This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2907886, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

MRIMA: An MRAM-based In-Memory Accelerator

Shaahin Angizi, *Student Member, IEEE,* Zhezhi He, *Student Member, IEEE,* Amro Awad, *Member, IEEE,* and Deliang Fan, *Member, IEEE*

***Abstract*—In this paper, we propose *MRIMA*, as a novel MRAM-based In-Memory Accelerator for non-volatile, flexible, and efficient in-memory computing. *MRIMA* transforms current Spin Transfer Torque Magnetic Random Access Memory (STT-MRAM) arrays to massively parallel computational units capable of working as both non-volatile memory and in-memory logic. Instead of integrating complex logic units in cost-sensitive memory, *MRIMA* exploits hardware-friendly bit-line computing methods to implement complete Boolean logic functions between operands within a memory array in a single clock cycle, overcoming the multi-cycle logic issue in contemporary Processing-In-Memory (PIM) platforms. We present practical case studies to demonstrate *MRIMA*'s acceleration for binary-weight and low bit-width Convolutional Neural Networks (CNN) as well as data encryption. Our device-to-architecture co-simulation results on CNN acceleration demonstrate that *MRIMA* can obtain 1.7× better energy-efficiency and 11.2× speed-up compared to ASICs, and, 1.8× better energy-efficiency and 2.4× speed-up over the best DRAM-based PIM solutions. As an AES in-memory encryption engine, *MRIMA* shows ∼77% and 21% lower energy consumption compared to CMOS-ASIC and recent domain wall-based design, respectively.**

***Index Terms*—Spintronics, in-memory processing platform, CNN, AES.**

I. Introduction

Over the past decades, the amount of data that is required to be processed and analyzed by computing systems has been increasing dramatically to exascale (10^{18} bytes/s or flops) [1], [2]. However, the inability of modern computing platforms to deliver both energy-efficient and high performance computing solutions leads to a gap between meets and needs [3], [4]. Unfortunately, with current Boolean logic and Complementary Metal Oxide Semiconductor (CMOS)-based computing platforms, such gap will keep widening mainly due to limitations in both *devices* and *architectures*. First, at *device* level, as depicted in Table I, the computing efficiency and performance of CMOS Boolean systems is beginning to stall due to approaching the end of Moore's law and also reaching its power wall (i.e. huge leakage power consumption limits the performance growth when technology scales down) [1], [5]. For example, the highest power efficiency of contemporary CPU and GPU system is only 10GFLOPS/W, which is difficult to substantially improve in the predictable scaled technology node [6]. Second, at the *architecture* level, as depicted in Table I, today's computers are based on Von-Neumann architecture with separate computing and memory units connecting via buses, which leads to memory wall (including long memory access latency, limited memory bandwidth, energy hungry data transfer) and huge leakage power for holding data in volatile memory [4], [7]. For example, it was reported that data transfer between CPUs and off-chip memory consumes two orders of magnitude more energy than a floating point operation [8]. Therefore, there is a great need to leverage innovations from both *device* and *architecture* to build an energy-efficient and high performance computing platform integrating memory and logic to break the existing memory and power walls.

Table I: Current computing platforms vs. *MRIMA*.

		Current Platforms	MRIMA
Device		CMOS transistors	STT-MRAM
Architecture		Von-Neumann	PIM
Breaking the memory wall	unity of memory & logic	✘	✓
	reduced data transfer	✘	✓
	non-volatility	✘	✓
Breaking the power wall	low leakage power	✘	✓
	scaling	✘	✓
	efficient data transfer	✘	✓

In the last two decades, Processing-in-Memory (PIM) architecture, as a potentially viable way to solve the memory wall challenge, have been well explored [4], [5], [9], [10], [11]. The key concept behind PIM is to embed logic units within memory to process data by leveraging the inherent parallel computing mechanism and exploiting large internal memory bandwidth. It could lead to remarkable savings in off-chip data communication energy and latency. PIM architectures ideally should be capable of performing bulk bit-wise operations which is needed in many applications [12]. The proposals for exploiting SRAM-based [13], [14] PIM architectures can be found in recent literature. However, PIM in context of main memory (DRAM- [5], [10]) has drawn much more attention in recent years mainly due to larger memory capacities and off-chip data transfer reduction as opposed to SRAM-based PIM. However, existing DRAM-based PIM architectures have major shortcomings, e.g., high refresh/leakage power, multi-cycle logic operations, operand data overwritten, operand locality, etc.

The PIM architecture has become even more intriguing when integrated with emerging Non-Volatile Memory (NVM) technology, such as Phase Change Memory (PCM) [15] and resistive RAM (ReRAM) [4]. ReRAM and PCM offer more packing density ($\sim 2-4\times$) than DRAM, and hence appear to be competitive alternatives to DRAM. However, they suffer from slower and more power hungry writing operations than DRAM [15]. In emerging NVM technologies, Magnetic RAM (MRAM) technology is another promising high performance candidate for both last level cache and main memory, due to its ultra-low switching energy, non-volatility, superior endurance, excellent retention time, high integration density and compatibility with CMOS technology. Meanwhile, MRAM technology is in the process of commercialization [16]. Hence, PIM in

S. Angizi, Z. He, A. Awad and D. Fan are with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, 32816. E-mail: {angizi, elliot.he}@knights.ucf.edu, {amro.awad, dfan}@ucf.edu.

This work is supported in part by the National Science Foundation under Grant No. 1740126 and Semiconductor Research Corporation nCORE.

the context of different NVMs, without sacrificing memory capacity, can open a new way to realize efficient in-memory computing paradigms [4], [12], [17]. However, existing NVM-based PIM architectures have unavoidably relied on external processing unit to perform complex logic operations which further limits their performance. Additionally, they have been limited to basic logic operations such as AND, OR and XOR so far [12], [18], which are not necessarily applicable to a wide variety of tasks except by imposing multi-cycle operations to realize specific functions such as addition [10], [19].

The main ***goal*** of this paper is to develop a multipurpose, flexible, non-volatile, parallel, and energy-efficient PIM architecture that could simultaneously work as a non-volatile memory and realize a high performance accelerator for both structured and non-structured data-intensive applications. The main contributions of this paper are summarized as follows:

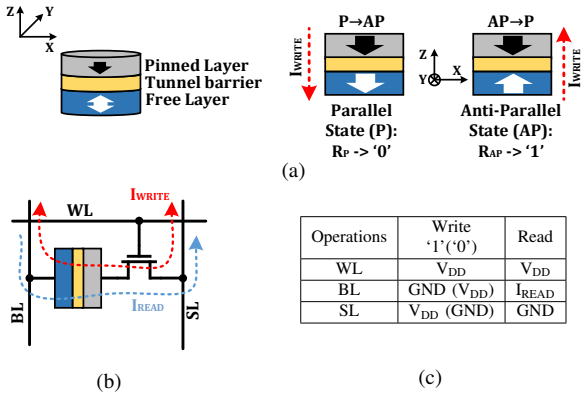
- We propose a novel STT-MRAM in-memory accelerator, *MIRMA*, that integrates important memory and logic functions. *MIRMA* is based on a set of novel microarchitectural and circuit-level schemes that position *MIRMA* as a massive data-parallel unit with negligible area overhead.
- We present *in-memory bit-wise adder* and *in-memory bit-wise convolver* architectures based on *MRIMA* to accelerate binary-weight and low bit-width CNNs and to demonstrate the effectiveness of *MRIMA*, with resource allocation optimization. We further propose detailed mapping methods that harness the full potential of PIM capabilities to reduce CNN's data movement overheads. *MRIMA* is fully capable of realizing CNN-in-memory.
- We present an *in-memory data encryptor*, which shows the superior performance of *MRIMA* as an in-memory encryption engine employing Advanced Encryption Standard (AES) algorithm.
- We discuss major challenges and opportunities for practical integration of *MRIMA* in other architectural layers considering cache coherence, memory layout and interleaving, etc.

The paper is organized as follows. Section II presents the background of MRAM and existing PIM challenges. In Section III, we propose *MRIMA* architecture and its system integration. Section IV presents the acceleration methods of *MRIMA* for different applications. We then present the experimental results to show the efficacy of the proposed platform in Section V. Section VI and Section VII discuss the architectural challenges and related works, respectively. Finally, Section VIII concludes the work.

II. Background

A. Fabrication and Commercialization of MRAM

Recent experiments and fabrication of nano-magnets demonstrate the ability to switch the magnetization using ultra-small current induced Spin-Transfer Torque (STT) or Spin-Orbit Torque (SOT) with high speed (sub-nanosecond), long endurance (10 years) and less than fJ/bit memory write energy (close to SRAM) [20], [21]. Various nanoscale spintronic devices have been explored to realize non-volatile storage devices for MRAM applications, including but not limited to Magnetic Tunnel Junction (MTJ) [22], Domain Wall Motion (DWM) device [23] and SOT-MTJ memory



Operations	Write '1'('0')	Read
WL	V_{DD}	V_{DD}
BL	GND (V_{DD})	I_{READ}
SL	V_{DD} (GND)	GND

(c)

Figure 1: (a) Device structure of MTJ, (b) 1T1R STT-MRAM, (c) Biasing conditions.

device [24]. Several companies, including IBM [25], and Everspin [16] are developing MRAM chips for next-generation universal NVM systems. In early 2016, Everspin announced 256Mb STT-MRAM chips based on MTJ with interface speed similar to DRAM and was planning 1Gb chips in near future [16]. Toshiba and SK Hynix co-developed a 4-Gbit STT-MRAM chip prototype and demonstrated at IEDM 2016 [26]. In summary, with the great advancement of fabrication technology and commercialization progress, MRAM is becoming a next-generation universal NVM technology, with potential applications in both last level cache and main memory. It will greatly change the state-of-the-art memory hierarchy due to its non-volatility, zero leakage power in un-accessed bit-cell, high integration density (2X more than SRAM), excellent endurance ($\sim 10^{15}$ cycles [27]) and compatibility with the CMOS fabrication process (back end of line) [22].

B. STT-MRAM

A typical MTJ structure (Fig. 1a), consists of two ferromagnetic layers with a tunnel barrier sandwiched between them. Due to the Tunnel MagnetoResistance (*TMR*) effect [28], the resistance of MTJ is high (low) when the magnetization of two ferromagnetic layers are in anti-parallel (parallel) state. The *TMR ratio* is defined as $(R_{AP}\text{-}R_P)/R_P$, which may vary from 10% to 400% depending on materials and temperature [28], [29]. Thus, the data are stored as the magnetization direction in the free layer, which could be programmed through current induced STT. Note that, the MTJ with Perpendicular Magnetic Anisotropy (PMA) is used in this work. The 1T1R memory bit-cell is widely used in the typical MRAM design, as depicted in Fig. 1b, which is controlled by Bit Line (BL), Word Line (WL), and Source Line (SL). The biasing conditions of memory read/ write are presented in Fig. 1c.

