Processing-in-Memory for Data-Intensive Applications,

From Device to Algorithm

by

Shaahin Angizi

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

ARIZONA STATE UNIVERSITY

May 2021

ABSTRACT

Over the past decades, the amount of data required to be processed and analyzed by computing systems has been increasing dramatically to exascale (10^{18} bytes/s or ops). However, modern computing platforms' inability to deliver both energy-efficient and high-performance computing solutions leads to a gap between meets and needs, especially in resource-constraint Internet of Things (IoT) devices. Unfortunately, such a gap will keep widening mainly due to limitations in both devices and architectures. With this motivation, this dissertation's focus is on cross-layer (device/circuit/architecture/application) co-design of energy-efficient and high-performance Processing-in-Memory (PIM) platforms for implementing complex big data applications, i.e., deep learning, bioinformatics, graph processing tasks, and data encryption. The dissertation shows how to leverage innovations from device, circuit, and architecture to integrate memory and logic to break the existing memory and power walls and dramatically increase computing efficiency of today's non-Von-Neumann computing systems.

The proposed PIM platforms transform current volatile and non-volatile random access memory arrays to computational units capable of working as both memory and low-area-overhead, massively parallel, fast, reconfigurable in-memory logic. Instead of integrating complex logic units in cost-sensitive memory, the explored designs exploit hardware-friendly bit-line computing methods to implement complete Boolean logic functions between operands within a memory array in a reduced clock cycle, overcoming the multi-cycle logic issue in modern PIM platforms. Besides, new customized in-memory algorithms and mapping methods are developed to convert the crucial iteratively-used big data application's functions to bit-wise PIM-supported logic. To quantitatively analyze the performance of various PIM platforms running big data applications, a generic and comprehensive evaluation framework is presented. The overall system computing performance (throughput, latency, energy efficiency) for each application is explored through the developed framework. The device-to-algorithm co-simulation results on neural network acceleration demonstrate that the proposed platforms can obtain $36.8\times$ higher energy-efficiency and $22\times$ speed-up compared to state-of-the-art Graphics Processing Unit (GPU). In accelerating bioinformatics tasks such as biological sequence alignment, the presented PIM designs result in $\sim 2\times$, $43.8\times$, $458\times$ more throughput per Watt compared to state-of-the-art Application-Specific Integrated Circuit (ASIC), Field-Programmable Gate Array (FPGA), and GPU platforms, respectively.

To my mother Sheida who showed me the way to become what I am today

To love of my life Shadi for all her love and patience

To my brother Shayan who is always there for me

ACKNOWLEDGMENTS

I would first like to express my sincere gratitude to my advisor, Dr. Deliang Fan, whose knowledge and expertise were invaluable in formulating my Ph.D. research direction based on my interest in the past five years. Your kindness, patience, insightful advice, and feedback enlightened me, polished my thinking, and steered my work to a higher level.

To my committee members Prof. Jae-sun Seo, Prof. Amro Awad, and Prof. Wei Zhang, I am very grateful for all the help, constructive comments, and insightful suggestions in several of my research projects and my dissertation. Besides, I would like to extend my sincere acknowledgment to my kind and smart lab colleagues, Zhezhi He, Adnan Siraj Rakin, Li Yang, Fan Zhang, Amitesh Sridharan, and Farhana Parveen, for their inspiring discussion and help on our collaborative works.

Last but not least, I would like to express my appreciation to my wife, parent, and brother, who always supported me and my decisions throughout my life. This dissertation is dedicated to you.

For clarification, Chapters 2 of this dissertation contains material from “Mrima: An mram-based in-memory accelerator.” published in IEEE TCAD (2019) [1], “Aligns: A processing-in-memory accelerator for dna short read alignment leveraging sot-mram.” published in 2019 ACM/IEEE DAC [2], “Exploring a SOT-MRAM based in-memory computing for data processing.” published in IEEE TMSCS (2018) [3], “GraphS: A graph processing accelerator leveraging SOT-MRAM.” published in 2019 DATE [4], and “PIMA-logic: a novel processing-in-memory architecture for highly flexible and energy-efficient logic computation.” published in 2018 ACM/IEEE DAC [5]. The dissertation author was the investigator and author of these papers.

Chapters 3 contains material from “Redram: A reconfigurable processing-in-dram platform for accelerating bulk bit-wise operations.” published in 2019 IEEE/ACM ICCAD [6], and “Graphide: A graph processing accelerator leveraging in-dram-computing.” published in 2019 GLSVLSI [7]. The dissertation author was the main investigator and author of these papers.

Chapters 4 contains material from “IMCE: Energy-efficient bit-wise in-memory convolution engine for deep neural network.” published in 2018 ASP-DAC [8], “Accelerating deep neural networks in processing-in-memory platforms: Analog or digital approach?.” published in 2019 ISVLSI [9], and “Mrima: An mram-based in-memory accelerator.” published in IEEE TCAD (2019) [1]. The dissertation author was the main investigator and author of these papers.

Chapters 5 contains material from “Aligns: A processing-in-memory accelerator for dna short read alignment leveraging sot-mram.” published in 2019 ACM/IEEE DAC [2], “Pim-aligner: A processing-in-mram platform for biological sequence alignment.” published in 2020 DATE [10], and “PIM-Assembler: A processing-in-memory platform for genome assembly.” published in 2020 ACM/IEEE DAC [10]. The dissertation author was the main investigator and author of these papers.

Chapter 6 contains material from “Redram: A reconfigurable processing-in-dram platform for accelerating bulk bit-wise operations.” published in 2019 IEEE/ACM ICCAD [6]. The dissertation author was the main investigator and author of this paper.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

In the past decades, the amount of data required to be processed by computing systems has been dramatically increasing to exascale (10^{18} bytes/s or flops) [11], [12]. However, the incapacity of modern computing platforms to deliver both energy-efficient and high-performance computing solutions leads to a gap between meets and needs [13], [14]. 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, 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 [11], [15]. For example, the highest power efficiency of modern CPU and GPU systems is only $\sim$10GFLOPS/W, which is difficult to substantially improve in the predictable scaled technology node [16]. Second, at the architecture level, as depicted in Fig. 1a, today's computers are based on Von-Neumann architecture with separate computing and memory units connecting via buses, which leads to memory wall imposing long memory access latency, limited memory bandwidth, energy-hungry data transfer, and huge leakage power for holding data in volatile memory [14], [17]. This comes from the fact that there is a massive number of instruction fetch and data transfer between computing and memory units. Therefore, there is a great need to leverage innovations from both device and architecture to build intelligent, reconfigurable,

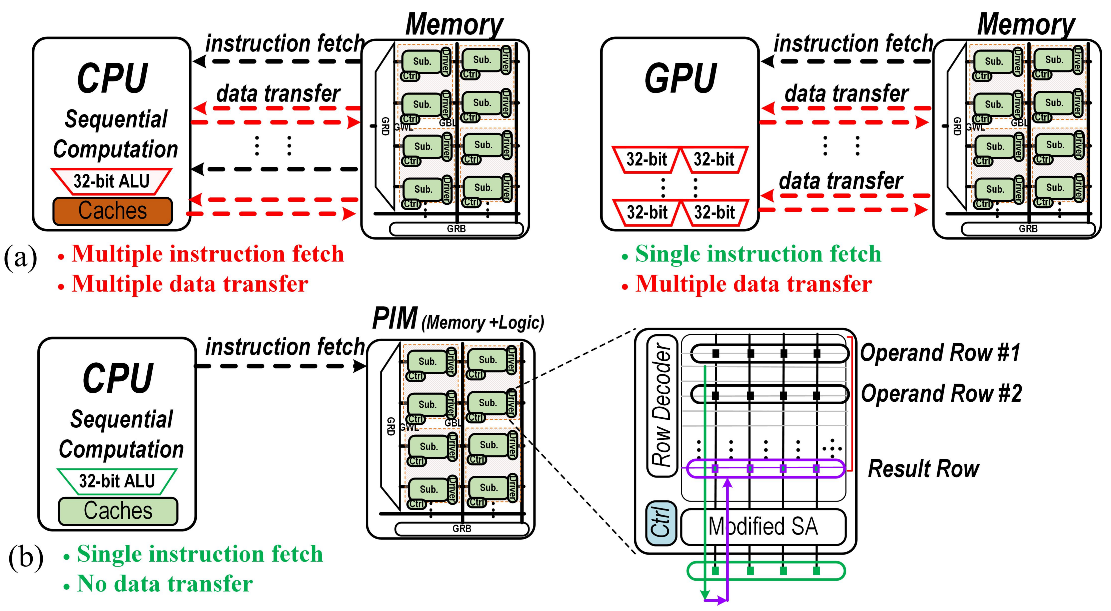


Figure 1: (a) General von-Neumann computing architecture in CPU and GPU vs. (b) Processing-in-Memory architecture.

energy-efficient, and high-performance computing platforms integrating memory and logic to break the existing memory and power walls.

In the last three decades, Processing-in-Memory (PIM) architecture, as a potentially viable way to solve the memory wall challenge, has been well explored [14], [15], [18]–[22]. The key concept behind PIM, as depicted in Fig. 1b, 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. Ideally, the PIM architectures must be capable of performing bulk bit-wise operations that are needed in many big data applications [23], [24]. Generally, at the sub-array level, a PIM holds the operand rows, e.g., #1 and #2 shown in Fig. 1b in two target rows of the memory. By receiving a particular instruction from the CPU side, the PIM's row decoder simultaneously activates the target rows and performs the bit-wise logic function between all the bit-cells in two rows, storing two operands. This could be achieved by modifying memory components at Sense Amplifiers (SA) level [23], memory bit-cell level [25], [26], or even adding combinational circuits after SA [8], [27], [28]. The proposals for exploiting SRAM-based [29], [30] PIM architectures can be found in recent literature. However, PIM in the context of main memory (DRAM- [15], [19]) 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) [31] and resistive RAM (ReRAM) [14]. 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 [31]. 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 [32]. Meanwhile, MRAM technology is in the process of commercialization [33]. Hence, PIM in the context of different NVMs, without sacrificing memory capacity, can open a new way to realize efficient in-memory computing paradigms [14], [23], [34].

1.1 Processing-in-Memory Opportunities and Challenges

1.1.1 PIM Opportunities

The PIM architecture offers two important opportunities:

- First, it can exploit the large internal memory bandwidth that gets larger when moving towards the memory bit-cell. Such bandwidth is otherwise wasted. Fig. 2a depicts the potential of internal bandwidth for DDR3-1600 and Hybrid Memory Cube (HMC). It can be observed that as moving from chip/die IO to bank row buffer, DDR and HMC achieve 57$\times$ and 222$\times$ bandwidth improvements, respectively [35], [36].
- Second, PIM eliminates data movement between the memory and processing host by performing the computation in the memory side. Fig. 2b reports energy consumption of performing integer (INT) and floating-point (FP) operations in the host as well as data movement energy between a host with a L1:32K/L2:256K/L3:4M/Main Memory:8G hierarchy at 45nm. It is reported that the data movement energy for the main memory is $\sim$100$\times$ larger than an FP operation, which shows the significance of data movement reduction.

1.1.2 PIM Challenges

There are two high-level challenges with PIM designs that need to be addressed:

- First, on the one hand, the memory industry is highly cost-sensitive. Therefore, inserting customized processing units with high-reconfigurability on the memory

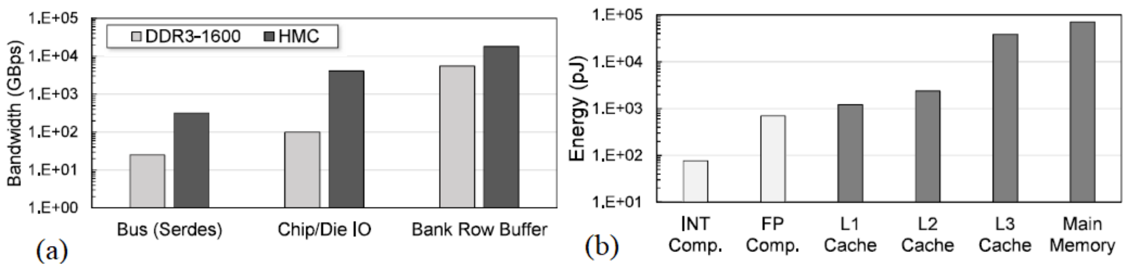


Figure 2: (a) The internal bandwidth for DDR3-1600 and HMC at chip/bank hierarchy, (b) Energy consumption for CPU data processing and data movement with a L1:32K/L2:256K/L3:4M/Main Memory:8G hierarchy at 45nm [35], [36].

side could not be accepted if it would incur large area-overhead [23]. On the other hand, for PIM to considerably reduce the data movement between memory and host processor, it has to be reconfigurable and supports a wide range of logic operations. Unfortunately, the existing PIM architectures have been limited to basic logic operations such as AND, OR, and XOR so far [23], [37], which are not necessarily applicable to a wide variety of tasks except by imposing multi-cycle operations to realize specific functions such as addition [19], [38]. This dual requires a synergic study at both device and circuit levels to realize a low-overhead and reconfigurable PIM platform.

- Second, to accelerate big data applications such as deep learning, graph processing, bioinformatics, etc., within the content of PIM, a synergic study at both architecture and algorithm levels is also needed to assure various applications can work with the provided PIM instructions. The existing big data processing algorithms are developed to work with von-Neumann computing architecture that will not necessarily fit the PIM concept. On the other side, the PIM architecture typically deals with a massive number of write-back operations that eventually may even fade the PIM benefits while working on the big data tasks. This problem is intensified when it comes to NVMs with costly write operations. Therefore, there is a great need for architecture and algorithm co-design and co-optimization on top of device and circuit levels.

1.2 Contributions

Motivated by the aforementioned opportunities and challenges, this dissertation focuses on hardware and software co-design and co-optimization of energy-efficient and high-performance PIM platforms for big data applications, leveraging innovations from circuit and architecture to integrate memory and logic to break the existing memory and power walls and to bridge memory and computing unit. The dissertation follows two main directions to address the discussed challenges, summarized in the following subsections.

1.2.1 Device-to-Architecture Co-Design for Reconfigurable PIM Logic Circuits

In this research direction, the dissertation explores how to exploit and redesign the existing NVM/VM circuits and architectures with minimal change to simultaneously work as a memory to store data and as new, intrinsic, parallel, fast, reconfigurable in-memory logic to process data within memory directly.

1.2.1.1 Processing-in-non-volatile Memories

In emerging resistive NVMs, like ReRAM and MRAM, the data are stored in terms of resistive states of memory cells. For a traditional NVM read operation, one selected memory cell will be activated and compared with a reference resistance through memory SA to read out data value. In the presented computational NVM designs, multiple resistive memory cells (i.e., data operands) could be activated and sensed simultaneously, leading to different parallel resistive levels at the SA side through modifying peripheral circuits. By carefully selecting different reference resistance levels, various Boolean logic outputs could be intrinsically 'read out' based on input operand data in the memory array. The device-to-architecture level contributions to NVMs are thoroughly discussed in Chapter 2. The evolution of proposed PIMs based on NVMs is shown starting from basic structures supporting bulk bit-wise (N)AND/(N)OR operations all the way to fully reconfigurable PIMs supporting X(N)OR and addition. The design scope and the selected publications related to this chapter are indicated in Table 1 under Technology: STT-MRAM and SOT-MRAM. Table 1 also lists the supported functions and applications regrading each work.

