Memory: An architectural perspective," *ACM Comput. Surv.*, vol. 45, no. 3, Jul. 2013.

[16] H. Kim *et al.*, "Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches," *ACM Trans. Storage*, vol. 10, no. 4, Oct. 2014.

[17] B. Chen *et al.*, "Physical mechanisms of endurance degradation in TMO-RRAM," in *IEDM*, 2011, pp. 12.3.1–12.3.4.

[18] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *MICRO*, 2009, p. 347–357.

[19] M. K. Qureshi *et al.*, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *MICRO*, 2009, pp. 14–23.

[20] S. Schechter *et al.*, "Use ECP, not ECC, for hard failures in resistive memories," in *ISCA*, 2010, p. 141–152.

[21] M. K. Qureshi *et al.*, "Improving read performance of Phase Change Memories via write cancellation and write pausing," in *HPCA*, 2010, pp. 1–11.

[22] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication," in *DAC*, 2016, pp. 1–6.

[23] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *ISCA*, 2016, pp. 27–39.

[24] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016, pp. 14–26.

[25] C. Hakert *et al.*, "Software-Based Memory Analysis Environments for In-Memory Wear-Leveling," in *ASP-DAC*, 2020, pp. 651–658.

[26] C. Hakert *et al.*, "SoftWear: Software-Only In-Memory Wear-Leveling for Non-Volatile Main Memory," *CoRR*, vol. abs/2004.03244, 2020.

[27] C.-F. Wu *et al.*, "Hot-spot suppression for resource-constrained image recognition devices with nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2567–2577, 2018.

[28] C.-H. Chen *et al.*, "Age-based PCM wear leveling with nearly zero search cost," in *DAC*, 2012, pp. 453–458.

[29] W. Chen *et al.*, "A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *ISSCC*, 2018.

[30] T.-H. Yang *et al.*, "Sparse ReRAM engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *ISCA*, 2019, p. 236–249.

[31] E. Ambrosi *et al.*, "Impact of oxide and electrode materials on the switching characteristics of oxide ReRAM devices," *Faraday discussions*, vol. 213, pp. 87–98, 2019.

[32] J. Lin *et al.*, "Rescuing memristor-based computing with non-linear resistance levels," in *DATE*, 2018, pp. 407–412.