C. PIM Challenges

Recent PIM architectures have faced several ***limitations*** and ***challenges***. Here, we briefly discuss some of them. ***First***, most of recent PIM designs offer application-specific acceleration architectures rather than a general-purpose platform for computation due to the device-circuit level limitations, so they are not necessarily applicable to other applications. For instance, the ReRAM crossbar-based designs [4], [17], [30], [31] have been widely used to accelerate CNNs. Ambit [10] and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2907886, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

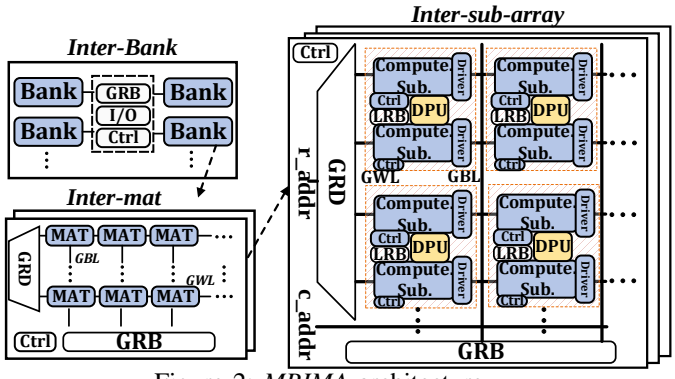


Figure 2: *MRIMA* architecture.

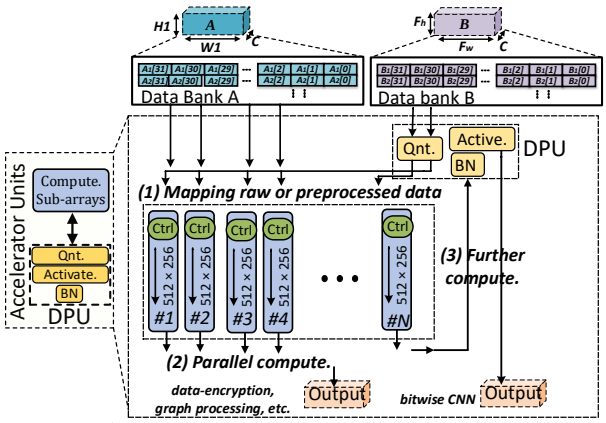


Figure 3: *MRIMA*'s acceleration steps.

Pinatubo [12] as recent in-memory accelerators enable only bulk bit-wise in-memory operations tailored for data-intensive applications. DRISA [5], Compute Cache [13] and CMP-PIM [32] optimize and exploit massive DRAM, SRAM and SOT-MRAM parallelism, respectively, by modifying memory peripherals like SAs at memory sub-array level to perform CNN acceleration. DW-AES [33], RIMPA [34] and HieIM [35] target for designing in-memory encryption engines by developing efficient in-memory XOR units. ***Second***, current PIM schemes unavoidably rely on external processing unit for performing more complex logic operations, otherwise PIM's performance degradation would be considerable due to multi-cycle logic operations. For instance, addition as a preeminent operation for a wide variety of applications, can be more efficiently performed by processor rather than a PIM platform. Recent in-memory addition techniques [19], [36], [34] do not show acceptable performance specially for multi-bit addition. The STT-CiM [37] presents an interesting way to realize in-memory bit-line addition by adding logic gates directly in reconfigurable SA. However, it requires additional memory cycles to save carry out bit back to the memory and uses it for computation of next bits. ***Third***, in addition to large refresh power of DRAM-based PIM architectures [5], [10], they are dealing with a destructive *data-overwritten* issue due to the charge sharing characteristic of capacitors. It means that the result of computation will ultimately overwrite the operands. To solve this issue in the context of DRAM, *multi-cycle operations* [5], [10] are set forth which further degrade PIM performance.

III. MRIMA Architecture

A. Architecture

The general memory organization of *MRIMA* is shown in Fig. 2. The main memory chip is basically divided into multiple Banks. Each bank, consists of multiple memory matrices (mats). Banks within the same chip typically share I/O and buffer, and banks in different chips work in a lock-step manner. The mats are connected to a Global Row Decoder (*GRD*) and a shared Global Row Buffer (*GRB*). Each mat consists of multiple memory sub-arrays connected to a *GRD* and *GRB*. According to the application type and physical address of operands within memory, *MRIMA*'s Controller (*Ctrl*) is able to configure the computational sub-arrays to perform data-parallel inter-sub-array computations. Every two computational sub-arrays share a Local Row Buffer (*LRB*) as well as a Digital Processing Unit (DPU) to further process the data (if necessary) in specific applications as will be discussed later. Fig. 3 gives an overview on *MRIMA*'s acceleration steps. Assume input tensors A and B (that can belong to a variety of applications) are initially stored in Data Banks of the memory. In the first step, either raw data or preprocessed data (by DPU) are mapped into the computational sub-arrays in specific mats. In the second step, parallel computational sub-arrays, which are designed to handle the computational load employing PIM techniques, perform bulk bit-wise operations between tensors and generate the output. This can be considered as the ultimate output in data-encryption or graph processing applications. Additionally, the generated data can be further processed by DPU to generate the output for neural network-based applications. We elaborate the above-mentioned steps in the rest of the paper.

B. Microarchitecture

Fig. 4a depicts the presented PIM sub-array architecture based on STT-MRAM. This architecture mainly consists of Write Driver (*WD*), modified Memory Row Decoder (*MRD*) (elaborated in Fig. 4b), Memory Column Decoder (*MCD*), reconfigurable Sense Amplifier (*SA*) (Fig. 4b), and can be adjusted by *Ctrl* unit (Fig. 4b) to work in dual mode that perform both memory write/read and bit-line computing.

The key idea to perform memory read and bit-line computing is to choose different thresholds (references) when sensing the selected memory cell(s). The proposed reconfigurable SA, as depicted in Fig. 4b, consists of two sub-SAs and totally six reference-resistance branches that can be selected by enable bits (EN_M, EN_{OR3}, EN_{OR2}, EN_{MAJ}, EN_{AND3}, EN_{AND2}) by the sub-array's *Ctrl* to realize the memory and computation schemes as tabulated in Table II. Such reconfigurable SA could implement memory read and one-threshold based logic functions only by activating one enable at a time e.g. by setting EN_{AND2} to '1', 2-input AND/NAND logic can be readily implemented between operands located in the same bit-line. Meanwhile, by activating two enables at a time e.g. EN_{OR2}, EN_{AND2}, two logic functions can be simultaneously implemented and further used to generate two-threshold based logic functions like XOR2/XNOR2, as explained accordingly.

1) Memory Mode: To *write* a data in a memory cell, the corresponding WL is activated using the *MRD*. Then

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2907886, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

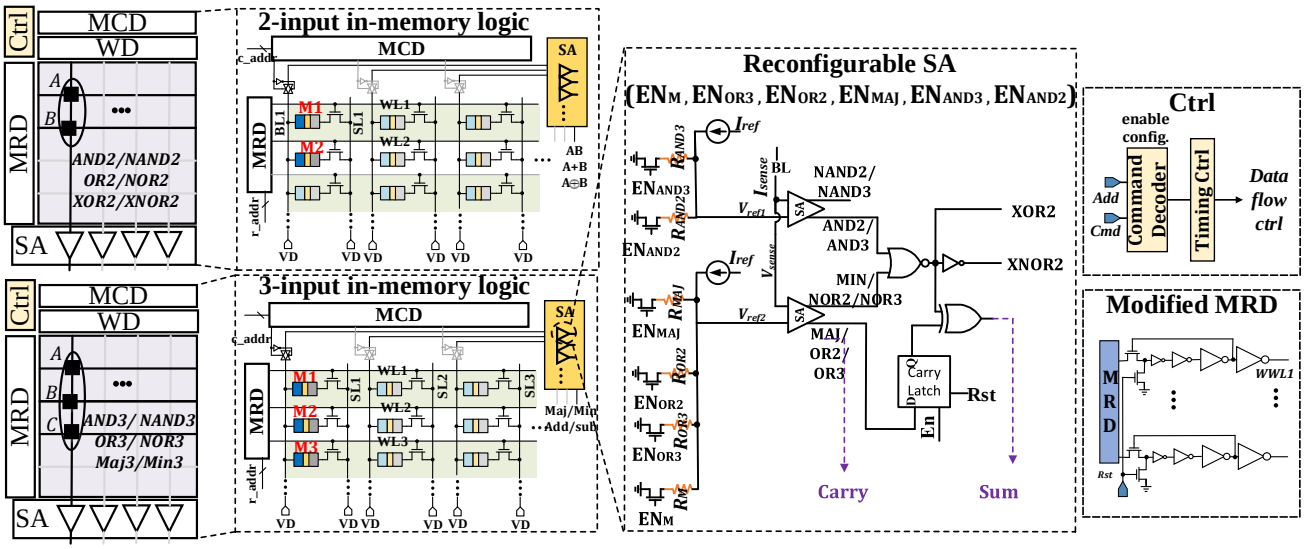


Figure 4: The *MRIMA*'s sub-array architecture: (a) Block level scheme and STT-MRAM realization of 2-input and 3-input in-memory logic methods, (b) Peripherals of computational sub-arrays to support computation.

Table II: Config. of enable bits for different functions.

Ops.	read/ NOT	OR2/ NOR2	AND2/ NAND2	XOR2/ XNOR2	MAJ/ MIN	OR3/ NOR3	AND3/ NAND3
EN_M	1	0	0	0	0	0	0
EN_{OR2}	0	1	0	1	0	0	0
EN_{AND2}	0	0	1	1	0	0	0
EN_{OR3}	0	0	0	0	0	1	0
EN_{AND3}	0	0	0	0	0	0	1
EN_{MAJ}	0	0	0	0	1	0	0

appropriate voltage difference (Fig. 1c) is applied to the corresponding BL and SL using the *WD* connected to them (the write current path is shown in Fig. 1b), leading to MTJ resistance in High-R_{AP} (/Low-R_P). To *read* a data from a memory cell, the corresponding WL is activated using the *MRD* and the corresponding BL is connected to the *SA* using the *MCD* (the read current path is shown in Fig. 1b). The idea of voltage comparison for memory read is shown in Fig. 5a, a single cell is addressed to generate a sense voltage (V_{sense}), which will be compared with memory mode reference voltage activated by EN_M ($V_{sense,P} < V_{ref,M} < V_{sense,AP}$). Now, if the path resistance is higher (/lower) than R_M (memory reference resistance), i.e. R_{AP} (/R_P), then the SA produces High (/Low) voltage indicating logic '1' (/'0'). Note that, one *SA* per BL is considered to maximize the output bandwidth.

•***Fast row copy (FRC):*** *MRIMA*'s FRC mechanism needs a consecutive memory read and write operations. In the first half-cycle, the source row is activated by sub-array's *MRD* and readout to *LRB* (shown in Fig. 2); in the second half-cycle, the data stored in buffer is written back to the destination row. It is noteworthy that FRC can be readily used in mat and bank levels considering inter-component's buffer (*GRB*) to accelerate copy operation in *MRIMA*'s sub-components.