1.2.1.2 Processing-in-volatile Memories

In the VM domain, novel reconfigurable processing-in-DRAM platforms are designed in this dissertation, which transform current DRAM architecture to massively parallel computational units exploiting the high internal bandwidth of modern memory chips. The proposed DRAM-based designs utilize the analog operation of DRAM sub-arrays and elevate it to implement a full set of 1- and 2-input bulk bit-wise operations in a single memory cycle based on a new dual-row activation mechanism. The circuit/architecture level contributions for VMs is presented in Chapter 3. The design scope and the selected publications related to this chapter are indicated in Table 1 under Technology: DRAM.

Table 1: Taxonomy of the proposed Processing-in-Platforms.

Reference	[39]	[40]	**[1]**	[20]	[10]	[41]
Technology	DWM	DWM	STTMRAM	SOT-MRAM	SOT-MRAM	STT-/SOT-/ReRAM/ DRAM/SRAM
Supported Functions	MAJ3/MIN3 MAJ5/MIN5/Add	MAJ3/MIN3 MAJ5/MIN5/Add	full set 1-/2- input Ops./MAJ3	full set 1-/2-input Ops MAJ3	full set 1-/2-/3-input Ops MAJ3/Add	full set 1-/2-/3-input Ops MAJ3/Add
Applications	image	image/DNN	encrypt./DNN	DNN	DNA Alignment	DNN

Reference	[42]	[38]	**[5]**	**[28]**	**[2]**	**[6]**
Technology	DWM	SHEDWM	SOT-MRAM	SOT-MRAM	SOT-MRAM	DRAM
Supported Functions	MAJn/MINn	full set 1-/2- input Ops./MAJ3	(N)AND2/(N)OR2 MAJ3/MAJ5	full set 1-/2-input Ops	full set 1-/2-input Ops MAJ3/Add	full set 1-/2-input Ops MAJ3/Add
Applications	image	encrypt.	encrypt.	DNN	DNA Alignment	encrypt./graph

Reference	[43]	[44]	[45]	[46]	[47]	[48]
Technology	DWM	SHEDWM	SOT-MRAM	SOT-MRAM	SOT-MRAM	DRAM
Supported Functions	MAJ3/MIN3 MAJ5/MIN5/Add	full set 1-/2-input Ops/MAJ3	(N)AND2/(N)OR2	(N)AND2/(N)OR2 Add/Sub	full set 1-/2- input Ops	XNOR2/MAJ3/ Add
Applications	image	-	DNN	DNN	DNN	DNA assembly

Reference	[49]	[50]	**[8]**	**[4]**	**[9]**	**[7]**
Technology	SHEDWM	DWM+SOTMRAM	SOT-MRAM	SOT-MRAM	STT-/SOT-/ReRAM/ DRAM/SRAM	DRAM
Functions	MAJn/MINn/ Add	full set 1-/2-input Ops	(N)AND2/(N)OR2	full set 1-/2-/3-input Ops MAJ3/Add	full set 1-/2-/3-input Ops MAJ3/Add	(N)AND2/(N)OR2/ MAJ3/MAJ5/Add
Applications	encrypt.	graph	DNN	graph	DNN	graph

1.2.2 Big Data Applications and Algorithms

In this research direction, new customized PIM-friendly algorithms are explored for big data applications based on the proposed computational NVM and VM circuits and architectures in Chapters 2 and 3 to convert the crucial iteratively-used functions to bit-wise PIM-supported functions. Besides, new data partitioning and mapping techniques are developed to improve the PIM performance further and reduce the number of write-back operations. The main high-level contributions of this dissertation are highlighted in the following subsections.

1.2.2.1 Bottom-up Evaluation Framework

In the first part of chapter 4, a generic and comprehensive evaluation framework is presented to quantitatively analyze the performance of various PIM platforms running big data applications. As discussed, the main advantageous concept behind PIM in NVM/VM is processing massive data within memory and eliminating off-chip data communication. Thus, the overall system computing performance (throughput, latency, energy efficiency) for each application can be explored through the developed framework in this chapter. The framework is then put into the test to quantitatively compare the analog and digital PIM acceleration solutions for Deep Neural Networks (DNNs). The observations are reported considering three key evaluation metrics, i.e., area, energy, and latency.

1.2.2.2 Deep Neural Networks

In the second part of Chapter 4, a practical DNN case study will be presented to demonstrate MRIMA's [1] acceleration for binary-weight and low bit-width convolutional neural networks. The device-to-architecture co-simulation results on DNN acceleration demonstrate that MRIMA can obtain 1.7$\times$ better energy-efficiency and 11.2$\times$ speed-up compared to ASICs, and, 1.8$\times$ better energy-efficiency and 2.4$\times$ speed-up over the best DRAM-based PIM solutions.

1.2.2.3 Genome Analysis

Chapter 5 describes the PIM Acceleration of genome analysis with a focus on DNA short read alignment and DNA assembly. For the first application, by selecting AlignS [2] and PIM-Aligner [10] platforms discussed in Chapter 2, a local data partitioning, mapping, and pipeline technique are presented to maximize the parallelism in multiple PIM computational sub-arrays while conducting the alignment task. The simulation results shows that PIM-Aligner outperforms recent platforms based on dynamic programming with $\sim$3.1$\times$ higher throughput per Watt. Besides, PIM-Aligner improves the short read alignment throughput per Watt per mm^2 by $\sim$9$\times$ and 1.9$\times$ compared to FM-index-based ASIC and processing-in-ReRAM designs, respectively. For the second application, a highly parallel and step-by-step hardware-friendly DNA assembly algorithm will be developed for PANDA [51] platform that only requires the developed in-memory logic operations. The platform is then configured with a novel data partitioning and mapping technique that provides local storage and processing to utilize the algorithm-level's parallelism fully. The cross-layer simulation results demonstrate that

PANDA platform reduces the run time and power, respectively, by a factor of 18 and 11 compared with CPU. Besides, speed-ups of up-to 2.5-10$\times$ can be obtained over other recent PIM platforms to perform the same task, like STT-MRAM, ReRAM, and DRAM.

1.2.2.4 Data Encryption

The first part of chapter 6 is dedicated to PIM acceleration of data encryption application. The Advanced Encryption Standard (AES) algorithm is selected as an instance to elucidate the mapping of its transformations leveraging one of the presented PIM designs in Chapter 3, i.e., ReDRAM [6], which reveals its benefits of energy-efficiency and high-throughput for in-memory data encryption applications. The ReDRAM achieves 23% lower energy consumption compared to CMOS-ASIC implementation and requires the least number of cycles compared with other processing-in-DRAM platforms and a general purpose platform.

1.2.2.5 Graph Processing

The second part of chapter 6 discusses the PIM acceleration of graph processing applications with mapping and partitioning. We show how the ReDRAM [6] can be leveraged to greatly reduce energy consumption and latency of complex in-DRAM logic computations relying on state-of-the-art mechanisms based on triple-row activation, dual-contact cells, row initialization, NOR style, etc. As a graph processing accelerator, ReDRAM reduces energy consumption and execution time $\sim$21$\times$ and 49$\times$, respectively, compared with GPUs.

Chapter 2

RECONFIGURABLE PIM BASED ON NON-VOLATILE MEMORIES

2.1 Introduction

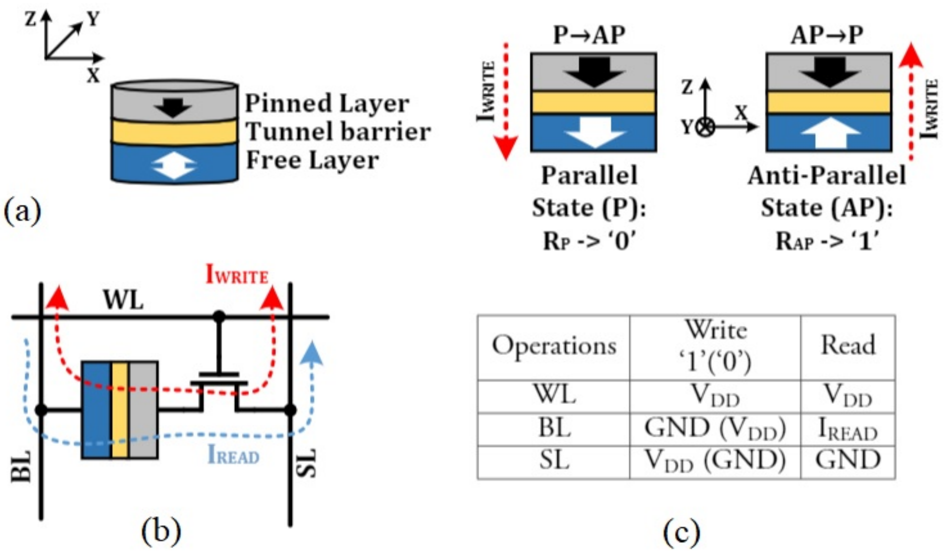
This chapter elaborates flexible, parallel, and energy-efficient PIM designs based on NVMs that can simultaneously work as a memory and realize a high-performance accelerator for both structured and non-structured data-intensive applications. Please note, while the proposed reconfigurable PIMs in this chapter leverage MRAM as the main storage unit, all new microarchitectural and circuit-level schemes presented here are tested and used in other NVMs such as ReRAM, PCM, etc.

2.1.1 Fabrication and Commercialization 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) [52], [53]. 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) [54], [55], Domain Wall Motion (DWM) device [39], [40], [42]–[44], [50], [56]–[58] and SOT-MTJ memory device [32], [45], [47], [59], [60], and Skyrmions [61]. Several companies, including IBM [62] and Everspin [33] 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 the near future [33]. Toshiba and SK Hynix co-developed a 4-Gbit STT-MRAM chip prototype and demonstrated it at IEDM 2016 [63]. In [64], a field-free switching SOT-MRAM on a 300 mm wafer was demonstrated with a reliable sub-ns switching and CMOS-compatible processes. In [65], an SOT-MRAM achieving 60-MHz write and 90-MHz was fabricated under a 55-nm CMOS process and then the first successful example of large-capacity SOT-MRAM fabrication (4 kB) on a single wafer was shown in [66]. 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 (2$\times$ more than SRAM), excellent endurance ($\sim 10^{15}$ cycles [67]), and compatibility with the CMOS fabrication process (back end of the line) [54].

2.1.2 STT-MRAM

A typical Magnetic Tunnel Junction (MTJ) structure, as shown in Fig. 3a, consists of two ferromagnetic layers with a tunnel barrier sandwiched between them. Due to the Tunnel MagnetoResistance (TMR) effect [68]–[71], 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}-R_P)/R_P, which may vary from 10% to 400% depending on materials and temperature [68]–[70], [72]. Thus, data are stored as the magnetization direction in the free layer, which can be flipped through current-



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

Figure 3: (a) Device structure of conventional Magnetic Tunnel Junction (MTJ) in parallel and anti-parallel states, with Spin-Transfer Torque (STT) switching scheme. (b) 1T1R STT-MRAM, (c) Biasing condition for memory operations.

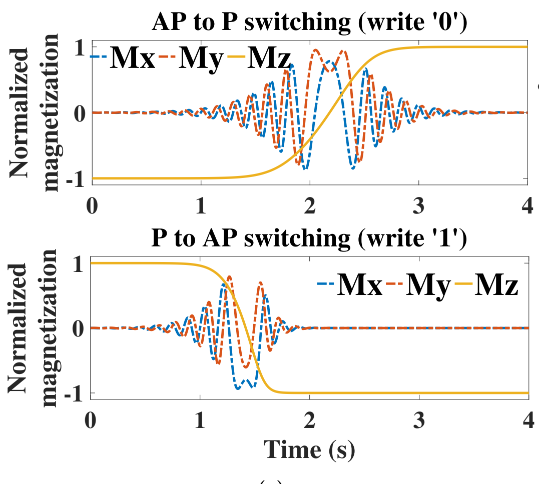
induced Spin-Transfer Torque (STT). Note that, the MTJ with Perpendicular Magnetic Anisotropy (PMA) is used in this dissertation. The 1T1R memory bit-cell is widely used in the typical MRAM design, as depicted in Fig. 3b, which is controlled by Bit Line (BL), Word Line (WL), and Source Line (SL). The biasing conditions of memory read/ write are presented in Fig. 3c. For both memory read and write operations, the WL is enabled, which turns on the access transistor. To write a data in a memory cell, the corresponding WL is activated using a Memory Row Decoder (MRD). Then appropriate voltage difference (Fig. 3c) is applied to the corresponding BL and SL using the Write Driver (WD) connected to them (the write current path is shown in Fig. 3b), leading to MTJ resistance in High-R_{AP} (/Low-R_P). For memory read, a sensing current (I_{READ}) is applied on the BL and consequently generates a sensing voltage, which can be detected by a Sense Amplifier (SA).

For the STT-MRAM modeling in this dissertation, the Non-Equilibrium Green's Function (NEGF) and Landau-Lifshitz-Gilbert(LLG) equation are used before the circuit-level simulation. The magnetization dynamics of MTJ's Free Layer-FL (m) can be modeled as [73], [74]:

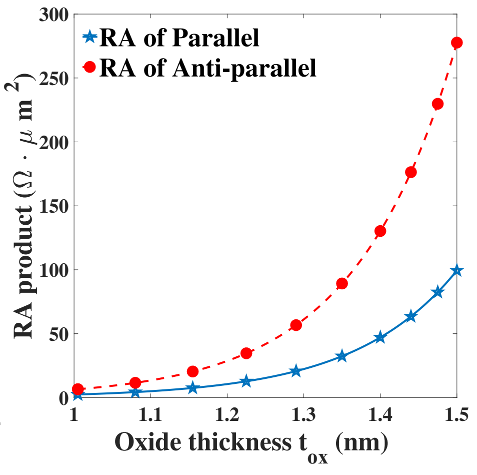
$$\frac{dm}{dt} = -|\gamma| m \times H_{\text{eff}} + \alpha\left(m \times \frac{dm}{dt}\right) + |\gamma|\beta(m \times m_{\text{p}} \times m) - |\gamma|\beta\epsilon'(m \times m_{\text{p}}) \quad (2.1)$$

$$\beta = |\frac{\hbar}{2\mu_0 e}| \frac{I_{\text{c}} P}{A_{\text{MTJ}} t_{\text{FL}} M_s} \quad (2.2)$$

where $\hbar$ is the reduced plank constant, γ is the gyromagnetic ratio, I_{c} is the charge current flowing through MTJ, t_{FL} is the thickness of free layer, ϵ' is the second Spin transfer torque coefficient, and H_{eff} is the effective magnetic field. P is the effective polarization factor, A_{MTJ} is the cross sectional area of MTJ, and m_{p} is the unit polarization direction. Fig. 4a shows the normalized magnetization dynamics of free layer in x-, y- and z-axis, when performing the STT-MRAM write scheme as described earlier [75].