2) Bit-line Computing Mode: The computational sub-array of *MRIMA* is designed to perform bulk bit-wise in-memory logic operations between two or three operands located in the same bit-line.

•***Two-input in-memory logic (IML2x):*** In this method, every two bits stored in an identical column can be selected employing the *MRD* and sensed simultaneously, as depicted in Fig. 4a R.H.S. Then, the equivalent resistance of such parallel connected STT-MRAMs and their cascaded access transistors are compared with a programmable reference by *SA*. Through selecting different reference resistances (R_{AND2}, R_{OR2}), the *SA* can perform basic 2-input in-memory Boolean functions (i.e. AND2 and OR2) e.g. to realize AND operation, R_{ref} is set at the midpoint of $R_{AP}//R_P$ ('1','0') and $R_{AP}//R_{AP}$ ('1','1'). Consider the data organization shown in Fig. 4a L.H.S., where A and B operands correspond to M1 and M2 memory cells in Fig. 4a R.H.S., respectively, IML2x method generates AB after *SA* in a single memory cycle. The idea of voltage comparison between V_{sense} and V_{ref} for IML2x is shown on Fig. 5b. It is worth pointing out that only one sub-SA is used during one-threshold logic operations to reduce the power consumption of sensing. Owing to the complementary outputs of sub-SAs, the reconfigurable SA can also provide NOT, 2-input NOR, NAND functions. The XOR2 logic is realized with two SAs (i.e. performing AND2 and NOR2 logic, simultaneously) and an additional CMOS NOR gate as shown in SA circuit in Fig. 4b.

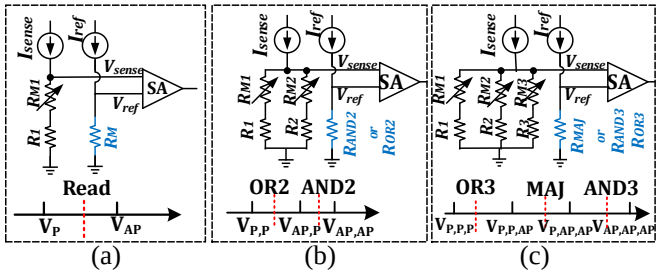


Figure 5: The idea of voltage comparison between V_{sense} and V_{ref} for (a) memory read, (b) IML2x, and (c) IML3x.

•***Three-input in-memory logic (IML3x):*** In this method, every three cells located in an identical column can be selected by *MRD* and sensed simultaneously to realize 3-input logic functions (i.e. AND3/NAND3, OR3/NOR3, MAJ/MIN). For

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2907886, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

instance, consider the data organization shown in Fig. 4a, where A, B, and C operands correspond to M1, M2, and M3 memory cells, respectively, the computational sub-array can perform majority function ($AB + AC + BC$) by setting EN_{MAJ} to '1'. As shown in Fig. 5c, to perform MAJ operation, R_{MAJ} is set at the midpoint of $R_P//R_P//R_{AP}$ ('0','0','1') and $R_P//R_{AP}//R_{AP}$ ('0','1', '1'). Note that, R_1, R_2 and R_3 in Fig. 5 denote the equivalent resistance of selecting transistor, wire, etc. cascaded within the sensing path. We take the average value across the memory array, since normally the equivalent resistance depends on the location of the selected memory cell.

In order to validate the variation tolerance of the sensing circuit, we have performed Monte-Carlo simulation with 10000 trials. A $\sigma = 2\%$ variation is added to the Resistance-Area product (RA_P), and a $\sigma = 5\%$ process variation (typical MTJ conductance variation [3]) is added on the *TMR*. The simulation result of sense voltage (V_{sense}) distributions in Fig. 6 shows the sense margin for memory read, IML2x, and IML3x. It can be seen that sense margin gradually reduces when increasing the number of fan-ins. To avoid logic failure and guarantee the output's reliability, we limited the number of sensed cells to 3. Such sense margin could be even improved by either increasing the sense current or oxide thickness (t_{ox}), but obviously by sacrificing the operation's energy-efficiency. To show this, we first increased the sense current (I_{sense}) from the initial value ($\sim$ $6.6\mu A$), plotted in Fig. 6c, to $\sim$ $18\mu A$ and re-ran the simulation for only IML3x to plot Fig. 6d. We observe that, as we increase the sense current, the voltage margin between two sensitive states ($R_P//R_P//R_{AP}$ and $R_{AP}//R_{AP}//R_P$) has increased from initial 6.31mv to 31.4mv. Note that we don't increase the sensing current above $20\mu A$ to make sure there is no read-write conflict.

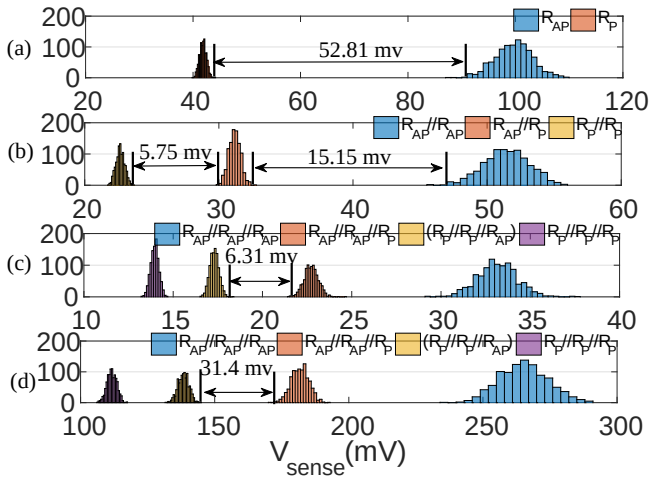


Figure 6: Monte-Carlo simulation of V_{sense} (with RA_P/TMR=2%/5% - t_{ox}=1.5nm) for (a) memory read, (b) IML2x, (c) IML3x when $I_{sense} = 6.6\mu A$, and (d) IML3x when $I_{sense} = 18\mu A$.

To further explore the correlation between I_{sense} and voltage margin for different *MRIMA*'s operations, Fig. 7a shows the voltage margin for memory read, IML2x, and IML3x operations when we gradually increase the I_{sense}. As can be seen, the larger I_{sense} is, the larger voltage margin is achieved

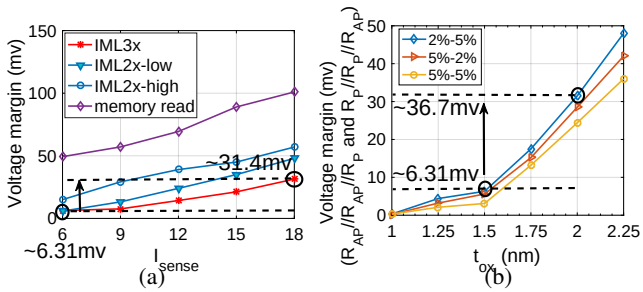


Figure 7: (a) Voltage margin between sensitive states of *MRIMA*'s operations vs. I_{sense} with a t_{ox}=1.5nm, (b) Voltage margin of IML3x operation vs. thickness of MTJ oxide with different variations on RA_P/TMR with a $I_{sense} = 6.6\mu A$.

for different operations. In addition, we investigate IML3x's voltage margin considering different stochastic variations on MTJ's RA_P/TMR (2%/5%, 5%/2%, and 5%/5%) in Fig. 7b by increasing t_{ox}, from 1nm to 2.25nm (as experimentally-demonstrated in [38]). We observe that, increasing t_{ox} from 1.5nm to 2nm leads to $\sim$30.4 mv increase in the sense margin, which considerably enhances the reliability of this operation in *MRIMA*.

In addition to the above-mentioned single-cycle logic operations, *MRIMA*'s sub-array can perform addition/ subtraction (add/sub) operation quite efficiently. With a careful observation on full-adder Boolean logic, we notice that carry-out can be directly produced by MAJ function (Carry in Fig. 4b) just by setting EN_{MAJ} to '1'. Accordingly, we devised a carry latch at this point to store intermediate carry outputs to be used in summation of next bits. Meanwhile, Sum output can be obtained by inserting a 2-input XOR gate in reconfigurable *SA*. Now, assume A, B, and C operands (in Fig. 4a), IML2x and IML3x are able to generate Sum (/Difference) and Carry (/Borrow) bits as will be elaborated in the next section. Parallel computing/read is implemented by using one *SA* per bit-line.

C. System Integration

While *MRIMA* is meant to be an independent high-performance and energy-efficient accelerator, we need to expose it to programmers and system-level libraries to utilize it. From a programmer perspective, *MRIMA* is more of a third party accelerator that can be connected directly to the memory bus or through PCI-Express lanes rather than a memory unit, thus it is integrated similar to that of Graphic-Processing Units (GPUs). Therefore, a virtual machine and ISA for general-purpose parallel thread execution need to be defined similar to PTX [39] for NVIDIA. Accordingly, the programs are translated at install time to the *MRIMA* hardware instruction set tabulated in Table III. The micro and control transfer instructions are not shown in the table.

Table III: The basic instructions of *MRIMA*.

opcode		operation	function
FRC		$B \leftarrow A$	Copy row A to Row B
IML2x	IML21	$A.B$	AND2/NAND2
	IML22	$A+B$	OR2/NOR2
	IML23	$A \oplus B$	XOR2/XNOR2
IML3x	IML31	$A.B.C$	AND3/NAND3
	IML32	$A+B+C$	OR3/NOR3
	IML33	$AB + AC + BC$	MAJ/MIN

The *MRIMA* commands/instructions can be directly copied/written to a predefined memory-mapped address ranges, e.g., defined in the memory type range registers (MTRRs), or programmed through writing to Memory-Mapped I/O regions that are allocated through a simple device driver to do initialization/cleanup for required software memory structures. Note that the first approach can potentially bring more performance gains compared to the later one; accessing *MRIMA* as an I/O device can incur significant overheads due to interrupts and page faults (in case of shared memory model). In contrast, memory-mapped *MRIMA* scheme can cause major contentions in the memory bus in case the processor is executing memory-intensive applications simultaneously. We leave choosing the scheme of integrating *MRIMA* to system architects based on their workloads and usecases. In both schemes for integrating *MRIMA*, the commands/instructions that *MRIMA* architecture accept is similar and based on the ISA.

IV. MRIMA Acceleration Methods

A. CNN Inference Accelerator

In CNNs, Multiplication and Accumulations (MAC) between input and kernels are the key and most computationally -expensive arithmetic operations that always take most fraction of execute time in different hardware implementations [40]. To eliminate the need for massive MAC operations and memory usage, researchers have come up with various quantized/binary CNNs [41], [42] by forcing the inputs/weights to be quantized/binary specifically in inference mode. In this work, we demonstrate that *MRIMA* can accelerate binary-weight CNNs (BWNNs) and low bit-width CNNs using its intrinsic *in-memory bit-wise adder and convolver*. Assume input feature maps (I) and kernels (W) are stored in data banks of memory (Fig. 3). In both networks, except for the inception layer, kernels need to be constantly quantized before mapping into computational sub-arrays. However, quantized shared kernels can be utilized for different inputs. DPU includes three ancillary units (i.e. Quantizer, Batch Normalization and Activation Function). Quantization is basically performed using DPU's Qnt. module and then results are mapped to the parallel sub-arrays (1^{st} step). In the 2^{nd} step, the parallel sub-arrays extract the features using *MRIMA*'s computation methods. Finally, DPU's Active. module activates the generated feature map and complete 3^{rd} step by producing output fmaps.

1) In-memory bit-wise adder: As the main operation of BWNNs, add/sub is the most critical unit of the accelerator [30], [42]. This unit must keep high throughput and resource efficiency while handling different input bit-widths at run-time. Therefore, here we propose a parallel in-memory adder (/subtractor) based on IML2x and IML3x methods to accelerate multi-bit add/sub operations. While there are few designs for in-memory adder/subtractor in literature [5], [19], [34], [36], to the best of our knowledge, this work is the first which presents a fast and fully parallel design in MRAM domain. Fig. 8 shows the requisite data organization and computation steps of binary-weight layers with a straightforward and intuitive example in Fig. 9 only considering add operations. Obviously sub can be implemented based on add.

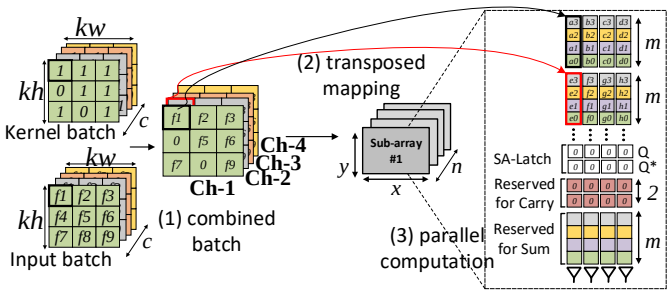


Figure 8: Data organization and computation steps of binary-weight layers.

(1) Initially, c channels (here, 4) in the size of $kh \times kw$ (here, 3×3) are selected from input batch and accordingly produce a combined batch w.r.t. the corresponding binary $\{0,1\}$ kernel batch. Note that, *MRIMA* only employs 2's complement-based data partitioning, mapping and computation method. (2) The combined batch's channels are transposed and mapped to the designated computational sub-arrays. Considering n-activated sub-arrays with the size of $x \times y$, each sub-array can handle the parallel add/sub of up to x elements of m-bit ($3m + 2 \leq y$) and so *MRIMA* could process $n \times x$ elements to maximize the throughput. Here, Ch-1 to Ch-4 are respectively transposed and mapped to sub-array #1. (3) After mapping, the parallel in-memory adder of *MRIMA* accelerator operates to produce the output feature maps. The memory sub-array organization for such parallel computation is delineated in Fig. 8 R.H.S. Two reserved rows for Carry results initialized by zero and m (here, 4) reserved rows are considered for Sum results. We have shown the current state (Q) as well as the next state (Q*) of *SA*'s latch after being enabled for further clarification. We use the add operation of two matrices of 4-bit elements (Ch1 and Ch2) in Fig. 9 to elaborate how addition operates in the *MRIMA*. Every two corresponding elements that are going to be added together have to be aligned in the same bit-line. Here, Ch1 and Ch2 should be aligned in the same sub-array. Ch1 elements take the first 4 rows of the sub-array followed by Ch2 in the next 4 rows.

The addition algorithm starts bit-by-bit from the LSBs of

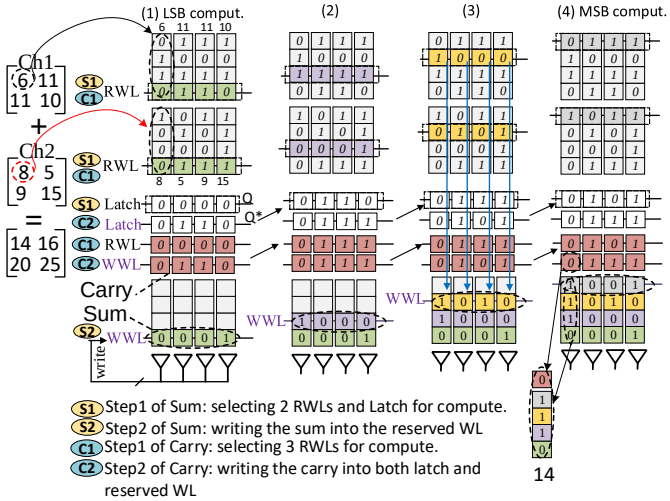


Figure 9: Parallel in-memory addition steps for generating sum and carry-out logic.

the two words and continues towards MSBs. There are 2 cycles for every bit-position computation divided into four steps indicated by S1, S2, C1, and C2. In step 1 of Sum (S1), 2 RWLs (accessing to LSBs of 4 elements) and Latch (storing zero) are enabled to generate the sum. The *SA*s use the 2 bit cells located in the same bit-lines as input operands for IML23 (see Table III) and carry latch's data as carry-in to generate sum based on the method explained in the previous subsection. During step 2 of Sum (S2), a WWL is activated to save back the Sum bit using FRC. In step 1 of Carry (C1), the same 2 operands in conjunction with one of the carry's reserved rows are enabled to generate the carry-out leveraging IML33. During step 2 of Carry (C2), FRC is activated to save back the carry-out bit into a reserved row and also in latch. This carry-out bit overwrites the data in the carry latch and becomes the carry-in of the next cycle. This process is concluded after $2\times m$ cycles, where m is number of bits in elements. To sum it up, *MRIMA*'s bit-wise adder supports different configurations of activation when weight is binary ($<$W:A$>=<1{:}m>$).

2) In-memory bit-wise convolver: The main idea of this scheme is to exploit logic *AND*, *bitcount*, and *bitshift* as rapid and parallelizable operations to accelerate low bit-width (quantized) MACs in convolutional layers. The AND-based convolution of k-bit fixed point integers has been presented in [41]. There are some other layers in CNNs, such as inception layer (directly taking image as inputs and not necessarily quantized) and Fully-Connected (FC) layer. These layer can be equivalently implemented by convolution operations using 1×1 kernels [41]. Thus, all layers could be implemented by convolution computation by exploiting these operations [41], [43]:

$$I * W = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} 2^{m+n} bitcount(AND2(C_n(W), C_m(I))) \quad (1)$$

Assume I is a sequence of M-bit input integers (3-bit as an example in Fig. 10) located in input fmap covered by sliding kernel of W, such that $I_i \in I$ is an M-bit vector representing a fixed-point integer. We index the bits of each I_i element from LSB to MSB with $m = [0, M-1]$, such that $m = 0$ and $m = M-1$ are corresponding to LSB and MSB, respectively. Accordingly, we represent a second sequence denoted as $C_m(I)$ including the combination of m^{th} bit of all I_i elements (shown by elliptic). For instance, $C_0(I)$ vector consists of LSBs of all I_i elements "0110". Considering W as a sequence of N-bit weight integers (3-bit, herein) located in sliding kernel with index of $n = [0, N-1]$, the second sequence can be similarly generated like $C_n(W)$. Now, by considering the set of all m^{th} value sequences, the I can be represented like $I = \sum_{m=0}^{M-1} 2^m c_m(I)$. Likewise, W can be represented like $W = \sum_{n=0}^{N-1} 2^n c_n(W)$.

As shown in data mapping step in Fig. 10, $C_2(W)$-$C_0(W)$ are consequently mapped to the designated sub-arrays of *MRIMA*. Accordingly, $C_2(I) - C_0(I)$ are mapped in the following memory rows in the same way. Now, computational sub-array can perform bit-wise parallel AND2 operation (IML21) of $C_n(W)$ and $C_m(I)$ as depicted in Fig. 10. The results of parallel AND operations stored within sub-array will be accordingly processed using bit-counter. Bitcount is translated

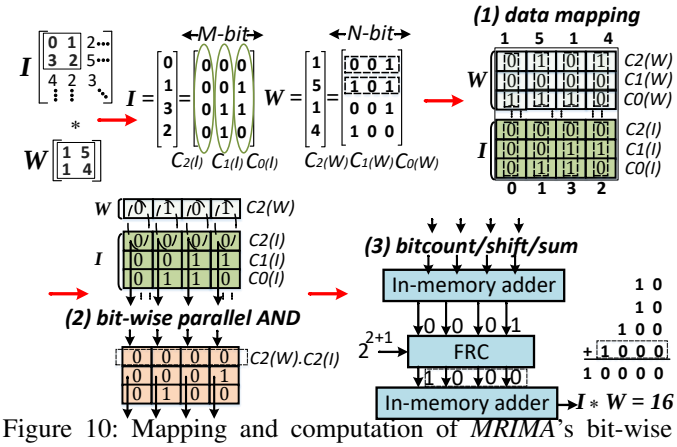


Figure 10: Mapping and computation of *MRIMA*'s bit-wise convolver.

to the addition of bits implemented by our in-memory adder. It passes the data to a shifter implemented by consecutive memory read and write operations (FRC). As depicted in Fig. 10, "0001", produced by in-memory adder is left-shifted by 3-bit ($\times 2^{2+1}$) to "1000". Eventually, in-memory bit-wise adder can produce the output fmaps. Note that *MRIMA*'s bit-wise convolver supports different configurations of weight and activation ($<$W:A$>=<n{:}m>$).

B. Data Encryption Accelerator

As emerging NVMs can potentially host persistent data, hence enable data remanence attacks, their deployment is often paired with some sort of encryption. Specifically, most state-of-the-art secure NVM systems use AES encryption engines to encrypt the data as it gets written to NVM. While the processor is typically the trust base, we can also rely on PIM to do the actual encryption without the need to bring the data all the way to the processor chip, decrypt it, then encrypt it with a new key and write it back again, but rather just doing it on the spot. There are many use-cases in which such in-memory encryption accelerator is useful: encrypting files with different keys, frequent updates for the keys, and frequent reassignment of memory pages for users with different keys. In all such cases, an efficient way of encrypting data is preferred; refreshing keys would no longer throttle memory bandwidth and limit performance of other running applications. We take the AES algorithm as an example to elucidate the mapping of transformations in *MRIMA*, which reveals its benefits of energy-efficiency and high throughput for in-memory data encryption applications. AES is an iterative symmetric-key cipher where both sender and receiver units use a single key for encryption and decryption. AES basically works on the standard input length of 16 bytes (128 bits) data organized in a 4×4 matrix (called state matrix (S_M)) while using 3 different key lengths (128, 192, and 256 bits) [33]. For 128-bit key length, AES encrypts the input data after 10 rounds of consecutive transformations enumerated as SubBytes, ShiftRows, MixColumns, and AddRoundKey (Fig. 11a).