(a)



(b)

Figure 4: (a) The normalized magnetization switching in x-, y- and z-axis. (b) The Resistance-Area product w.r.t the thickness of MTJ tunnel oxide (t_{ox}) [75].

Based on the simulation parameters listed in Table 2, the magnetization dynamic from LLG equation can provide the relative angle θ between the magnetization of Pinned Layer-PL ($\hat{z}$) and FL (m). Therefore, the real-time conductance of MTJ (G_{MTJ}) is given by [76]:

$$G_{\mathrm{MTJ}} = \frac{G_{\mathrm{P}} + G_{\mathrm{AP}}}{2} + \frac{G_{\mathrm{P}} - G_{\mathrm{AP}}}{2} \cos\theta \tag{2.3}$$

where G_{P} and G_{AP} are the conductance of MTJ in parallel ($\theta = 0$) and anti-parallel ($\theta = 180$) configurations. Both G_{P} and G_{AP} are obtained from the atomistic level simulation framework based on Non-Equilibrium Green's Function (NEGF) [77], while the Resistance-Area Product with respect to the thickness of MTJ tunnel oxide is shown in Fig. 4b.

Table 2: Simulations Parameters for MTJ.

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

2.1.3 SOT-MRAM

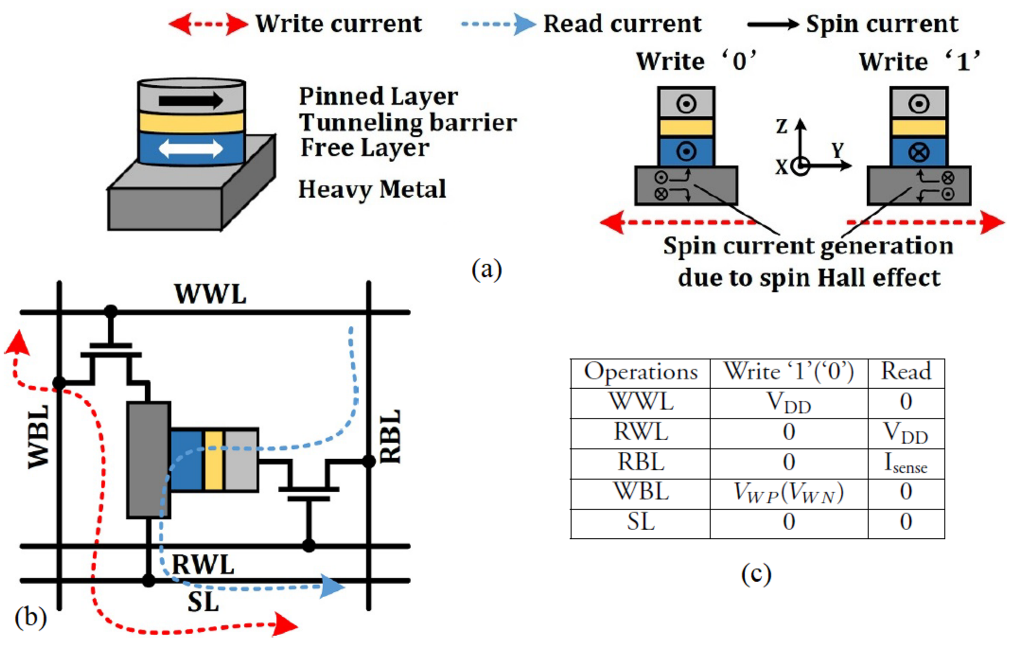
As shown in Fig. 3b, in the typical STT-MRAM design, only one access transistor is used for both memory write and read, which suffers several limitations due to the intrinsic device physics and structure, including long write latency (>10-ns); high write current (>2 MA/cm^2) and thus large writing power and area (due to large transistor sizing); shared read and write paths causing read-write conflict; asymmetric writing of data '0' and '1' due to different spin polarization factor of fixed and free ferromagnetic layers; reliability concern due to tunnel oxide breakdown in large write voltage [78], [79].

In order to address the above limitations of STT-MRAM, the recent application of SOT has been explored to switch the adjacent MTJ free layer magnetization (i.e., programming MTJ resistance) much more energy efficiently in I/FM/HM structure (I: Insulator, FM: Ferro-Magnet, and HM: Heavy Metal) [3], [53]. Fig. 5a presents the device structure of SOT-MTJ, which is an MTJ mounted on a heavy metal substrate. When electrons flow through the non-magnetic heavy metal substrate (in the $\pm$y direction) with strong spin-orbit coupling, the electrons with the reverse direction of rotation accumulate on the opposite surfaces of HM. Thus, a pure spin current (I_s) in the $\pm$z direction is generated, which exerts an SOT on the adjacent FM and switches the magnetization. The relationship between the generated spin current (I_s) and the applied charge current (I_c) can be expressed as:

$$I_s = P_{she}(\sigma \times I_c) \tag{2.4}$$

$$P_{she} = \frac{I_s}{I_c} = \frac{A_{FM}}{A_{HM}}\theta_{sh}\Big(1 - sech\big(\frac{t_{HM}}{\lambda_{sf}}\big)\Big) \tag{2.5}$$

where P_{she} is spin Hall injection efficiency. σ is the electron spin polarization, which is transverse to both the spin current and charge current directions. A_{FM} is the area of the adjacent FM area and A_{HM} is the cross-sectional area of HM in the direction of current flow. θ_{sh} is the spin Hall angle, which is defined as the ratio of generated spin current density to the applied charge current density. t_{HM} is the thickness of HM substrate, and λ_{sf} is the spin flip length. Recently, large spin Hall angle was experimentally demonstrated in different heavy metal materials, such as Pt [80], β-Ta [81], β-W [82],



Operations	Write '1'('0')	Read
WWL	V_{DD}	0
RWL	0	V_{DD}
RBL	0	I_{sense}
WBL	$V_{WP}(V_{WN})$	0
SL	0	0

(c)

Figure 5: (a) The stacking device structure of MTJ and heavy metal substrate, which uses spin-orbit torque induced magnetization switching scheme. (b) Bit-cell schematic of SOT-MRAM with two access transistors (1R/1W). (c) Biasing condition for memory operations.

and CuBi alloys [83]. High magnetization switching speed (<1ns) of SOT-MTJ is achieved mainly due to larger spin injection efficiency compared to the conventional MTJ with an STT-switching scheme. Therefore, it is much more efficient to choose SOT-induced switching scheme as the next generation MRAM design.

Fig. 5b shows the corresponding 2T1R SOT-MRAM bit-cell design with separated write and read access transistors, correspondingly controlled by Write Bit Line (WBL), Write Word Line (WWL), Read Bit Line (RBL), Read Word Line (RWL), and the shared Source Line (SL). The memory read and write biasing conditions are presented in Fig. 5c. For memory write, WWL is pulled high, which turns on the write access transistor. Then, to write '1' (or '0'), a positive voltage V_{WP} (or negative voltage V_{WN}) is applied to WBL with SL connected to ground. For memory read, RWL is set to

Table 3: Simulation Parameters for SOT-MTJ.

Parameter	**Value**
Free layer dimension,$(W \times L \times t)_{FM}$	$60 \times 40 \times 2\ nm^3$
SHM dimension, $(W \times L \times t)_{HM}$	$60 \times 80 \times 2\ nm^3$
Demagnetization Factor, D_x, D_y, D_z	$0.066, 0.911, 0.022$
Spin flip length, λ_{sf}	$1.4\ nm$
Spin hall angle, θ_{sh}	0.3
Gilbert Damping Factor, α	0.007
Saturation Magnetization, M_s	$850\ kA/m$
Oxide thickness, t_{ox}	$1.2\ nm$
RA product, RA_p / TMR	$10.58\ \Omega \cdot \mu m^2$ / 171.2%
Supply voltage	$1\ V$
CMOS technology	$45\ nm$
SOT-MRAM cell area	69F^2
Access transistor width	4.5F
Cell aspect Ratio	1.91

V$_{\text{DD}}$, and the read access transistor is switched on. A sensing current (I$_{\text{sense}}$) flowing through SOT-MTJ consequently generates a sensing voltage (V$_{\text{sense}}$) on RBL, which can be detected by the SA.

The magnetization dynamics of SOT-MTJ's FL (m) can be also modeled by the modified LLG equation, which can be mathematically described as:

$$(1+\alpha^2)\frac{dm}{dt} = -|\gamma|\mu_0 m \times H - \alpha|\gamma| m \times m \times H - m \times m \times \frac{I_s}{qN_s} + \alpha m \times \frac{I_s}{qN_s} \tag{2.6}$$

where α is Gilbert damping factor, γ is the gyromagnetic ratio, and μ_0 is the vacuum permeability. H is the effective field, which includes dipolar coupling field, demagnetization field, thermal noise field and anisotropy field. $N_s = M_s V/\mu_B$ is the number of spins, μ_B is Bohr magneton, M_s and V are the saturation magnetization and volume of ferromagnet, respectively. The simulation parameters are listed in Table 3. In order to realize the desired 1ns switching speed, about 130μA writing current is required which leads to 1V and -0.35V for V$_{\text{WP}}$ and V$_{\text{WN}}$, respectively. Based on the simulation parameters listed in Table 3, the magnetization dynamic from LLG equation can provide the relative angle θ between the magnetization of PL and FL. Therefore, the real-time conductance of MTJ (G_{MTJ}) is given by the Eq. (2.3), where again both G_P and G_{AP} are obtained from the atomistic level simulation framework based on Non-Equilibrium Green's Function (NEGF) [77], while the Resistance-Area Product with respect to the thickness of MTJ tunnel oxide is listed in Table 3.

2.1.4 Challenges

In this subsection, the main limitations and challenges of recent PIM platforms are discussed:

- First, most recent PIM designs offer application-specific acceleration circuits/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 [14], [34], [84], [85] have been widely used to accelerate Convolutional Neural Networks (CNNs). Ambit [19] and Pinatubo [23] as recent in-memory accelerators enable only bulk bit-wise in-memory operations tailored for data-intensive applications. DRISA [15], Compute Cache [29], and CMP-PIM [28] 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 [86], RIMPA [49], and HieIM [87] target for designing in-memory encryption engines by developing efficient in-memory XOR units.
- Second, current PIM schemes unavoidably rely on external processing units for performing more complex logic operations, otherwise PIM's performance degra-

dation 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 a processor than a PIM platform. Recent in-memory addition techniques [38], [49], [88] do not show acceptable performance specially for multi-bit addition. The STT-CiM [89] 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 carryout bit back to the memory and use it for computation of next bits.

- Third, none of these designs can perform computing (Boolean logic functions) between any two bits irrespective of their locations in the memory array. Processing data (operands) can be stored in different memory locations with distinct physical addresses. Therefore, existing bit-wise PIM schemes unavoidably impose multi-cycle operations to align operands in the same column [23], [29], [89] or row [49], [90] to process data within memory. For instance, RIMPA [49]/Pinatubo [23] require at least 2 cycles (read/write) to line operands in the same row/column to realize a 2-input in-memory AND function. Thus, such *operand-locality* issue is a very important un-addressed topic in previously reported PIM architectures [29].

We explore and address the first two PIM challenges in Section 2.3 and the third challenge in Section 2.4.

2.2 Overall PIM Architecture

The general memory organization to realize PIM in NVMs is shown in Fig. 6. The main memory chip is basically divided into multiple Banks. Each bank consists of

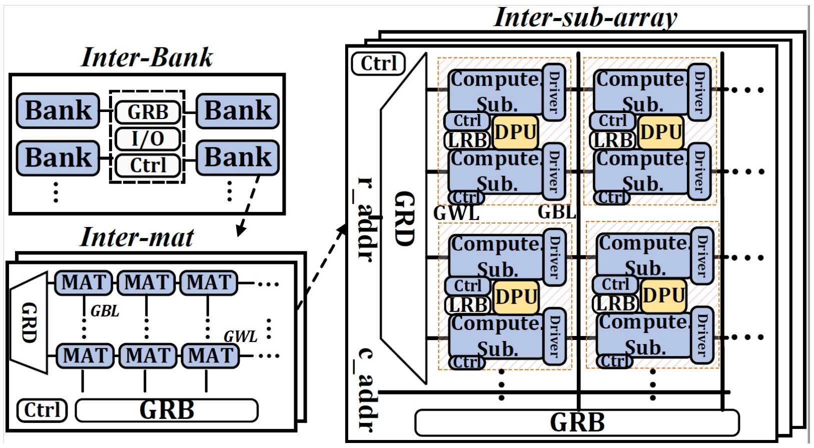


Figure 6: The overall PIM architecture used in Chapter 2.

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 computational memory sub-arrays (i.e., PIM-enhanced sub-array) connected to a GRD and GRB.

According to the application type and physical address of operands within memory, the PIM's Controller (Ctrl) can 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. 7 gives an overview of the PIM's acceleration steps. Assume input tensors A and B (that can belong to various 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. The

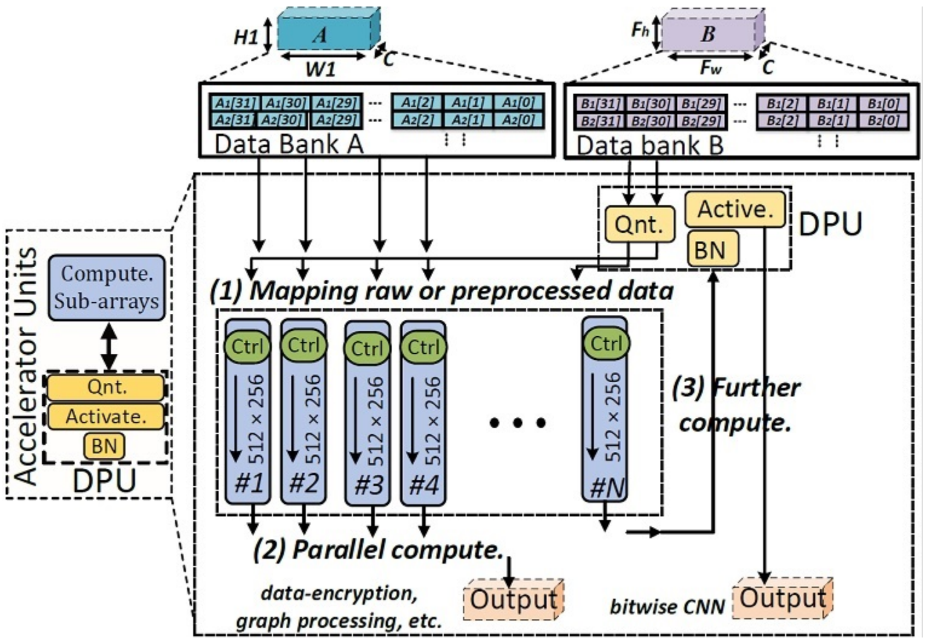


Figure 7: The PIM's acceleration steps. The size of the computational sub-arrays could be tailored.

results at this step 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.