To facilitate working with input data, each byte in input data is distributed into 8-bit (Fig. 11b). So, 8 memory arrays are filled by 4×4 bitmatrices. Mapping of four AES transformations to *MRIMA* is shown in Fig. 11d. In SubBytes stage,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2907886, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

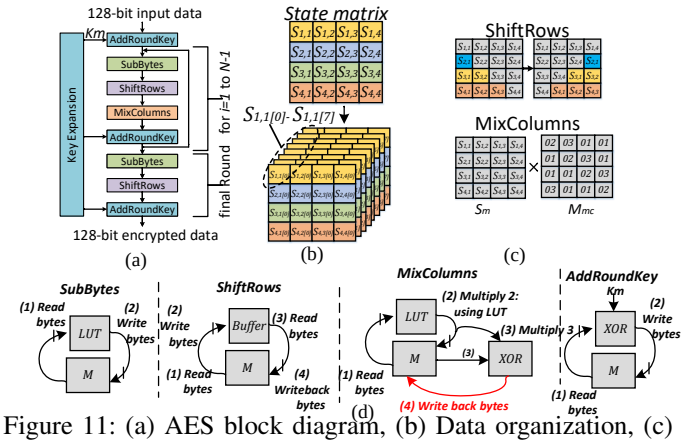


Figure 11: (a) AES block diagram, (b) Data organization, (c) ShiftRows and MixColumns transformation, (d) Mapping of AES's transformations to *MRIMA*.

each byte of S_M will undergo a Look-Up Table (LUT) based transformation using S-box which was conventionally stored in SRAM with significant leakage power. However, it can be readily implemented within the *MRIMA* sub-arrays with no add-on circuits with consecutive read/write operations as shown in Fig. 11d. In ShiftRows stage, S_M will undergo a cyclical shift operation by a certain offset. One of the STT-MRAM arrays is considered as a buffer to temporarily save the readout data. In this way, after reading the data from second to fourth row (3 rows), they can be easily rewritten to the memory with desired order. In MixColumns and AddRoundKey stage, parallel in-memory `XOR2` operation (IML23) along with `FRC` operations are used.

V. Evaluation

•***Evaluation platform:*** To assess the performance of *MRIMA* as a new PIM platform, a comprehensive device-to-architecture evaluation framework along with two in-house simulators are developed. First, at the device level, we jointly use the Non-Equilibrium Green's Function (NEGF) and Landau-Lifshitz-Gilbert (LLG) equations to model STT-MRAM bitcell (i.e. MTJ) [3], [44] based on device parameters tabulated in Table IV. For the circuit level simulation, a Verilog-A model of 1T1R STT-MRAM device is developed to co-simulate with the interface CMOS circuits in Cadence Spectre and SPICE. 45nm NCSU Product Development Kit (PDK) library [45] is used in SPICE to verify the proposed design and acquire the performance of designs. Second, an architectural-level simulator is built based on NVSim [46]. NVSim is a non-volatile memory circuit simulator and reports MRAM performance parameters, which can be calibrated with fabrication MRAMs. Based on the device/circuit level results, our simulator can alter the configuration files (.cfg) corresponding to different array organization and report performance metrics for PIM operations. The controllers and add-on circuits are synthesized by Design Compiler [47] with an industry library. Third, a behavioral-level simulator is developed in Matlab calculating the latency and energy that *MRIMA* spends considering a particular application. In addition, it has a mapping optimization framework for the CNN and data encryption applications. Besides, we developed a comprehensive Verilog model for DPU interacting with our SPICE level circuit implementation to run the simulation and perform the evaluation. There are two activation functions being used in *MRIMA* (i.e. $\frac{tanh(x)+1}{2}$ and $sign(x)$). From hardware implementation perspective, activation functions were developed using lookup-table-based transformations [48] with case-statement codes. Batch normalization unit alleviates the information loss during quantization by normalizing the input batch to have zero mean and unit variance. It generally performs an affine function $y = kx + h$ [49], where y and x denote the corresponding output and input feature map pixels, respectively. During inference mode, all the other parameters are pre-computed and stored in *MRIMA* arrays. Therefore, BN can fetch each pixel of feature maps, fed forward to the batch-norm layer, and write back the corresponding normalized pixel employing an internal, multiplexed CMOS adder and multiplier to perform this computation efficiently.

Table IV: Simulations Parameters.

Parameter	Value
Free layer dimension $(W \times L \times t)_{FL}$	$65 \times 65 \times 2\ nm^3$
Polarization factor, P	0.4
Gilbert Damping Factor, α	0.007
Saturation Magnetization, M_s	$850\ kA/m$
Oxide thickness, t_{ox}	$1.5\ nm$
RA product, RA_p / TMR	$10.58\ \Omega \cdot \mu m^2$ / 171.2%
Supply voltage	$1\ V$
CMOS technology	$45\ nm$
STT-MRAM cell area	$48F^2$
Access transistor width	$9F$
Cell aspect Ratio	1.34

•***Experimental setup for MRIMA:*** We configure *MRIMA*'s memory organization with 512 rows and 256 columns per sub-array with total 16 sub-arrays per mat in a H-tree routing manner, 2×2 mats (with 2/2 and 2/2 as row and column activations) per bank, 4×4 banks (with 1/4 and 4/4 as row and column activations) per group; in total 4 groups and 512Mb total capacity.

•***Area and peak performance:*** The area of *MRIMA* is $109.6mm^2$. Fig. 12a shows the breakdown of the area overhead resulted from add-on hardware to original MRAM chip. Our experiments show that, in total, *MRIMA* imposes 5.6% area overhead to the memory die, where Pinatubo [12], RIMPA [34], and DRISA [5] incur 0.9%, 17%, and 5% area overhead, respectively. We observe that the modified controller and drivers contribute more than 50% of this area overhead in a memory group. Obviously, the choice of the number of sub-arrays is a trade-off between peak $GOps/s$ and area overhead. Enlarging the chip area brings in a higher performance for *MRIMA* and other PIM designs due to the increased number of sub-arrays, though the die size directly impacts the chip cost. Fig. 12b shows this trade-off considering both computational and power efficiency metrics [17], [31]. With current configuration, the computational efficiency of *MRIMA* is 1521.83 $GOps/s/mm^2$ which is higher than PipeLayer-ReRAM [31] (1485), ISAAC-ReRAM [17] (478.9) and DaDianNao-ASIC [50] (63.46). Power efficiency of *MRIMA* is 455.48 $GOps/W$ which is higher than PipeLayer-ReRAM (142.9), ISAAC-ReRAM (380.7) and DaDianNao-ASIC (286.4). To have a fair comparison, the area-normalized results will be reported in section 5.2 for different platforms.

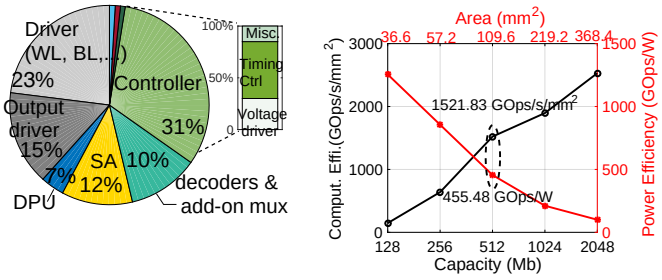


Figure 12: (a) Area overhead of *MRIMA* in a memory group, (b) Area-peak performance trade-off.

A. Non-structured Bulk Benchmark Evaluation

We first analyze the logic performance of *MRIMA* compared to recent PIM platforms taking intrinsically-non-structured ISCAS85 benchmarks. A logic netlist in Berkeley Logic Interchange Format (.blif) is fed into ThrEshold Logic Synthesizer (TELS) [51] to obtain synthesized logic networks. Meanwhile, parameters such as fan-in restriction is set up during the synthesis. The synthesized networks are then mapped to *MRIMA* using the developed simulator to assess the performance. Fig. 13 gives energy and delay of ISCAS85 combinational circuit benchmarks implemented using *MRIMA*, Pinatubo [12], STT-CiM [37], RIMPA [34], HieIM [35], and Ambit [10]. To have an impartial comparison, Pinatubo, as a general system architecture for NVMs, is implemented with the same standard STT-MRAM, SOT-MRAM, and ReRAM technologies. We observe that *MRIMA* spends the lowest energy and delay compared to the counterparts in different benchmarks. (1) *MRIMA* reduces the energy consumption by $\sim$ 72%, 61.2%, 75.5%, and 86.2% compared to Pinatubo-STT [12], STT-CiM [37], HieIM [35], and Ambit [10], respectively. This considerable improvement mainly comes from the proposed logic efficiency and reduced-cycle operations. (2) *MRIMA* outperforms the mentioned PIM architectures respectively with 40.8%, 38.3%, 66.7%, and 95% reduction in delay on different benchmarks. For five more complex benchmarks (i.e. c2670, c3540, c5315, c6288, and c7552), as logic complexity increases, *MRIMA* can show much better performance compared to the rest.

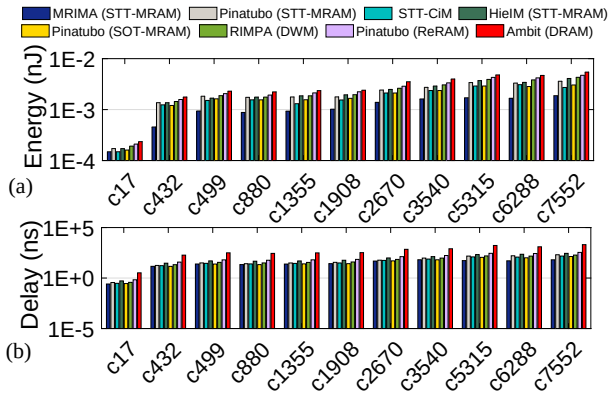


Figure 13: (a) Energy and (b) Delay of ISCAS85 benchmarks (Y-axis: Log scale).

B. CNN Acceleration Performance

In this subsection, we compare *MRIMA* with state-of-the-art DRAM-, ReRAM-, ASIC-, and GPU-based solutions for the CNN inference acceleration.

1) Modeling Setup: **Bit-width:** Four bit-width configurations of <W:I> (<1:1>,<1:2>,<1:4>,<1:8>) are considered for the evaluation with an 8-bit gradient (<G>). **Data-set:** The SVHN data-set [52] is selected. The images are re-sized to 40×40 and fed to the model. **Model:** A CNN with 6 binary-weight convolutional layers, 2 (average) pooling layers and 2 FC layers is adopted. FC layers are equivalently implemented by bit-wise convolutions. **Training:** We used open source algorithm by DoReFa-Net [41] where all the operations can be accelerated significantly using bit-wise convolution of fixed-point integers. We adopt batch normalization and different dropout techniques to accelerate and avoid over-fitting. The model is trained on TensorFlow [53] with 100 epochs and the lowest test error of epoch is reported.