2.3 Evolution of the Proposed MRAM-based PIM Platforms

2.3.1 Basic PIM Supporting (N)AND, (N)OR

In emerging resistive NVMs, like MRAM and ReRAM, the data are stored in resistive states of memory cells as discussed in Sections 2.1.2 and 2.1.3. In the traditional NVM's read operation, one selected memory cell will be activated and compared with a reference resistance through memory SA to read out data value. Therefore, firstly,

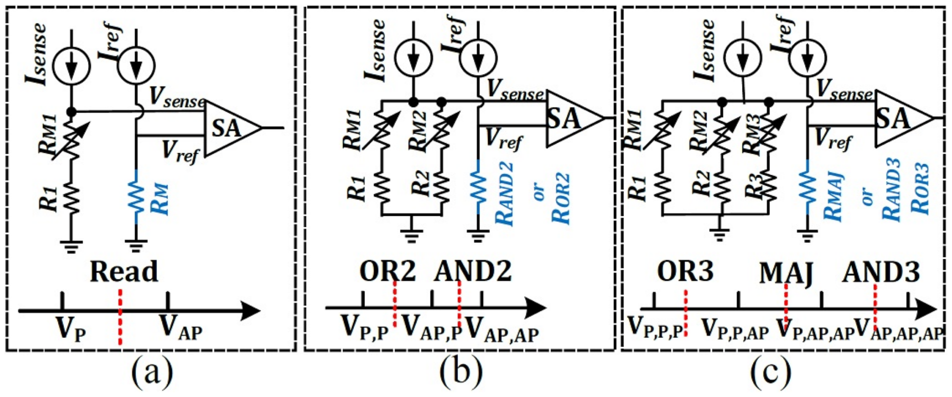


Figure 8: The idea of voltage comparison between $\mathrm{V_{sense}}$ and V_{ref} for (a) memory read, (b) 2-input in-memory logic, i.e., IML2x, and (c) 3-input in-memory logic, i.e., IML3x. Note that, R_{Mi} and R_i denote the equivalent resistance of the non-volatile component and selecting transistor, wire, etc. respectively.

the corresponding WL(/RWL) is activated using the MRD and the corresponding BL(/RBL) is connected to the SA using the Memory Column Decoder (MCD) (the read current path is shown in Fig. 3b). The idea of voltage comparison for memory read is shown in Fig. 8a, a single cell is addressed to generate a sense voltage ($\mathrm{V_{sense}}$), which will be compared with memory mode reference voltage activated by an enable signal EN_M ($\mathrm{V_{sense,P}<V_{ref,M}<V_{sense,AP}}$). 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(/RBL) is considered in the whole dissertation to maximize the output bandwidth.

With a careful study of this operation, new peripheral circuits are designed in this chapter such that multiple resistive memory cells (i.e., data operands) could be activated and sensed simultaneously, leading to different parallel resistive levels at the SA side. In this way, by carefully selecting different reference resistance levels, various Boolean logic outputs could be intrinsically 'read out' based on input operand data in the memory array.

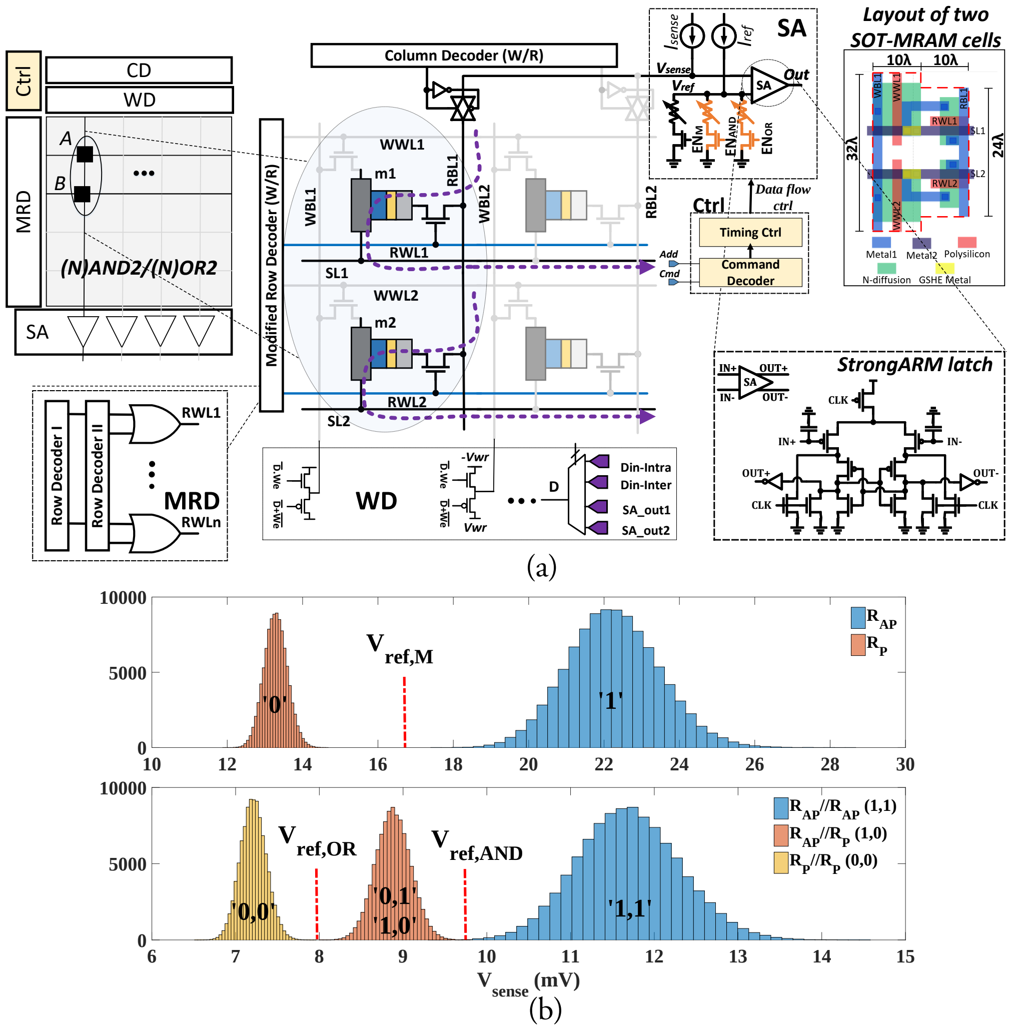


Figure 9: (a) Proposed PIM sub-array architecture based on SOT-MRAM supporting (N)AND, (N)OR functions with peripherals [8], [59], [91]. The layout of two adjacent SOT-MRAM cells is also indicated. (b) Monte-Carlo simulation result of the sense voltage (V_{sense}) distribution.

The first idea was rather simple [8], [59], [91], where every two bits stored in the identical column could be selected and sensed simultaneously, as depicted in Fig. 9a. To do this, the MRD was modified to support multi-line enable function through combining two single-line enable decoders with their outputs connected to OR gates. To activate the computing current path as shown in Fig. 9a for the first column, RWL1 and RWL2 are activated by the MRD while SL1 and SL2 are grounded and all the other WLs and SLs are kept deactivated. The MCD/CD activates the RBL1 to be connected to the SA. Now the sense (read) current is applied to RBL1 and with that the equivalent resistance voltage of such parallel connected SOT-MRAMs (m1 and m2) and their cascaded access transistors can be compared with a specific reference voltage generated by SA. Through selecting different reference resistances by new enable signals (EN_M, EN_{AND}, EN_{OR}) as shown in SA box in Fig. 9a, the SA can perform basic memory and in-memory Boolean functions (i.e., (N)AND2 and (N)OR2). For (N)AND2 operation, R_{ref} is set at the midpoint of $R_{AP}//R_P$ ('1','0') and $R_{AP}//R_{AP}$ ('1','1') as shown in Fig. 8b. Thus only when both of the selected MRAM bit-cells are in an anti-parallel state (i.e., binary input: '1', '1'), the output is high, whereas the output is low. Similarly, for (N)OR2 operation, R_{ref} is set at the midpoint of $R_P//R_P$ and $R_P//R_{AP}$ and only when both of the two selected MRAM bit-cells are in the parallel state (i.e., binary input: '0', '0'), the output is low, whereas the output is high.

To validate the sense circuit's variation tolerance, we have performed a Monte-Carlo simulation with 100000 trials. A $\sigma = 5\%$ variation is added on the Resistance-Area product (RA_P) and a $\sigma = 10\%$ process variation is added on the TMR. The simulation result of sense voltage (V_{sense}) distributions in Fig. 9b shows the sense margin of in-memory computing. It will be reduced by increasing the logic fan-in (i.e., number of parallel memory cells). It is worth pointing out that this design does not necessarily rely on a certain NVM technology or cell structure. As long as the technology is based on resistive-cell, i.e., PCM and ReRAM, the presented SA can readily perform in-memory computation. Based on our experiments, leveraging PCM and ReRAM cells (with higher ON/OFF ratio) leads to a significantly larger read margin compared with SOT-MRAM, which further translates to much higher reliability even by activating more number of rows (e.g., up to 64-row operation for PCM [23]). Therefore, it is possible to use other types of emerging NVMs to achieve a better read margin. Notwithstanding, PCM and ReRAM consume more power than SOT-MRAM if converted to the PIM platform mainly due to their relatively higher writing power, which inevitably causes overall power increase when dealing with complex real-world applications requiring massive intermediate operand data write-back into memory.

While the proposed PIM design in Fig. 9a could implement any in-memory Boolean logic functions based on universal NAND2/NOR2 functions, it requires multiple cycles. It means the operation's result has to be written back into the memory after each memory cycle. Such write-back operation reduces the platform's performance and energy-efficiency in computationally-intensive big data applications and eventually may even fade the PIM advantages. This motivated us to move forward and design reconfigurable complete PIM platforms supporting more Boolean functions.

2.3.2 Reconfigurable Complete PIM Supporting X(N)OR

In [3], [28], an enhanced and reconfigurable PIM platform on top of the previous design is proposed. In the new design, every RBL is routed to a Modified Sense Amplifier (MSA), as shown in Fig. 10. The new MSA consists of two sub-SAs and three reference resistors compared to the first design with one SA in Section 2.3.1. Every two bits stored in the identical column can be selected with the MRD and sensed simultaneously as shown in Fig. 10a. Again, the equivalent resistance of such parallel SOT-MRAMs and their cascaded access transistors is compared with MSA's programmable reference. In the new design, through selecting two reference resistances (i.e., EN_{AND}, EN_{OR}), two sub-SAs can operate simultaneously to realize two basic in-

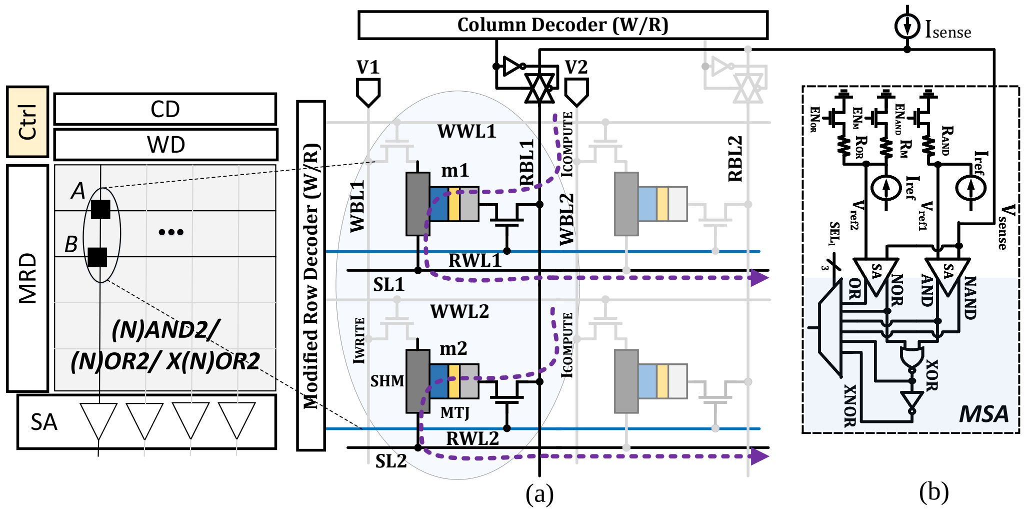


Figure 10: (a) Proposed in-memory processing sub-array architecture based on SOT-MRAM supporting (N)AND, (N)OR, X(N)OR functions [3], [28], (b) Modified Sense Amplifer with two sub-SAs and three reference resistors.

memory Boolean functions, i.e., (N)AND2 and (N)OR2 at the same time, as shown in Fig. 10b. This provides more flexibility to the PIM to implement more complex logic functions through combining the outputs. The X(N)OR2 logic can be realized with two sub-SA's outputs (AND2 and NOR2 logic) with an extra CMOS NOR2 gate after the outputs in the MSA. As shown in Fig. 10b, the operation of such sense circuit is determined by the control signals (EN_{AND}, EN_M, EN_{OR}), while the desired result is acquired by the select signal (SEL) of the output multiplexer [28], [92]. It is noteworthy that only one SA is used during (N)AND2/(N)OR2/memory read operation to reduce the power consumption of sensing. Parallel computing/read is implemented by using one SA per bit-line.

Fig. 11 depicts the transient simulation result of the sense circuit under a 2ns period clock signal (CLK), which takes the data stored in MRAM1 (m1) and MRAM2 (m2) as inputs. When CLK is high, the sense amplifier is in the pre-charge phase, and the output is reset to '0'. When CLK is low, the sense amplifier is in the sampling phase and generates logic computation results depending on the reference voltage configuration. V_{cmp} plots the comparison between sense voltage (V_{sense}) and two reference voltages, i.e., V_{ref1} and V_{ref2}. Again, V_{ref1} is set to $(V_{AP,AP}+V_{AP,P})/2$, and V_{ref2} is set to $(V_{P,P}+V_{AP,P})/2$, for performing AND2 and OR2, respectively. It is noteworthy that V_{cmp}'s ripple comes from the kickback noise of the SA's clock switching. In general, the transient curve demonstrates the correct logic operation of MSA with ~200 ps output latency.

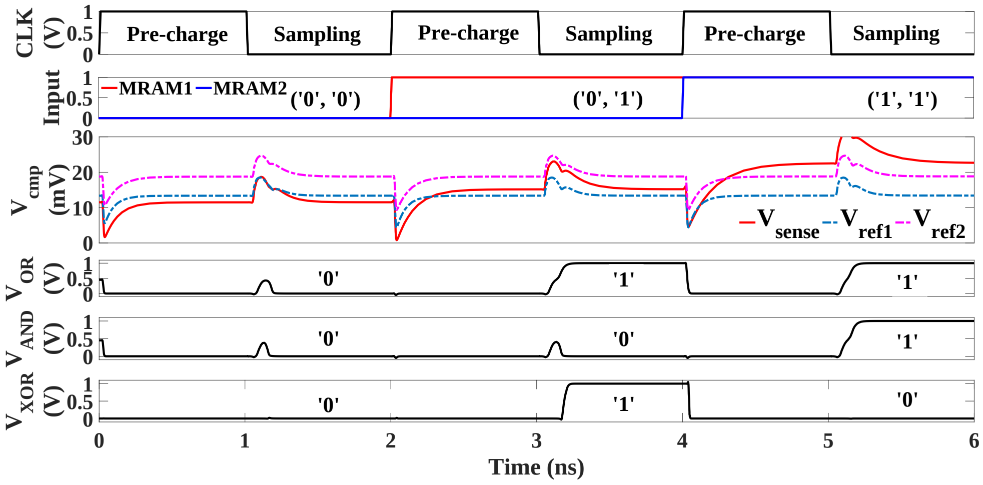


Figure 11: Transient simulation results of in-memory computing operations (i.e., AND, OR and XOR) [3].

2.3.3 Reconfigurable PIM Supporting Two-Cycle In-Memory Addition

2.3.3.1 Design I: MRIMA based on STT-MRAM

Aiming to provide more flexibility and reconfigurability for the PIM platforms, a new PIM sub-array architecture based on STT-MRAM, named MRIMA was presented in [1]. This in-memory circuit design, as depicted in Fig. 12a, mainly consists of

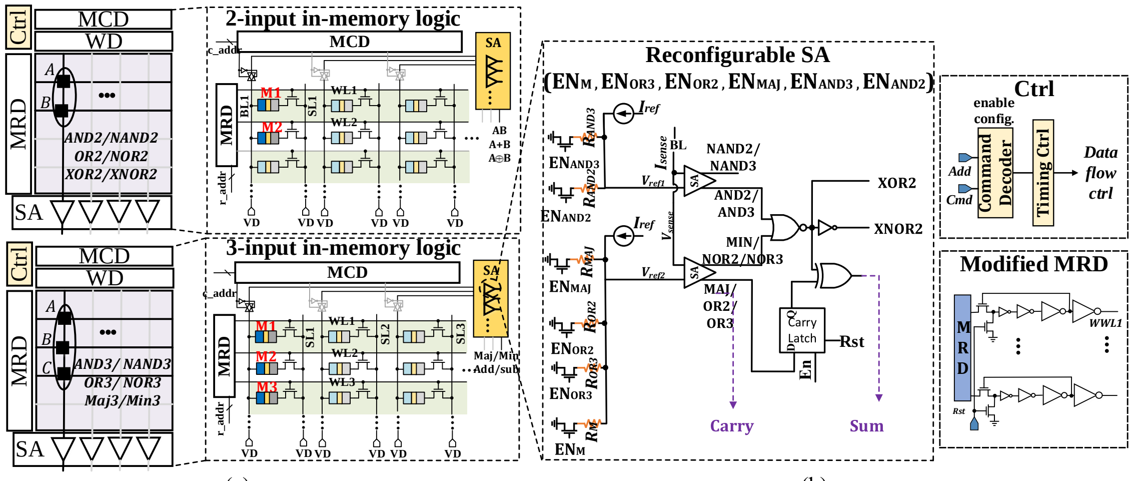


Figure 12: The MRIMA's sub-array architecture [1]: (a) Block level scheme and STT-MRAM realization of 2-input and 3-input in-memory logic methods, (b) Peripherals of computational sub-arrays to support computation.

Write Driver (WD), MRD (elaborated in Fig. 12b), MCD, SA (Fig. 12b), and can be adjusted by Ctrl unit (Fig. 12b) to work in dual mode that perform both memory write/read and bit-line computing. The proposed reconfigurable SA, as depicted in Fig. 12b, 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 4. Such reconfigurable SA could implement memory read and one-threshold based logic functions on top the discussed bit-line computing scheme by activating one enable at a time, e.g., by setting EN_{AND2} to '1', (N)AND2 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 X(N)OR2, as in Section 2.3.2. Here, we elaborate on the main functions supported in MRIMA.

Table 4: Configuration of MRIMA's enable bits for different functions.

Ops.	read/NOT	(N)OR2/NOR2	(N)AND2	X(N)OR2	MAJ/MIN	(N)OR3	(N)AND3
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

2.3.3.1.1 Fast row copy (FRC)

MRIMA's FRC mechanism needs 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. 6); in the second half-cycle, the data stored in the 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.

2.3.3.1.2 Two-input in-memory logic (IML2x)

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. The IML2x is essentially the same as the 2-input PIM operation in the previous designs, where, every two bits stored in an identical column can be selected employing the MRD and sensed simultaneously, as depicted in Fig. 12a. The equivalent resistance of such parallel connected STT-MRAMs and their cascaded access transistors is 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., (N)AND2 and (N)OR2) in a single memory cycle. The idea of voltage comparison between V_{sense} and V_{ref} for IML2x is shown on Fig. 8b. The XOR2 logic is also realized with two SAs and an additional CMOS NOR gate similar to the presented design in Fig. 10b.

2.3.3.1.3 Three-input in-memory logic (IML3x)

In the IML3x, every three cells located in an identical column can be selected by MRD and sensed simultaneously to realize 3-input logic functions (i.e., (N)AND3, (N)OR3, MAJ/MIN). For instance, consider the data organization shown in Fig. 12a, 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. 8c, 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. 8 denote the equivalent resistance of selecting transistor, wire, etc. cascaded within the sensing path. In our experiment, the average value across the memory array was taken, since normally the equivalent resistance depends on the location of the selected memory cell.

A comprehensive study on the MRIMA's sensing circuit's variation tolerance is done by running the Monte-Carlo simulation with 10000 trials. A $\sigma = 2\%$ variation is added to the RA_P, and a $\sigma = 5\%$ process variation (typical MTJ conductance variation [13]) is added on the TMR. The simulation result of V_{sense} distributions in Fig. 13 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 three. Such sense margin could be even improved by either increasing the sense current or oxide

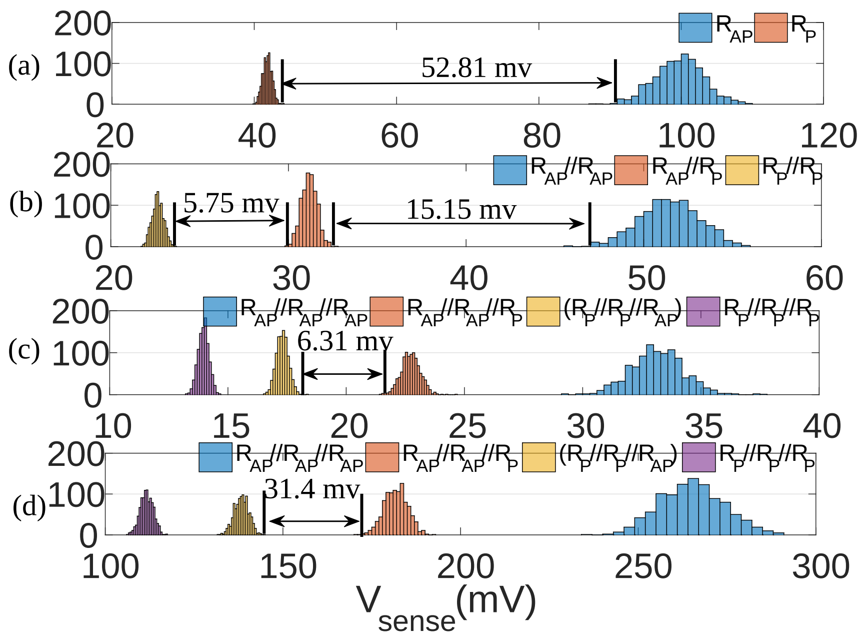


Figure 13: Monte-Carlo simulation of V$_{\text{sense}}$ (with RA$_{\text{P}}$/TMR=2%/5% - t_{ox}=1.5nm) for (a) memory read, (b) IML2x, (c) IML3x when $I_{sense} = 6.6\mu A$, and (d) IML3x when $I_{sense} = 18\mu A$ [1].

thickness (t_{ox}), but obviously by sacrificing the operation's energy-efficiency. To show this, the sense current (I_{sense}) is increased from the initial value ($\sim 6.6\mu A$), plotted in Fig. 13c, to $\sim 18\mu A$ and the simulation for only IML3x was re-run to plot Fig. 13d. By increasing 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 the sensing current is not increased above $20\mu A$ to make sure there is no read-write conflict.

To further explore the correlation between I_{sense} and voltage margin for different MRIMA's operations, Fig. 14a shows the voltage margin for memory read, IML2x, and IML3x operations when the I_{sense} is gradually increased. As can be seen, the larger I_{sense} is, the larger voltage margin is achieved for different operations. In addition, the IML3x's voltage margin considering different stochastic variations on MTJ's RA$_{\text{P}}$/TMR

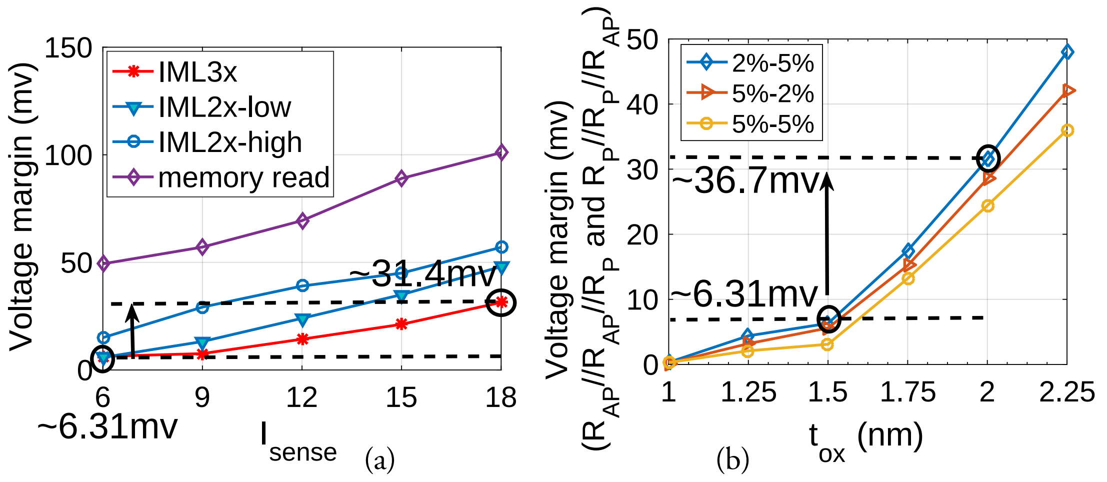


Figure 14: (a) Voltage margin between sensitive states of MRIMA's operations vs. I_{sense} with a t_{ox}=1.5nm, (b) Voltage margin of IML3x operation vs. thickness of MTJ oxide with different variations on RAP/TMR with a $I_{sense} = 6.6\mu A$.

(2%/5%, 5%/2%, and 5%/5%) is explored in Fig. 14b by increasing t_{ox}, from 1nm to 2.25nm (as experimentally-demonstrated in [93]). 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.

2.3.3.1.4 Two-cycle in-memory addition

In addition to the above-mentioned single-cycle logic operations, MRIMA's sub-array can perform addition/ subtraction (add/sub) operation quite efficiently. In the full-adder Boolean logic, the carry-out can be directly produced by MAJ function (Carry in Fig. 12b) just by setting EN_{MAJ} to '1'. Accordingly, a carry latch is inserted at this point to store intermediate carry outputs to be used in the summation of the next bits. Meanwhile, Sum output can be obtained by inserting a 2-input XOR gate in the reconfigurable SA, taking the latch output and in-memory XOR2 output as the inputs. Now, assume A, B, and C operands (in Fig. 12a), IML2x and IML3x are able to generate Sum (/Difference) based on XOR3 and Carry (/Borrow) bits, and perform parallel multi-bit addition operation as will be delineated in Chapter 4.

2.3.3.1.5 System integration

While MRIMA is meant to be an independent high-performance and energy-efficient accelerator, it needs to be expose it to programmers and system-level libraries to utilize it. From a programmer's 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. Accordingly, the programs are translated at install time to the MRIMA hardware instruction set tabulated in Table 5. The micro and control transfer instructions are not shown in the table.

Table 5: 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 than the later one; accessing MRIMA as an I/O device can incur significant overheads due to interrupts and page faults (in the shared memory model). In contrast, a memory-mapped MRIMA scheme can cause significant contentions in the memory bus if the processor executes memory-intensive applications simultaneously. Choosing the scheme of integrating MRIMA is left to system architects based on their workloads and use-cases. In both schemes for integrating MRIMA, the commands/instructions that MRIMA architecture accepts are similar and based on the ISA.

2.3.3.1.6 Non-structured bulk benchmark evaluation

The logic performance of MRIMA compared to recent PIM platforms is analyzed taking intrinsically-non-structured ISCAS85 benchmarks. A logic netlist in Berkeley Logic Interchange Format (.blif) is fed into ThrEshold Logic Synthesizer (TELS) [94] 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 to assess the performance[1]. Fig. 15 gives energy and delay of ISCAS85 combinational circuit benchmarks implemented using MRIMA, Pinatubo [23], STT-CiM [89], RIMPA [49], HieIM [87], and Ambit [19]. To have an impartial comparison, Pinatubo, a general system architecture for NVMs, is implemented with the same standard STT-MRAM, SOT-MRAM, and ReRAM technologies.

Based on the figure, MRIMA spends the lowest energy and delay compared to the counterparts in different benchmarks. (1) MRIMA reduces the energy consumption by ~72%, 61.2%, 75.5%, and 86.2% compared to Pinatubo-STT [23], STT-CiM [89], HieIM [87], and Ambit [19], respectively. This considerable improvement mainly

[1]The full bottom-up evaluation framework is explained in Chapter 4.

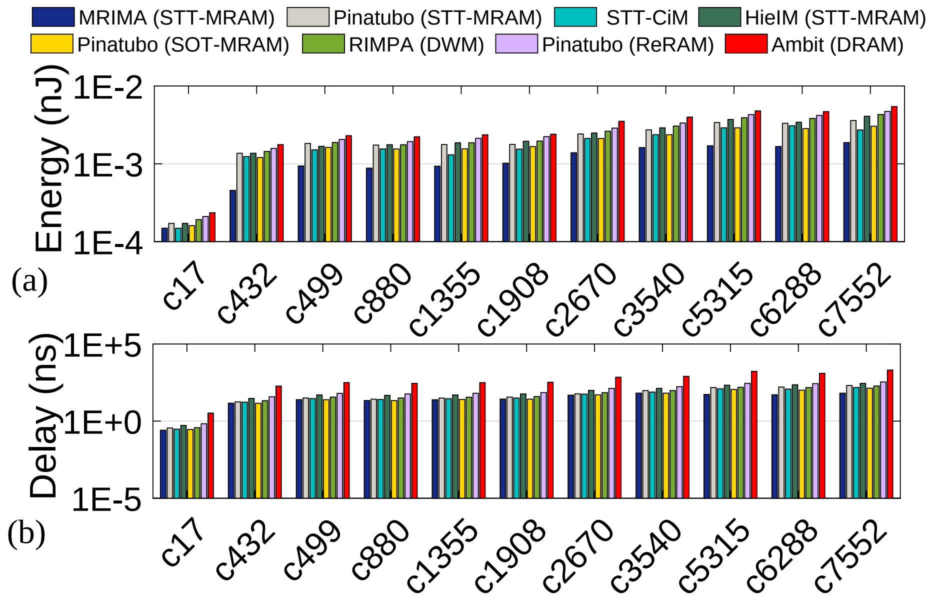


Figure 15: (a) Energy and (b) Delay of ISCAS85 benchmarks (Y-axis: Log scale).