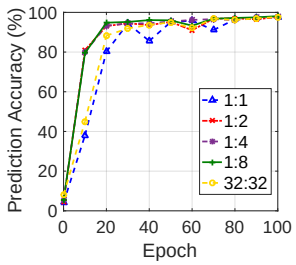
2) Accelerators' Setup: ***DRAM:*** We developed a DRISA-like [5] accelerator for binary-weight CNNs. Two different computing methods of DRISA named $3T1C$ and $1T1C\text{-}adder$ were selected for comparison. The 3T1C uses DRAM cells themselves for computing and naturally performs NOR logic on BLs. However, 1T1C-adder exploits a large n-bit adder circuit for n-bit BLs after SAs. We accordingly modified CACTI-3DD [54] for evaluation of DRAM's solutions. Similar to [5], the controllers and adders were synthesized in Design Compiler [47]. ***ReRAM:*** A Prime-like [4] accelerator with two full functional (FF) sub-arrays and one buffer sub-array per bank (totally 64 sub-arrays) were considered for evaluation. In FF subarrays, for each mat, there are 256×256 ReRAM cells and eight 6-bit reconfigurable SAs. For evaluation, NVSim simulator [46] was extensively modified to emulate Prime functionality. Note that the default NVSim's ReRAM cell file (.cell) was adopted for the assessment. ***STT-MRAM:*** A STT-CiM-like [37] accelerator was developed with the exactly same memory configuration as *MRIMA* considering 512 rows and 256 columns computational sub-arrays and 512Mb total memory capacity. We used the same peripheral circuitry and DPU as in *MRIMA* to perform an impartial comparison. Accordingly, we used the evaluation platform developed for *MRIMA* to assess STT-CiM performance in accelerating CNNs. ***ASIC:*** We developed a DaDianNao-like [50] accelerator. To have a fair comparison, we select two versions with either 8×8 tiles or 16×16 tiles. Accordingly, we synthesized the designs with Design Compiler [47] under 45 nm process node. The eDRAM and SRAM performance were estimated using CACTI [55]. ***GPU:*** We used the NVIDIA GTX 1080Ti Pascal GPU. It has 3584 CUDA cores running at 1.5GHz (11TFLOPs peak performance). The energy consumption was measured with NVIDIA's system management interface. Similar to [5], we scaled the achieved results by 50% to exclude the energy consumed by cooling, etc. Accordingly, based on bit-width configuration of <I> i.e. 1, 2, 4, 8, we aggressively scaled GPU results by ×32, ×16, ×8, and ×4, respectively, to get the peak performance for each quantized networks. Note that, GPU doesn't support fixed point CNN and real scale ratio should be less than these numbers [5], [8].

3) Accuracy: Fig. 14a tabulates the test error results and relative complexity of the discussed model under various con-

figurations. Complexity of inference and training are achieved using $W \times I$ and $W \times I + W \times G$, respectively. Generally, experiments replicate the conclusion drawn by [41] that weights, inputs and gradients are progressively more sensitive to bit-width changes. Fig. 14b depicts the prediction accuracy curve vs. number of epoch in different configurations. We observe that the low bit-width networks can keep the accuracy high compared to the original 32-bit case.

Config.		Complexity		Test Error
W	I	Inference	Training	our Model
32	32	-(†)	-	2.4%
1	1	1	9	3.1%
1	2	2	10	2.6%
1	4	4	12	2.4%
1	8	8	16	2.3%

(†) The computation complexity of 32:32 is not shown, since it is not computationally efficient to perform bit-wise convolution of 32:32 configuration [41] and it is already reported in previous works.



(a) (b)

Figure 14: (a) Test error of the CNN model, (b) Prediction accuracy vs. epoch.

4) Energy Consumption: Fig. 15 shows the *MRIMA*'s energy-efficiency results on CNN application with a batch size of 8 and 64 in different <W:I>. As shown, *MRIMA* solution offers the highest energy-efficiency normalized to area compared to others owning to its energy-efficient and parallel operations. We observe that *MRIMA* achieves ∼1.8× and 2.1× higher energy-efficiency than that of DRAM-3T1C and 1T1C-adder, respectively. The main reason here is the energy-efficiency of operations in *MRIMA*; as discussed earlier, *MRIMA* can finish the operation (such as `IML21`) in one-single cycle, however similar operation in DRAM-3T1C imposes multi-cycle operations to avoid destructive data-overwritten. Besides, the n-bit adder located after SAs in DRAM-1T1C-adder solution will bring higher performance compared to 1T1C, though it has limited its energy-efficiency. Fig. 15 shows that *MRIMA* solution is 1.7× more energy-efficient than the best ASIC solution. In addition, it shows ∼8.5× saving in energy compared to ReRAM solution. It is worth pointing out that *MRIMA* doesn't follow the conventional ReRAM-based crossbar designs to realize CNN-in-memory, which brings significant energy-efficiency due to eliminating DAC/ADC units. Compared to STT-CiM counterpart, *MRIMA* obtains on average 1.4× higher energy-efficiency normalized to area. It is worth pointing out that STT-CiM imposes additional memory cycles and consecutively energy to save Carry bit in addition operation. This was alleviated using *MRIMA*s in-SA latch as explained in Section III.B.

5) Performance: Fig. 16 shows the *MRIMA*'s performance results on CNN application in different <W:I>. It shows that *MRIMA* solution is 2.4× faster than the best DRAM solution (1T1C-adder) and 11.2× faster than ASIC64 solution. This is mainly because of (1) ultra-fast and parallel in-memory operations of *MRIMA* compared to multi-cycle DRAM operations and (2) the existing mismatch between computation and data movement in ASIC designs and even 1T1C-adder solution. As a result, ASIC256 with more tiles does not show higher performance. We can also observe that the larger the

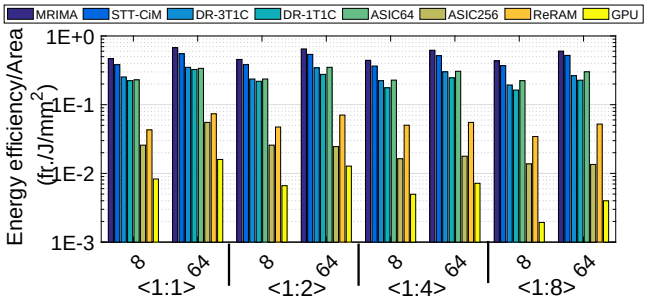


Figure 15: Energy-efficiency of different CNN accelerators.

activation's bit-width is, the higher performance is obtained for *MRIMA* solution compared to DRAMs owning to its more paralleled computations. Additionally, we see that *MRIMA* is 8.1× faster that ReRAM solution. Note that ReRAM design employs matrix splitting due to intrinsically limited bit levels of ReRAM device so multiple sub-arrays should be occupied, besides ReRAM-based crossbar has a large peripheral circuit's overhead such as buffers and DAC/ADC which contribute more than 85% of area [4]. *MRIMA* achieves on average 1.5× higher speed-up compared with STT-CiM, with the exactly same memory configuration. This mainly comes from *MRIMA*'s fast and fully parallel operations.

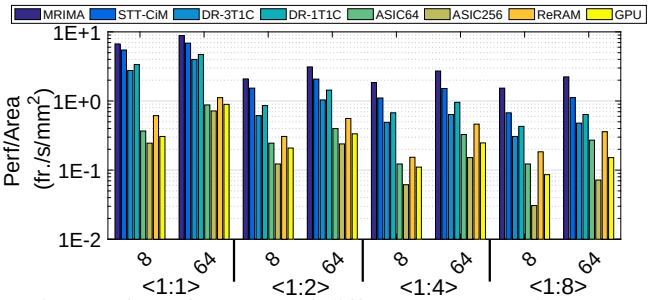


Figure 16: Performance of different CNN accelerators.

Fig. 17 shows the breakdown of the energy and delay measurement of convolutional layers for four PIM-based solutions, i.e. *MRIMA*, STT-CiM, DR-3T1C, and ReRAM into the read and write parts for two bit-width configurations <1:1> and <1:4>. We can observe that *MRIMA* outperforms other platforms in terms of number write-back operations leading to a reduced energy and delay. Note that, while the other PIM counterpart designs based on NVMs such as Prime [4], ISAAC [17], etc. propose to implement full bit-wise CNN inside ReRAM, *MRIMA* proposes an alternative way, not only taking advantage of a higher endurance memory (MRAM), but also providing a faster and more energy-efficient computation solution for such data-intensive application. As a numerical evaluation, assuming the most write-intensive application in this work, i.e. bit-wise CNN with <1:8> configuration, with the same layer structure, *MRIMA* requires 9224 $\simeq 10^5$ write cycles. Therefore, even by reusing computational sub-arrays by repeatedly R&W operations, *MRIMA* can readily run the application.

6) Memory Wall: Fig. 18 depicts the memory bottleneck ratio i.e. the time fraction at which the computation has to wait for data and on-/off-chip data transfer obstructs its performance (memory wall happens). The evaluation is per-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2907886, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

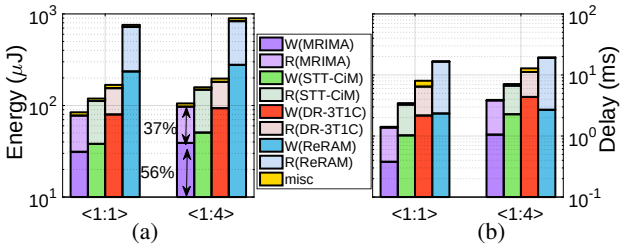


Figure 17: Break-down of (a) Energy and (b) Delay of four PIM platforms.

formed according to the peak performance and experimentally extracted results for each platform considering number of memory access in each bit-width configuration. The results[1] show the *MRIMA*'s favorable solution for solving memory wall issue. (1) We observe that *MRIMA*, STT-CiM and DRAM-3T1C solutions spend less than ∼15% time for memory access and data transfer. While ASIC- and DRAM-1T1C accelerators spend more than 90% time waiting for the loading data. (2) In larger activation bit-widths (<I>=4 and 8), ReRAM solution shows lower memory bottleneck ratio compared with *MRIMA*. This comes from two sources: (1) increased number of computational cycles and (2) unbalanced computation and data movement of *MRIMA* due to limitation in number of activated sub-arrays when operands get larger.

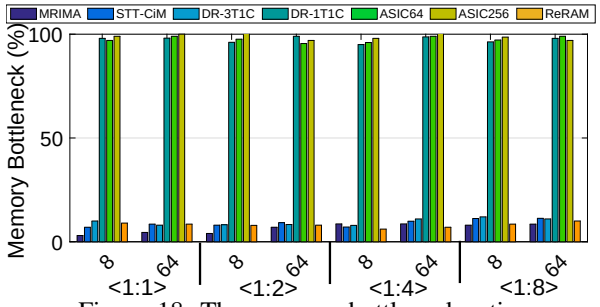


Figure 18: The memory bottleneck ratio.