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 than the rest.

2.3.3.2 Design II: AlignS based on SOT-MRAM

In AlignS [2], as depicted in Fig. 16, to realize in-memory XNOR2 logic between every two bits stored in the identical column, a capacitive voltage divider is presented after reconfigurable SA's OR2 and NAND2 outputs (Fig. 16b Ⓐ) driving a CMOS inverter (with low-Vth PMOS and high-Vth NMOS) to realize a NAND2 function between outputs, thereby enabling a multi-kilobyte-wide bit-wise XNOR2 of two rows

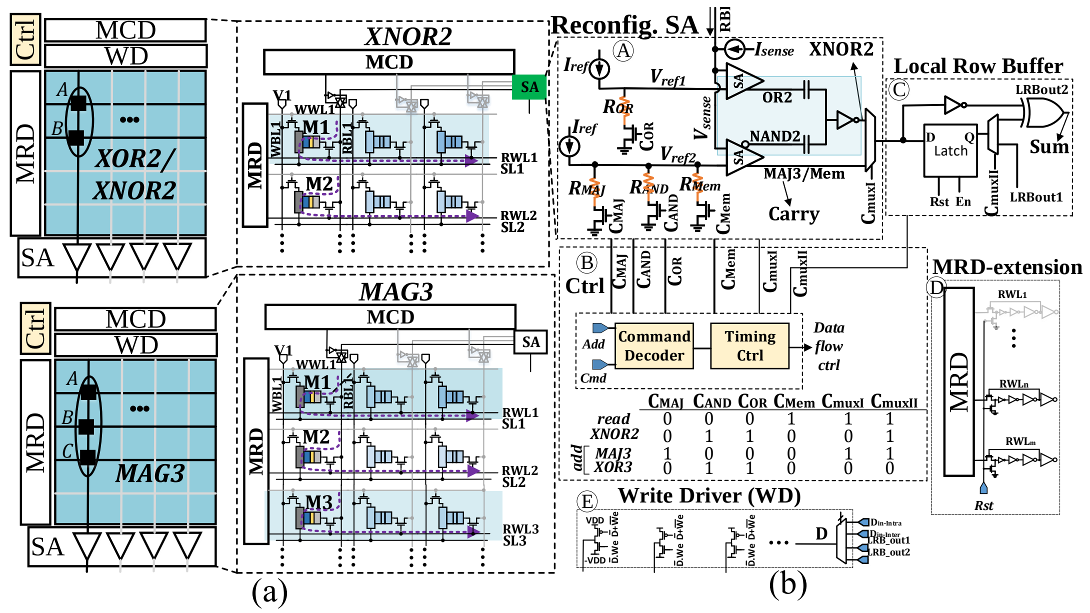


Figure 16: (a) Block level scheme of computational sub-array and SOT-MRAM realization of 2-input and 3-input in-memory logic methods in AlignS [2], (b) AlignS's reconfigurable SA and peripheral circuitry.

in AlignS's sub-arrays. Note that, the dual-threshold technique can eliminate the leakage current through a transistor, thereby decreasing leakage power consumption while maintaining performance [95], [96].

AlignS's sub-array can also perform add/sub operation quite efficiently. The carry-out of the full-adder can be directly produced by MAJ function (Carry in Fig. 16b Ⓐ) just by setting C_{MAJ} to '1' in a single memory cycle. Meanwhile, the existing latch in LRB (Fig. 16b Ⓒ) is equipped with additional NOT and XOR2 gates to first store intermediate carry outputs and then perform the summation of next bits using two XOR2 gates (implementing XOR3). Now, assume A, B, and C operands (Fig. 16a), the 3- and 2-input in-memory logic schemes can generate Carry(/Borrow) and Sum (/Difference), respectively, in two consecutive cycles. The Ctrl's configuration for such add operation is shown in Fig. 16b Ⓑ.

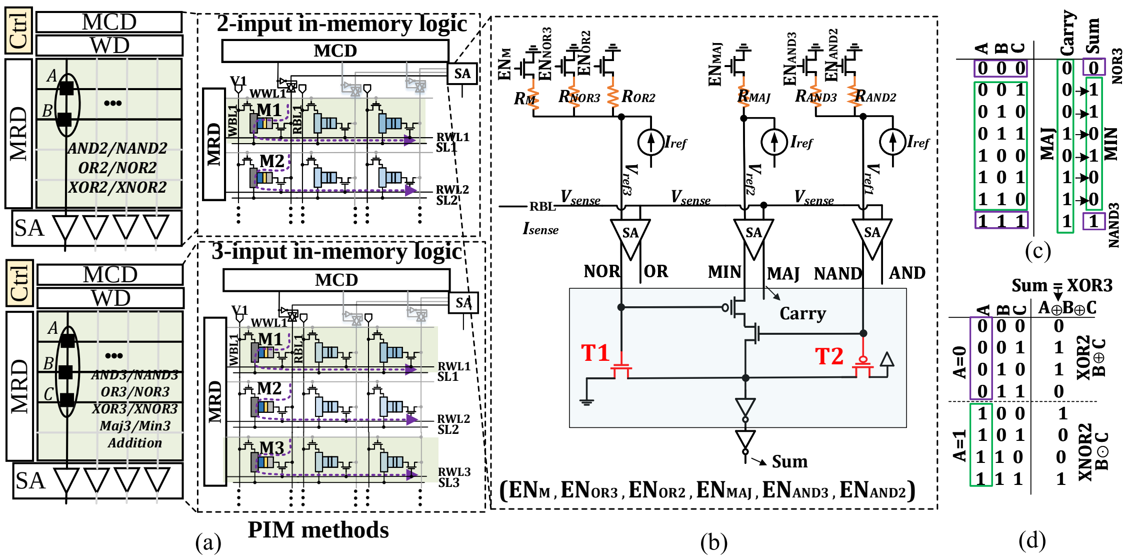


Figure 17: (a) Block level scheme of computational sub-array and SOT-MRAM realization of 2-input and 3-input in-memory logic methods in GraphS [4], (b) Reconfigurable SA, (c) Truth table of addition operation implementation, (d) Truth table for realizing X(N)OR2.

2.3.4 Reconfigurable PIM Supporting One-Cycle In-Memory Addition

The GraphS's reconfigurable SA[2] [4], as depicted in Fig. 17b, consists of three 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 6. Such reconfigurable SA could again implement memory read and one-threshold-based logic functions only by activating one enable at a time. Meanwhile, by activating two or three enables at a time, two or three logic functions can be simultaneously implemented and further used to generate complex logic functions like X(N)OR3, as explained accordingly.

[2]A variation of this design is named PIM-Aligner [10].

GraphS supports both IML2x and IML3x operations. In IML3x, every three cells located in an identical column can be selected by MRD and sensed simultaneously to realize 3-input majority/minority functions (MAJ/MIN) in a single sensing cycle. Consider the data organization shown in Fig. 17a where A, B and C operands correspond to M1, M2, and M3 memory cells, respectively, the computational sub-array can perform $AB + AC + BC$ Boolean function by setting EN_{MAJ} to ‘1’.

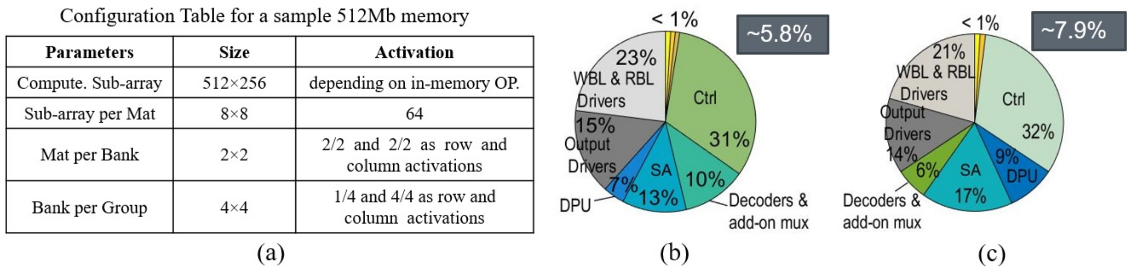
Table 6: Configuration of enable bits for different functions.

Ops.	read	(N)OR2	(N)AND2	MAJ/MIN	(N)OR3	(N)AND3	Add/XNOR3/X(N)OR2
EN_M	1	0	0	0	0	0	0
EN_{OR2}	0	1	0	0	0	0	0
EN_{AND2}	0	0	1	0	0	0	0
EN_{OR3}	0	0	0	0	1	0	1
EN_{AND3}	0	0	0	0	0	1	1
EN_{MAJ}	0	0	0	1	0	0	1

Besides, with careful observation on the Full-Adder (FA) truth table, we realized that in six out of eight possible input combinations, Sum output can be directly obtained by inverted Carry signal as shown in Fig. 17c. Keep this fact in mind that FA’s Carry can be resulted from MAJ function; the proposed reconfigurable SA can implement such Sum output readily by MIN (majority-not) function inspired by [97]. As shown in Fig. 17b-c, the Sum signal is directly connected to the MIN output. However, for two extreme cases, i.e., (0,0,0) and (1,1,1), the MIN signal is disconnected and Sum can be respectively implemented by NOR3 (T1:ON, T2:OFF $\rightarrow$ Sum=‘0’) and NAND3 functions (T1:OFF, T2:ON $\rightarrow$ Sum=‘1’). This is realized by adding two pass transistors in the MIN function path. Note that, considering the fact that Sum output is the XOR3 function, the proposed reconfigurable SA can also implement 2-input and 3-input XOR functions, without imposing additional XOR gates like previous works [1], [2], [19], [30] as shown in Fig. 17d. Now, assume A, B and C as input operands (in Fig. 17a), IML3x can generate Sum(/Difference) and Carry(/Borrow) bits in a single cycle. To the best of our knowledge, the design proposed here is the first PIM that can directly implement in-memory addition in a single memory cycle.

2.3.5 Area Overhead Analysis

Here, two PIM designs, i.e., MRIMA [1], and GraphS [4] as the two most-enhanced platforms that support a wide range of single- and two-cycle logic operations are selected for the area-overhead analysis. Fig. 18b-c shows the breakdown of the area overhead for MRIMA supporting two-cycle addition scheme and GraphS supporting one-cycle addition scheme with the same configuration (Fig. 18a) for a sample 512Mb memory. MRIMA and GraphS impose $\sim$5.8% and $\sim$7.9% area overhead[3] to the memory, respectively, to realize such in-memory computation support. Therefore, the proposed PIM platforms respect the memory's cost-sensitivity while offering fast in-memory logic operations.


Configuration Table for a sample 512Mb memory

Parameters	Size	Activation
Compute. Sub-array	512×256	depending on in-memory OP.
Sub-array per Mat	8×8	64
Mat per Bank	2×2	2/2 and 2/2 as row and column activations
Bank per Group	4×4	1/4 and 4/4 as row and column activations

Figure 18: (a) Configuration table for a sample 512Mb memory, (b) MRIMA area overhead, (c) GraphS area overhead.

[3]The full bottom-up evaluation framework is explained in Chapter 4.

2.4 Highly Flexible and Energy-Efficient In-Memory Logic Computation

2.4.1 PIMA-Logic

To efficiently address the operand locality challenge in NVMs as the third challenge discussed in Section 2.1.4, a new column-wise near-memory majority operation using a Resistive Unit (RUnit) is proposed in [5]. The RUnit is a low-overhead and highly-efficient solution to process operands located in one memory row either in sub-array or inter-component level. In the sub-array level as shown in Fig. 19 L.H.S., a Mode demultiplexer (MDMUX) is devised to switch between basic PIM mode discussed in Section 2.3.1 and the proposed enhanced one (PIM+RUnit). As it can be seen in the block-level sub-array architecture, each SA's output is routed to MDMUX. According to the mode selector, output data can be routed to either GMUX[4] or RUnit.

The key idea behind RUnit is to realize a majority logic after SAs to further process the data avoiding unnecessary write-back and accelerating in-memory processing. As shown in Fig. 19 Ⓕ, the in-block circuit design of RUnit consists of n resistors (n=# of SAs) that can contribute in parallel to design a voltage divider driving a static CMOS inverter. To do the computation, MCD is modified (similar to that of MRD) such that it can activate more than one RBL at the same time. As a result, more than one column can be sensed and routed from SAs to RUnit. Considering a similar resistance (R), the input voltage of the inverter (V_i) can be simply derived,

$$V_i = \frac{k.V_{DD}}{w} \tag{2.7}$$

[4]Glossary: CMUX: Control Mux located at RUnit, Din-Intra: Data input to sub-array, Din-Inter: Data input coming from other sub-arrays, GMUX: Global Mux for interfacing with intra- and inter-subarrays, MDMUX: Mode D-Mux specially designed to select Runit for more complex computations.

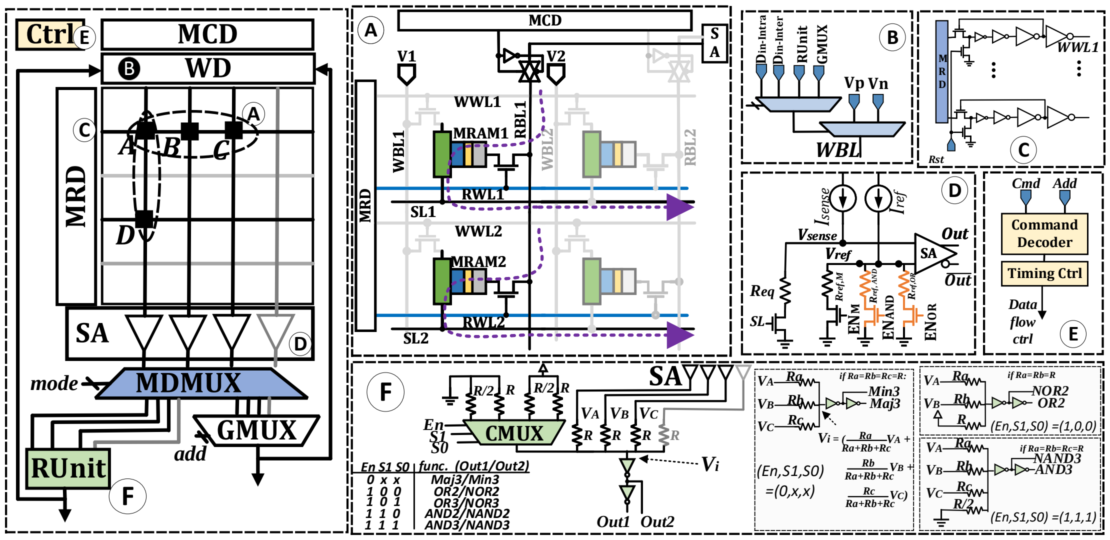


Figure 19: The PIMA-Logic sub-array architecture [5]. Left: block level sub-array architecture, Middle: SOT-MRAM realization, and Right: functional blocks used in sub-array.

Where k denotes the number of SA outputs carrying V_{DD} and w represents the total number of unit resistors (R) connected to the inverter. Thus, the first inverter acts as a threshold detector by amplifying deviation from $\frac{V_{DD}}{2}$ and realizes a minority function. Then, the second inverter yields the majority function output. In addition to majority/minority function-based computing, RUnit is equipped with CMUX to assign different weighted inputs to V_i. This could be used to directly implement column-wise multi-input AND/OR functions. For instance, as shown in Fig. 19 Ⓕ, 3-input NAND function can be efficiently designed by setting $(En, S1, S2)$=(1,1,1). To avoid logic failure due to a large number of inputs, causing the V_i to be close to $\frac{V_{DD}}{2}$, the number of activated columns are limited to three. However, when CMUX is deactivated ($(En, S1, S2)$=(0,x,x)), the simulations showed that up to five columns can be reliably sensed and computed. To better explain the proposed circuit, let's assume A, B, and C operands are located in the way shown in L.H.S. of Fig. 19. For calculating the minority and majority functions in a single cycle, 3 RBLs are activated simultaneously and sensed. CMUX is set by Ctrl to $(En, S1, S2)$=(0,x,x). It is expected that the result of sensed RBLs to be zero after the second inverter if at least two of the three SA outputs are '0' ($k = 0, 1$), and the result to be V_{DD}, if at least two of three SA outputs are carrying '1'($k = 2, 3$), in this way:

$$\begin{cases} V_i < \frac{V_{DD}}{2} \Rightarrow V_{Out1} = 0, & k = 0, 1 \\ V_i > \frac{V_{DD}}{2} \Rightarrow V_{Out1} = V_{DD}, & k = 2, 3 \end{cases} \tag{2.8}$$

It is noteworthy that considering non-aligned data either in the same row or column, the PIMA-Logic's column-wise and row-wise operations need more than one cycle to line data in either the same row or column to perform the computation.

Fig. 20 intuitively depicts performing some simple Boolean functions within PIMA-Logic compared to Pinatubo [23]. As it can be seen, A and B operands can be processed (AB) efficiently in one single cycle regardless of their physical address using conventional row-wise PIM operation (Conv. PIM) and column-wise operation using RUnit of PIMA-Logic. However, a similar function is implemented in 3 cycles using Pinatubo when operands are not aligned in one column. This can be further explored while computing more complex logic functions. As shown, the majority function ($AB+AC+BC$) can be computed in one single cycle using PIMA-Logic; however, Pinatubo needs more than ten cycles to perform such function.

In the inter-component level, we consider two RUnit per component (Sub-array, MAT, Bank). If the operands are in different subarrays (/MATs/Banks) within one MAT (/Bank/memory chip), PIMA-Logic performs inter-component operations employing RUnit added on the row buffer. The first operand row is read into a devised buffer, and accordingly second operand is read via GBL. After computation, the final result is latched in the row buffer.

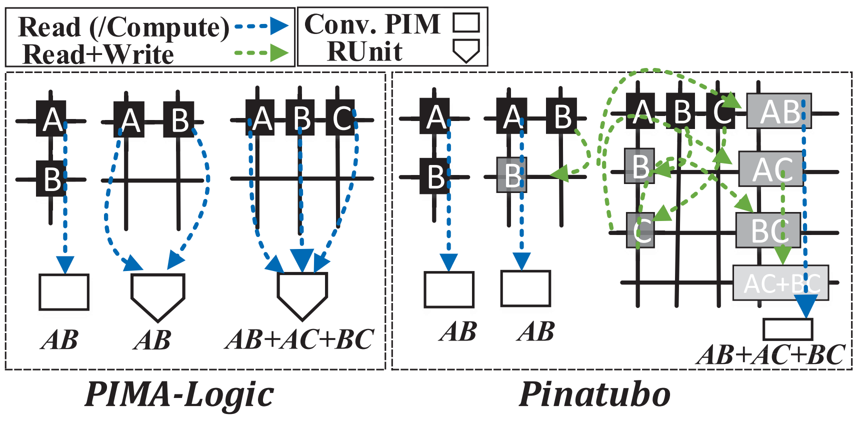


Figure 20: Performing Boolean functions using PIMA-Logic and Pinatubo [23].

For further exploration, two experiments are conducted to thrive the superiority of PIMA-Logic compared to two recent PIM architectures (i.e., RIMPA [49], and Pinatubo [23]). Table 7 tabulates the synthesis of 13 standard functions [98], to represent all 256 possible 3-variable Boolean functions, utilizing different platforms. To perform an impartial comparison, it is assumed that initial physical addresses for all operands are either in the Same Column (SC) or the Same Row (SR). Based on Table 7, PIMA-Logic can show up to 36.5% and 43.9% improvement in terms of the average number of cycles compared with RIMPA and Pinatubo, respectively, for processing 13 functions with SR condition. In the second experiment, data is Randomly-Distributed (RD) in the memory prior to the computation. In this case, PIMA-Logic can show up to 43.1% and 50.8% improvements compared to RIMPA and Pinatubo, respectively.

As an instance of combinational logic circuits, we show the realization of a full-adder within PIMA-Logic in Fig. 21. Assuming A, B and C are initially located in a memory row, Carry output (C_{out}) is generated in a single cycle (see function 9 in Table 7), accordingly Sum can be generated as $Sum = M5(A, B, C, \overline{C_{out}}, \overline{C_{out}})$ after three cycles. The idea can be generalized by implementing an efficient in-memory 4-bit

Table 7: Synthesis comparison of the 13 standard functions.

No.	Standard Function	RIMPA[49]			Pinatubo[23]			PIMA-Logic		
		SC	SR	RD	SC	SR	RD	SC	SR	RD
1	F=AB'C	7	5	5	5	7	9	5	3	3
2	F=AB	3	1	3	1	3	5	1	1	1
3	F=A'BC+A'B'C'	19	15	17	15	19	17	9	11	9
4	F=A'BC+AB'C'	13	17	13	13	17	17	11	11	9
5	F=A'B+BC'	13	9	11	9	13	11	9	7	7
6	F=AB'+A'BC	11	11	11	11	11	15	9	7	7
7	F=A'BC+ABC'+A'B'C'	21	25	21	21	25	25	13	13	11
8	F=A	1	1	1	1	1	1	1	1	1
9	F=AB+BC+CA	5	1	3	7	13	11	5	1	1
10	F=A'B+B'C	9	9	9	9	9	11	9	5	5
11	F=A'B+BC+AB'C'	17	21	19	17	21	21	17	13	11
12	F=AB+A'B'	11	9	9	1	3	3	5	5	5
13	F=ABC'+A'B'C'+AB'C+A'BC	33	29	31	29	31	31	25	19	17
	Total Number of Cycles	162	153	153	139	173	177	119	97	87
	Average Number of Cycles	12.46	11.76	11.76	10.69	13.3	13.6	**9.15**	**7.46**	**6.69**
	Improvement Percentage	26.5%	36.5%	43.1%	14.4%	43.9%	50.8%	-	-	-

ripple Carry Adder (RCA) in Fig. 21. As shown, column-wise computation employing RUnit could be adopted to realize such complex circuits very efficiently.

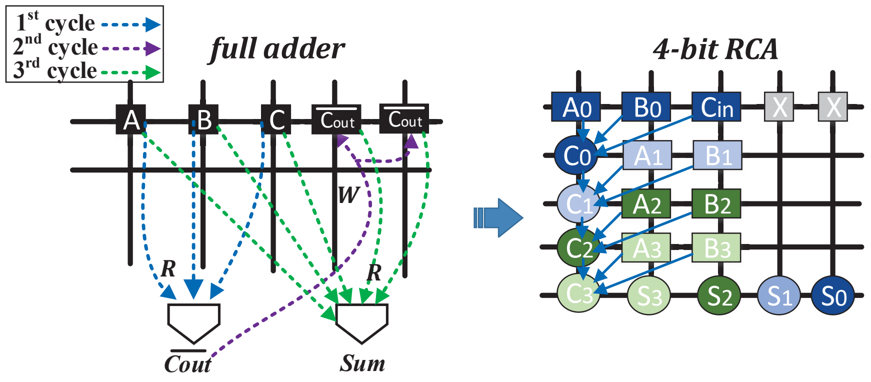


Figure 21: Realization of in-memory full adder and 4-bit RCA in PIMA-Logic.

2.4.1.1 Logic Performance

To evaluate the logic performance of PIMA-Logic, the device-to-circuit level data[5] is extracted. The simulation is initially carried out in Cadence Spectre with NCSU 45nm CMOS PDK [99]. SOT-MRAM device model of Fig. 5b is used in the circuit simulation. MTJ resistance (R_{MTJ}) is obtained from the NEGF approach [77], while the heavy metal resistance (R_{HM}) is calculated based on the resistivity and device dimension. Accordingly, a logic netlist in Berkeley Logic Interchange Format (.blif) is fed into ThrEshold Logic Synthesizer (TELS) [94] to obtain synthesized logic networks. Meanwhile, parameters such as fan-in restriction are set up during the synthesis. The synthesized networks are then mapped to PIMA-Logic using an in-house developed Matlab code to assess the performance. Fig. 22 gives ISCAS85 combinational circuit benchmarks implemented using PIMA-Logic, RIMPA [49],and Pinatubo [23]. To have an impartial comparison, Pinatubo, a general system architecture for NVMs, is implemented with standard STT-MRAM and identical SOT-MRAM cells.

As shown, PIMA-Logic exhibits the lowest energy and delay compared to the counterparts in different benchmarks. (1) PIMA-Logic reduces the energy consumption by ~56%, 67%, and 74.4% compared to Pinatubo-SOT, Pinatubo-STT, and RIMPA, respectively. This considerable improvement mainly comes from proposed logic efficiency and reduced-cycle operations. (2) PIMA-Logic outperforms mentioned PIM architectures with 31.6%, 40%, and 52% reduction in delay on different benchmarks. It is worth pointing out that for five more complex benchmarks (i.e., c2670, c3540, c5315, c6288, and c7552), as logic complexity increases, PIMA-Logic can show much better performance compared to the rest.

[5]The full bottom-up evaluation framework is explained in Chapter 4.

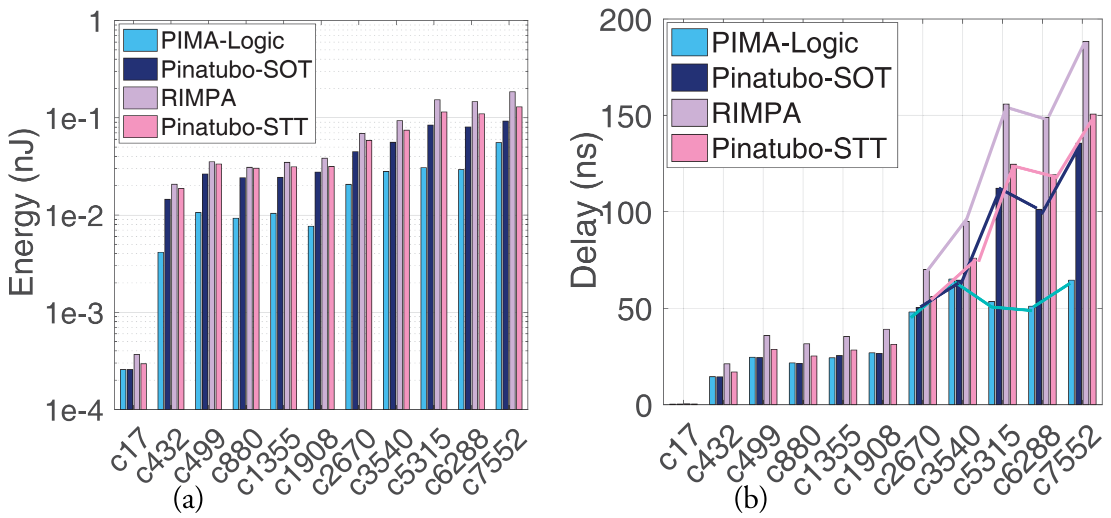


Figure 22: (a) Energy consumption and (b) Delay of ISCAS85 benchmarks mapped to three different PIM architectures (Y-axis in energy plot: Log scale).

2.5 Summary

Chapter 2 focuses on designing customized SA designs for NVMs to provide high-reconfigurability on the memory side with low area-overhead. In this way, the evolution of proposed SA designs is shown from the basic (N)AND2/(N)OR2 and X(N)OR2-support all the way double and single-cycle in-memory addition schemes. The proposed PIM platforms in this chapter could be considered as a generic PIM solution for big data computation, overcoming the first and second challenges discussed in Section 2.1.4. For instance, the proposed GraphS [4] provides a full set of 1-/2-/3-input Boolean logic functions (i.e., NOT, AND/NAND, OR/NOR, XOR/XNOR, add/sub) by configuring the enable signals with ∼7.9% area overhead to the memory die. Moreover, to address the operand locality issue as the third challenge, PIMA-Logic [5] is proposed to perform row-wise and column-wise operations. However, our main observation is, it is more suitable for unstructured benchmarks such as ISCAS85, and when it comes to structured bulk benchmarks, PIMA-Logic can sacrifice the PIM parallelism.

Chapter 3

RECONFIGURABLE PIM BASED ON VOLATILE MEMORIES

3.1 Introduction

In this chapter, two reconfigurable DRAM-based PIM accelerators are presented, which transform current DRAM architecture to massively parallel computational units exploiting the high internal bandwidth of modern memory chips. The presented designs leverage the well-known analog operation of DRAM sub-arrays and elevate it to implement a full set of 1- and 2-input bulk bit-wise operations (NOT, (N)AND, (N)OR, and even X(N)OR) between operands stored in the same bit-line, based on new row activation mechanisms with a modest change to peripheral circuits such as sense amplifiers.

3.1.1 Processing-in-DRAM Platforms

At the top architectural level, a DRAM hierarchy includes channels, modules, and ranks. With a typically 64-bits wide data bus, each memory rank consists of multiple memory chips. The memory chips are designed with various configurations and operate simultaneously [19], [100], [101]. The memory chip is spilt into several memory banks. Each bank is composed of 2D sub-arrays of memory bit-cells that are virtually-ordered in memory matrices (mats). Banks located in the same chip typically share buffer, and I/O and banks located in different chips work in a lock-step manner. As depicted in Fig. 23a, the memory sub-array consists of 1) Memory rows (normally 2^9 or 2^{10}) connected

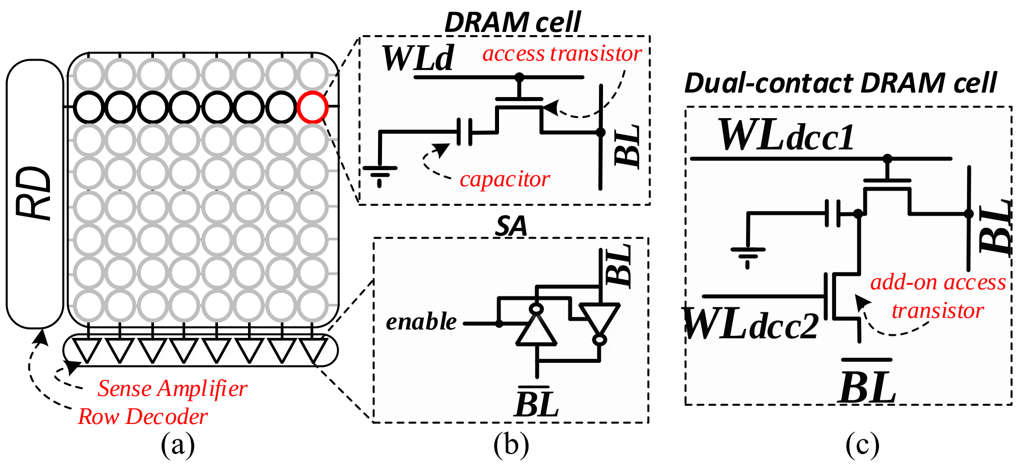


Figure 23: (a) DRAM sub-array organization, (b) DRAM cell and Sense Amplifier, (c) Dual-contact DRAM cell.

to DRAM cells, 2) A Sense Amplifiers' (SA) row, and 3) A memory Row Decoder (RD) connected to the word-lines. Structurally, a DRAM cell is composed of two modules, a storage module (capacitor) and an access module (Access Transistor-AT), as shown in Fig. 23b. The gate and drain of DRAM's AT are connected to the Word Line (WL) and Bit Line (BL), respectively. DRAM cell stores the binary data by the charge of the capacitor. It encodes a fully-charged (V_{dd}) capacitor as logic '1' and no-charge capacitor as logic '0'.