7) Resource Utilization: The less memory wall ratio can be interpreted as the higher resource utilization ratio for the accelerators, which is shown in Fig. 19. For instance, in <1:8>, *MRIMA*, STT-CiM, DRAM-3T1C and ReRAM solutions utilize the highest ratio (up to 65%) which reconfirms the results reported in Fig. 18.

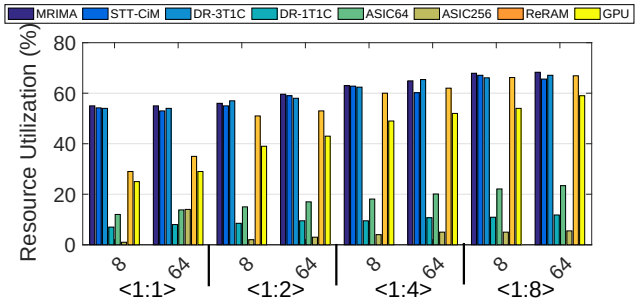


Figure 19: The resource utilization ratio.

[1]GPU data could not be accurately reported for this evaluation.

C. Encryption Accelerator Performance

We assess the performance of 128-bit AES implemented by *MRIMA*, general purpose processor (GPP), ASIC, CMOL [56], DW-AES [33], RIMPA [34], Ambit [10] and Pinatubo [12] in terms of energy consumption and number of cycles required for the process. For evaluation of AES performance in GPP, we have used similar method in [33] at 2GHz. AES C code is extracted from [57] and compiled, then cycle-accurate architecture simulator gem5 [58] is employed to take AES binary and accordingly system level processor power evaluating tool McPAT [59] is used to estimate power dissipation. For evaluation of AES in CMOS ASIC (1.133GHz), Synopsys Design Compiler tool is used. Fig. 20a and Fig. 20b show the breakdown of energy[2] and number of cycles required for different AES transformations after mapping to the different platforms, respectively.

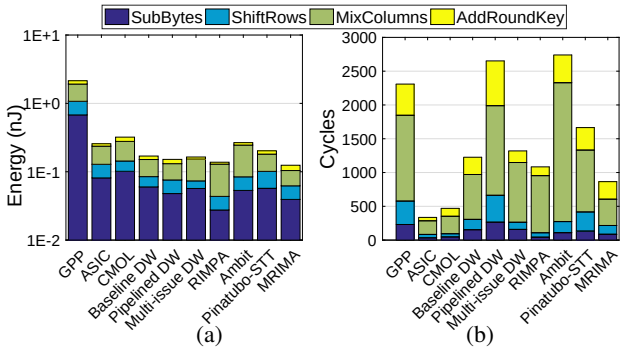


Figure 20: Breakdown of (a) Energy consumption and (b) Delay of different AES implementations.

We observe that MixColumns consumes the most clock cycles as well as energy due to the high number of resources (memory and in-memory XOR) that it takes during operation. In some of the XOR-unfriendly platforms such as Ambit [10] and RIMPA [34], MixColumns contributes to more than 70% of the energy consumption and number of cycles. The results reveal the *MRIMA*'s energy-efficiency (Fig. 20a) compared to other implementations. For instance, it can achieve ∼77% and 21% lower energy consumption compared to CMOS-ASIC and RIMPA implementations, respectively at 30MHz. However, from the number of cycles perspective, ASIC (with 336 cycles) and CMOL (470) designs show better performance compared to *MRIMA* (865).

VI. Discussions

•***Cache coherence:*** One major concern that is common across most off-chip accelerators is cache coherence. When *MRIMA* updates data directly in memory, there could be stale copies of the updated memory locations in the cache, thus data inconsistency issues may arise. Similarly, if the processor updates cached copies from memory locations that *MRIMA* will process later, *MRIMA* could actually use wrong/stale values. There are several ways to solve such issues in accelerators, the most common one is to rely on operating system (OS) to unmap the physical pages accessible by *MRIMA* from

[2]Y-axis in Fig. 20a: Log scale.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2907886, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Table V: Related works comparison.

Features	DRISA [5]	Ambit [10]	Prime [4]	HieIM [35]	MPIM [18]	ISAAC [17]	Pinatubo [12]	RIMPA [34]	STT-CiM [37]	MRIMA
technology	DRAM	DRAM	ReRAM	STT-MRAM	ReRAM	ReRAM	PCM/ReRAM/...	DWM	STT-MRAM	STT-MRAM
volatility	volatile	volatile	non-volatile	non-volatile	non-volatile	non-volatile	non-volatile	non-volatile	non-volatile	non-volatile
in-memory logic	AND/NAND OR/NOR Maj-logic	AND/NAND OR/NOR MAj-logic	N/A	AND/NAND OR/NOR	AND/NAND OR/NOR XOR/XNOR	N/A	AND/NAND OR/NOR XOR/XNOR	AND/NAND OR/NOR Maj-logic	AND/NAND OR/NOR XOR	AND/NAND OR/NOR XOR/XNOR Maj-logic
in-memory convolver	3T1C/1T1C-NOR/ 1T1C-mixed/1T1C-adder	N/A	crossbar	N/A	crossbar	crossbar	N/A	N/A	N/A	bit-wise adder/ bit-wise convolver

any process that can run while computing in *MRIMA*. The other solution, which tends to be more expensive hardware-wise, is to allow the memory controller to snoop coherence transactions and pass them to *MRIMA* architecture, i.e., adding *MRIMA* to the coherence domain. We believe that the decision on how to implement coherence with *MRIMA* is highly-dependant on its usecase and can evolve with time similar to coherence with GPUs.

•***Virtual memory:*** *MRIMA* has its own ISA with operations that can potentially use virtual addresses. To use virtual addresses, *MRIMA*'s $Ctrl$ must have the ability to translate virtual addresses to physical addresses. While in theory this looks as simple as passing the address of the page table root to *MRIMA* and giving *MRIMA*'s $Ctrl$ the ability to walk the page table, it is way more complicated in real-world designs. The main challenge here is that the page table can be scattered across different DIMMs and channels, while *MRIMA* operates within a memory module. To avoid the complexity of virtual memory when using *MRIMA*, system architects can opt for designating a continuous physical range that can be used by *MRIMA* and the user/application can use physical addresses for operands. Directly operating on physical addresses can limit multi-tasking on *MRIMA*, however, we leave supporting multi-tasking in *MRIMA* through virtual memory support as future work.

•***Memory layout and interleaving:*** *MRIMA* strives to optimize for performance and power-efficiency. While high-performance memory systems rely on channel interleaving to maximize the memory bandwidth, *MRIMA* adopts a different approach through maximizing spatial locality and allocating memory as close to their corresponding operands as possible. The main goal is to reduce the data movement across memory modules and hence reducing operations latency and energy costs. As exposing a programmer directly to the layout of memory is challenging, *MRIMA* architecture can rely on compiler passes that take memory layout and the program as input, then assign physical addresses that are adequate to each operation without impacting the symantics of the application.

•***Reliability:*** Many ECC-enabled DIMMs rely on calculating some hamming code at the memory controller and use it to correct any soft errors. Unfortunately, such a feature is not available for *MRIMA* as the data being processed are not visible to the memory controller. Note that this issue is common across all PIM designs. To overcome this issue, *MRIMA* can potentially augment each row with additional ECC bits that can be calculated and verified at the memory module level or bank level. Augmenting *MRIMA* with reliability guarantees is left as future work.

VII. Related Works

There is a great deal of PIM accelerators that present reconfigurable platforms or application-specific logics in or close to memory die [9], [11], [31], [60], [61], [62], [63], [64]. Due to the lack of space, we shall restrict our comparison (provided in Table V) to nine different platforms given four important features, i.e. fabrication technology, volatility, supporting in-memory logic and in-memory convolver. Table V includes: DRISA [5] and Ambit [10] as DRAM-based PIM architectures, PRIME [4] and ISAAC [17] as only crossbar-based dot-product engines for Neural Network acceleration based on ReRAM not supporting in-memory-logic, Pinatubo [12] as a general architecture capable of doing bulk bit-wise operations, MPIM [18] as a multi-purpose ReRAM-based PIM, RIMPA [34] as a threshold logic PIM architecture based on Domain Wall-RAM (DW-RAM) as well as HieIM [35] and STT-CiM [37] as recent STT-MRAM-based PIM platforms. To the best of our knowledge, this is the first work that proposes an STT-MRAM PIM as an accelerator for a wide variety of tasks such as CNN acceleration and data encryption. Based on Table V, *MRIMA* is the only PIM architecture that not only benefits from non-volatility, but also can implement a full set of 2- and 3-input Boolean in-memory logic functions as well as majority-based logic operations using its distinct computing methods. Additionally, it can be as powerful as DRISA [5] and MPIM [18] to accelerate in-memory convolution very efficiently using different bit-wise methods.

VIII. Conclusion

In this paper, we proposed *MRIMA*, as a novel MRAM-based In-Memory Accelerator for multipurpose, highly flexible and efficient computation. *MRIMA* was developed based on STT-MRAM array and optimized to achieve high performance. It can be reconfigured to efficiently perform non-volatile memory operations and in-memory logic. Our simulation results on CNN acceleration task demonstrate that *MRIMA* can obtain $1.7\times$ better energy-efficiency and $11.2\times$ speedup compared to ASICs, and $1.8\times$ better energy-efficiency $2.4\times$ speedup over the best DRAM-based PIM solutions. As an AES in-memory encryption engine, *MRIMA* shows $\sim$77% and 21% lower energy consumption compared to CMOS-ASIC and recent domain wall-based design, respectively.

References

[1] Y. Wang, H. Yu, L. Ni, G.-B. Huang, M. Yan, C. Weng, W. Yang, and J. Zhao, "An energy-efficient nonvolatile in-memory computing architecture for extreme learning machine by domain-wall nanowire devices," *IEEE Transactions on Nanotechnology*, vol. 14, no. 6, pp. 998–1012, 2015.

[2] "Fact sheet: Big data across the federal government (2012)." [Online]. Available: http://www.whitehouse.gov/sites/default/files/microsites/ostp/big data fact sheet 3 29 2012.pdf

[3] X. Fong, Y. Kim, K. Yogendra, D. Fan, A. Sengupta, A. Raghunathan, and K. Roy, "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 1–22, 2016.

[4] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.

[5] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 288–301.

[6] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, "Rram-based analog approximate computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 12, pp. 1905–1917, 2015.

[7] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, and H. Yang, "Time: A training-in-memory architecture for memristor-based deep neural networks," in *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*. IEEE, 2017, pp. 1–6.

[8] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "Gpus and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, 2011.

[9] M. N. Bojnordi and E. Ipek, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 1–13.

[10] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 273–287.

[11] S. Angizi, Z. He, and D. Fan, "Parapim: a parallel processing-in-memory accelerator for binary-weight deep neural networks," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ACM, 2019, pp. 127–132.

[12] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 2016, pp. 1–6.

[13] S. Aga *et al.*, "Compute caches," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 481–492.

[14] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," *arXiv preprint arXiv:1805.03718*, 2018.

[15] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 2–13.

[16] "Everspin announces sampling of the world's first 1-gigabit mram product. 2016." [Online]. Available: https://www.everspin.com

[17] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[18] M. Imani, Y. Kim, and T. Rosing, "Mpim: Multi-purpose in-memory processing using configurable resistive memory," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 757–763.

[19] S. Angizi, Z. He, N. Bagherzadeh, and D. Fan, "Design and evaluation of a spintronic in-memory processing platform for non-volatile data encryption," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[20] H. Zhao, B. Glass, P. K. Amiri, A. Lyle, Y. Zhang, Y.-J. Chen, G. Rowlands, P. Upadhyaya, Z. Zeng, J. Katine *et al.*, "Sub-200 ps spin transfer torque switching in in-plane magnetic tunnel junctions with interface perpendicular anisotropy," *Journal of Physics D: Applied Physics*, vol. 45, no. 2, p. 025001, 2011.

[21] S. Fukami, T. Anekawa, C. Zhang, and H. Ohno, "A spin–orbit torque switching scheme with collinear magnetic easy axis and current configuration," *Nature nanotechnology*, 2016.

[22] G. Rowlands, T. Rahman, J. Katine, J. Langer, A. Lyle, H. Zhao, J. Alzate, A. Kovalev, Y. Tserkovnyak, Z. Zeng *et al.*, "Deep subnanosecond spin torque switching in magnetic tunnel junctions with combined in-plane and perpendicular polarizers," *Applied Physics Letters*, vol. 98, no. 10, p. 102509, 2011.

[23] T. Kawahara, "Challenges toward gigabit-scale spin-transfer torque random access memory and beyond for normally off, green information technology infrastructure," *Journal of Applied Physics*, vol. 109, no. 7, p. 07D325, 2011.

[24] F. Parveen, S. Angizi, Z. He, and D. Fan, "Low power in-memory computing based on dual-mode sot-mram," in *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*. IEEE, 2017, pp. 1–6.

[25] W. J. Gallagher and S. S. Parkin, "Development of the magnetic tunnel junction mram at ibm: From first junctions to a 16-mb mram demonstrator chip," *IBM Journal of Research and Development*, vol. 50, no. 1, pp. 5–23, 2006.

[26] S.-W. Chung *et al.*, "4gbit density stt-mram using perpendicular mtj realized with compact cell structure," in *IEDM*. IEEE, 2016.

[27] J. Kan, C. Park, C. Ching, J. Ahn, L. Xue, R. Wang, A. Kontos, S. Liang, M. Bangar, H. Chen *et al.*, "Systematic validation of 2x nm diameter perpendicular mtj arrays and mgo barrier for sub-10 nm embedded stt-mram with practically unlimited endurance," in *Electron Devices Meeting (IEDM), 2016 IEEE International*. IEEE, 2016, pp. 27–4.

[28] M. Bowen, M. Bibes, A. Barthélémy, J.-P. Contour, A. Anane, Y. Lemaıtre, and A. Fert, "Nearly total spin polarization in la 2/3 sr 1/3 mno 3 from tunneling experiments," *Applied Physics Letters*, vol. 82, no. 2, pp. 233–235, 2003.

[29] J. Hayakawa *et al.*, "Dependence of giant tunnel magnetoresistance of sputtered cofeb/mgo/cofeb magnetic tunnel junctions on mgo barrier thickness and annealing temperature," *Japanese Journal of Applied Physics*, vol. 44, no. 4L, p. L587, 2005.

[30] T. Tang *et al.*, "Binary convolutional neural network on rram," in *22nd ASP-DAC*. IEEE, 2017, pp. 782–787.

[31] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 541–552.

[32] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 105.

[33] Y. Wang, L. Ni, C.-H. Chang, and H. Yu, "Dw-aes: A domain-wall nanowire-based aes for high throughput and energy-efficient data encryption in non-volatile memory," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 11, pp. 2426–2440, 2016.

[34] S. Angizi, Z. He, F. Parveen, and D. Fan, "Rimpa: A new reconfigurable dual-mode in-memory processing architecture with spin hall effect-driven domain wall motion device," in *VLSI (ISVLSI), 2017 IEEE Computer Society Annual Symposium on*. IEEE, 2017, pp. 45–50.

[35] F. Parveen, Z. He, S. Angizi, and D. Fan, "Hielm: Highly flexible in-memory computing using stt mram," in *Design Automation Conference (ASP-DAC), 2018 23rd Asia and South Pacific*. IEEE, 2018, pp. 361–366.

[36] M. Zabihi, Z. Chowdhury, Z. Zhao, U. R. Karpuzcu, J.-P. Wang, and S. Sapatnekar, "In-memory processing on the spintronic cram: From hardware design to application mapping," *IEEE Transactions on Computers*, 2018.

[37] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic ram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 3, pp. 470–483, 2018.

[38] S. Yuasa, T. Nagahama, A. Fukushima, Y. Suzuki, and K. Ando, "Giant room-temperature magnetoresistance in single-crystal fe/mgo/fe magnetic tunnel junctions," *Nature materials*, vol. 3, no. 12, p. 868, 2004.

[39] (2018) Parallel thread execution isa version 6.1. [Online]. Available: http://docs.nvidia.com/cuda/parallel-thread-execution/index.html

[40] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An architecture for ultralow power binary-weight cnn acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 48–60, 2018.

[41] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[42] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

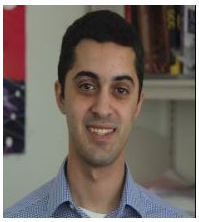
[43] S. Angizi, Z. He, F. Parveen, and D. Fan, "Imce: Energy-efficient bit-wise in-memory convolution engine for deep neural network," in *Design Automation Conference (ASP-DAC), 2018 23rd Asia and South Pacific*. IEEE, 2018, pp. 111–116.

[44] X. Fong, S. K. Gupta, N. N. Mojumder, S. H. Choday, C. Augustine, and K. Roy, "Knack: A hybrid spin-charge mixed-mode simulator for evaluating different genres of spin-transfer torque mram bit-cells," in *2011*

International Conference on Simulation of Semiconductor Processes and Devices, Sept 2011, pp. 51–54.
[45] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents
[46] X. Dong, C. Xu, N. Jouppi, and Y. Xie, "Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory," in *Emerging Memory Technologies*. Springer, 2014, pp. 15–50.
[47] S. D. C. P. V. . Synopsys, Inc.
[48] M. Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," *IEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 6, pp. 403–411, 2003.
[49] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA FPGA*. ACM, 2017, pp. 15–24.
[50] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014, pp. 609–622.
[51] R. Zhang, P. Gupta, L. Zhong, and N. K. Jha, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 24, no. 1, pp. 107–118, 2005.
[52] Y. Netzer *et al.*, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2, 2011, p. 5.
[53] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.
[54] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*. IEEE, 2012, pp. 33–38.
[55] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP Laboratories*, pp. 22–31, 2009.
[56] Z. Abid, A. Alma'Aitah, M. Barua, and W. Wang, "Efficient cmol gate designs for cryptography applications," *IEEE transactions on nanotechnology*, vol. 8, no. 3, pp. 315–321, 2009.
[57] K. Malbrain, "Byte-oriented-aes: a public domain byte-oriented implementation of aes in c," 2009.
[58] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.
[59] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 469–480.
[60] F. Parveen, S. Angizi, and D. Fan, "Imflexcom: Energy efficient in-memory flexible computing using dual-mode sot-mram," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 3, p. 35, 2018.
[61] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, K. Hsieh, K. T. Malladi, H. Zheng, and O. Mutlu, "Lazypim: An efficient cache coherence mechanism for processing-in-memory," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 46–50, 2017.
[62] S. Angizi, Z. He, and D. Fan, "Pima-logic: A novel processing-in-memory architecture for highly flexible and energy-efficient logic computation," in *Design Automation Conference (DAC)*. IEEE/ACM, 2018.
[63] L. Wang, W. Kang, F. Ebrahimi, X. Li, Y. Huang, C. Zhao, K. L. Wang, and W. Zhao, "Voltage-controlled magnetic tunnel junctions for processing-in-memory implementation," *IEEE Electron Device Letters*, vol. 39, no. 3, pp. 440–443, 2018.
[64] F. Parveen, S. Angizi, Z. He, and D. Fan, "Imcs2: Novel device-to-architecture co-design for low-power in-memory computing platform using coterminous spin switch," *IEEE Transactions on Magnetics*, vol. 54, no. 7, pp. 1–14, 2018.

Shaahin Angizi (S'15) received his B.Sc. in Computer Engineering, Hardware from South Tehran Branch of Azad University, Tehran, Iran in 2012 and his M.Sc. in Computer Engineering, Computer Systems Architecture from Science and Research Branch of Azad University, Tabriz, Iran in 2014. He is currently working toward the Ph.D. degree in Computer Engineering at University of Central Florida, Orlando, USA. His research interests include in-memory computing, deep learning, low power VLSI designs, Spin-based computing and Quantum-dot Cellular Automata (QCA).

Zhezhi He (S'16) received his B.S. degree in Information Science and Engineering from Southeast University, Nanjing, China, in 2012, and M.E. degree in Electrical and Computer Engineering from Oregon State University, Corvallis, OR, USA, in 2015. Currently he is pursuing Ph.D degree in Electrical Engineering at University of Central Florida. His research interests include neuromorphic computing, analog and mixed signal circuit design, and emerging technology.



Amro Awad is currently an Assistant Professor in the Electrical and Computer Engineering (ECE) Department at University of Central Florida (UCF). His research interests include non-volatile memory (NVM) technologies, virtual memory, hardware security and workload cloning. Before joining UCF in Fall 2017, he was a Senior Member of Technical Staff (SMTS) at the Scalable Computer Architecture group (org. 1422) in Sandia National Laboratories, Albuquerque NM. During his Ph.D., Dr. Awad had several stints at LANL, HP Labs and AMD Research. His research papers have been published in top-venues in computer architecture, such as ISCA, HPCA, ASPLOS, PACT and ICS. He was also a recipient of the Air Force Facutly fellowship for summer 2018.

Deliang Fan (M'15) received his B.S. degree in Electronic Information Engineering from Zhejiang University, China, in 2010. He received M.S. and Ph.D. degree in Electrical and Computer Engineering from Purdue University, West Lafayette, IN, USA, in 2012 and 2015, respectively. He joined the Department of Electrical and Computer Engineering at University of Central Florida, Orlando, FL, as an Assistant Professor in 2015. His primary research interest lies in Ultra-low Power Brain-inspired (Neuromorphic), Non-Boolean and Boolean Computing Using Emerging Nanoscale Devices like Spin-Transfer Torque Devices and Memristors. His other research interests include nanoscale physics based spintronic device modeling and simulation, low power digital and mixed-signal CMOS circuit design.