3.1.1.1 Read/Write Operation

For read/write operation, both BL and $\overline{\text{BL}}$ are initially pulled to $\frac{V_{dd}}{2}$. Technically, accessing data from a DRAM's sub-array after the initial state is accomplished with three commands [19], [102] by the memory controller: 1) With the activation command (i.e., ACTIVATE), a target row is activated, and stored row data is transferred from the DRAM row cells to the SA row. Fig. 23b depicts the connection between a cell and the

SA via a BL. The selected cell typically shares its charge value ($0/V_{dd}$) with the BL, which slightly changes the initial BL's voltage ($\frac{V_{dd}}{2} \pm \delta$). Then, the memory controller activates the $enable$ signal that makes the SA amplify the δ towards the original value of the data through voltage amplification leveraging the switching threshold of SA's inverter [102]. 2) By a WRITE/READ command, the data can be then moved to/from SA from/to DRAM bus. It is noteworthy that several WRITE/READ commands can be issued to one row. 3) With a PRECHARGE command, both BL and $\overline{\text{BL}}$ precharge again to the initial state and get ready for the next access cycle.

3.1.1.2 Initialization and Copy Operation

For a very fast in-memory copy operation (<100ns) within DRAM sub-arrays, instead of $\sim 1\mu s$ copy operation in von-Neumann computing architecture, *RowClone*-Fast Parallel Mode (FPM) [103] offers a new method that does not require sending the data to the processing units. In this method, issuing two back-to-back ACTIVATE commands (without PRECHARGE command in between) to the source and destination rows can realize a 90ns and multi-kilo byte in-memory copy operation. This technique has been further exploited for row initialization to effectively copy a preset DRAM row ('1' or '0') to a single or multiple destination row(s), incurring a 0.01% overhead to memory chip area [103].

3.1.1.3 Not Operation

The Dual-Contact Cell (DCC) has been used so far to realize an in-memory NOT operation [19], [104], [105], as depicted Fig. 23c. DCC is developed on top of the typical DRAM cell but has one more AT linked to $\overline{\mathrm{BL}}$. It works by issuing two back-to-back ACTIVATE commands [19]. The controller activates the WL_{dcc1} of the DRAM cell (Fig. 23c), and reads out the data, and sends it to the SA via BL. Accordingly, the controller activates WL_{dcc2} to connect $\overline{\mathrm{BL}}$ (inverted data) to the same capacitor writing the negated result back to the DCC.

3.1.1.4 Other Logic Functions

To implement the logic operation in processing-in-DRAM architecture, the RowClone idea was extended in the Ambit [19] to realize three-input majority-based operations (Maj3) in memory through simultaneously issuing the ACTIVATE command to three rows with a PRECHARGE command afterward, named *Triple Row Activation (TRA)* mechanism. Ambit incurs just 1% area overhead to DRAM chip [19]. Having one row as the control (D_k), as illustrated in Fig. 24a, initialized by '0'/'1', TRA implements in-memory AND2/ OR2 based on Maj3 function via charge sharing among connected DRAM cells (D_k, D_i and D_j) and writes the result back on D_r cell. In addition, Ambit employs the TRA method along with DCCs to implement complementary operations. Nevertheless, Ambit deals with multi-cycle PIM operations to realize other logic functions like X(N)OR2. The DRISA-3T1C mechanism [15] alternatively leverages the 3-transistor 1-capacitor DRAM design [106]. As shown in Fig. 24b, such cell design is composed of two separated write/read ATs and one additional transistor for decoupling the capacitor from the read BL (rBL). This additional transistor links the two input DRAM cells in a NOR style on the rBL to perform the Boolean-complete NOR2 function. DRISA-3T1C incurs a large area overhead and needs multi-cycle operations to realize different logic functions.

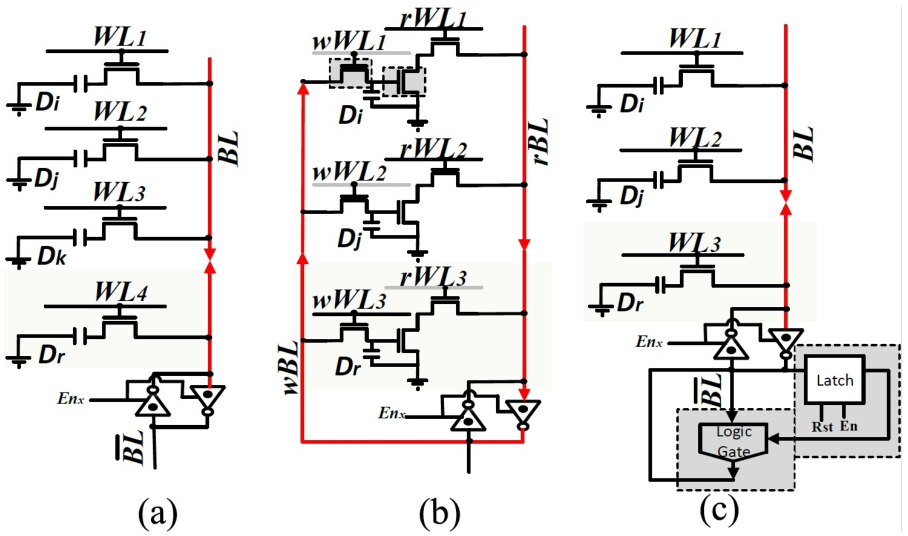


Figure 24: (a) TRA mechanism in Ambit [19], (b) 3T1C mechanism in DRISA [15], (c) 1T1C-logic mechanism in DRISA [15]. Glossary- D_i/D_j: input rows data, D_k: initialized row data, D_r result row data.

As depicted in Fig. 24c, DRISA-1T1C mechanism [15] performs in-memory operations via an upgraded SA consisting of a CMOS logic gate and a latch. This mechanism performs in-memory operations in two consecutive cycles: 1) reading out D_i and storing in the latch as the first input of CMOS logic, and 2) reading out D_j as the second input to perform the computation. This method requires excessive cycles to realize other logic functions and imposes 12 transistors to each SA. The Dracc [107] recently designs a carry look-ahead adder by improving Ambit [19] to accelerate convolutional neural networks.

3.1.2 Challenges

There are three main challenges in the existing processing-in-DRAM platforms that make them inefficient acceleration solutions for various big data applications, and this chapter aims to resolve them:

- Limited throughput (Challenge-1): Due to the intrinsic complexity of X(N)OR-based logic implementations, current PIM designs (such as Ambit [19], DRISA [15], and Dracc [107]) are not able to offer a high-throughput and area-efficient X(N)OR or addition in-memory operation despite utilizing maximum internal DRAM bandwidth and memory-level parallelism for NOT, (N)AND, (N)OR, and MAJ/MIN logic functions. Moreover, while the DRISA-1T1C method could implement either XNOR or XOR functions as the add-on logic gate, it requires at least two consecutive cycles to perform the computation, limiting other logic implementation. We address this challenge by proposing the DRA mechanism in Sections 3.2 and 3.3.
- Row initialization (Challenge-2): Given R=A*op*B function ($op \in$ AND2/OR2), TRA-based method [19], [102] takes 4 consecutive steps to calculate one result as it relies on row initialization: 1-RowClone data of row A to row D_i (Copying first operand to a computation row to avoid data-overwritten), 2-RowClone of row B to D_j, 3-RowClone of ctrl row to D_k (Copying initialized control row to a computation row), 4-TRA and RowClone data of row D_i to R row (Computation and Writing-back the result). Therefore, the TRA method needs an averagely 360ns to perform such in-memory operations. When it comes to X(N)OR2 operation, Ambit requires at least three row-initialization steps to process two input rows. Obviously, this row-initialization load could adversely impact the PIM's

energy-efficiency especially dealing with such big data problems. This challenge is addressed in Section 3.3 through the proposed sense amplifier, which totally eliminates the need for initialization in performing X(N)OR-based logics.

- Reliability concerns (Challenge-3): By simultaneously activating three cells in TRA method, the deviation on the BL might be smaller than typical one-cell read operation in DRAM. This can elongate the sense amplification state or even adversely affect the reliability of the result [19], [102]. The problem can even be intensified when multiple TRA are needed to implement X(N)OR-based computations. To explore and address these challenges, we perform an extensive Monte-Carlo simulation on the proposed designs in Sections 3.2 and 3.3.

3.2 Design I: GraphiDe

The GraphiDe is developed in [7] to be an independent, high-performance, and energy-efficient accelerator based on main memory architecture. Each GraphiDe's bank consists of multiple memory matrices (mats). The general mat organization of GraphiDe is shown in Fig. 25a. Each mat consists of multiple computational memory sub-arrays connected to a Global Row Decoder (GRD) and a shared Global Row Buffer (GRB). According to the physical address of operands within memory, GraphiDe's Controller (Ctrl) is able to configure the sub-arrays to perform data-parallel intra-sub-array computations. The proposed design is motivated by the Ambit [19] PIM method, which leverages charge sharing among different rows to perform logic operations, but with significant modifications. We divide the GraphiDe's sub-array row space into two distinct regions as depicted in Fig. 25b: 1- Data rows (500 rows out of 512) that include the typical DRAM cells connected to a regular Row Decoder (RD), and 2- Computa-

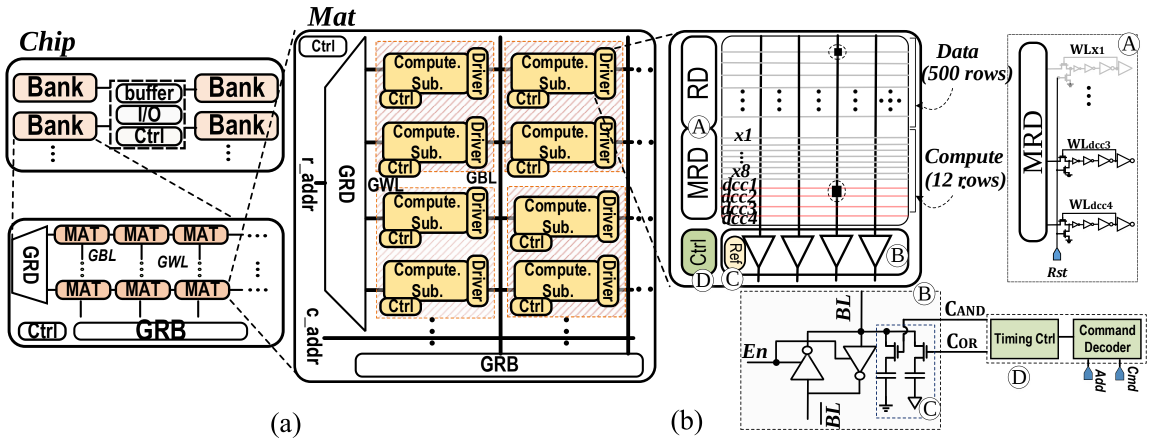


Figure 25: (a) The GraphiDe memory organization, (b) Block level scheme of computational sub-array and peripheral circuitry [7].

tion rows (12), connected to a Modified Row Decoder (MRD) (Fig. 25b Ⓐ), which enables bulk bit-wise in-memory operations between copied operands. Eight computational rows ($x1, ..., x8$) include typical DRAM cells and four rows ($dcc1, ..., dcc4$) are allocated to special DCCs enabling NOT function in every sub-array. In the following, we propose dual-row in-memory AND-OR and addition operations that further enhance Ambit to perform bulk bit-wise operations.

3.2.1 GraphiDe's Dual Row Activation Mechanism

The TRA mechanism imposes an excessive latency and energy to the memory chip, which could be alleviated by rethinking the process. As discussed, given R=A*op*B function (*op* $\in$ AND2/OR2), Ambit takes 4 consecutive steps to calculate the result with RowClone operations. As a matter of fact, every RowClone command imposes ~90ns [103], therefore the TRA method takes an averagely 360ns to perform in-memory operations. Our key idea to perform dual-row bit-line computing in GraphiDe is still based on majority function but by selecting different thresholds (references) when performing the charge sharing between selected memory cell(s). The proposed reconfigurable SA, as depicted in Fig. 25b Ⓑ, consists of a regular SA with two back-to-back inverters connected to two fixed reference-capacitor branches Ⓒ that can be selected by control bits (C_{AND}, C_{OR}) by the sub-array's Ctrl Ⓓ. This design forms a capacitive voltage divider between two selected cells by MRD and the activated reference (connected to either GND or Vdd), driving a CMOS inverter, to implement AND2 or OR2 functions, respectively.

GraphiDe's Dual Row Activation method (DRA) eliminates the need for the third RowClone step in Ambit's AND2/OR2 operations. It saves two initialized memory rows used for controls per sub-array at the cost of adding two low-overhead reference capacitors in the SA unit. Fig. 26 shows the realization of AND2 operation in GraphiDe's sub-array. Consider A and B operands are RowCloned from Data rows to $x1$ and $x2$ rows ① and both BL and $\overline{\text{BL}}$ are precharged to $\frac{V_{dd}}{2}$. The DRA simultaneously activates two WLs, and the corresponding reference (C_{AND}) for charge-sharing ②. During sense amplification ③, with the similar capacitance (C_c) of memory cells and the reference, input voltage of first inverter (V_i) in SA is simply derived as $V_i = \frac{n.V_{dd}}{C}$, where n denotes the number of DRAM cells storing logic '1' and C represents the total number of unit capacitors (C_c) connected to the inverter. Thus, the inverter acts as a threshold detector by amplifying deviation from $\frac{V_{dd}}{2}$ and realizes a NAND2 function on $\overline{\text{BL}}$ and consequently AND2 function (AB) on BL. GraphiDe can perform such DRA-based operations in $\sim$240ns by eliminating the need for the third RowClone step in Ambit's operations. In this work, we use Ambit's TRA method to directly realize in-memory majority function (Maj3) and AND2/OR2 operations are realized through the DRA method.

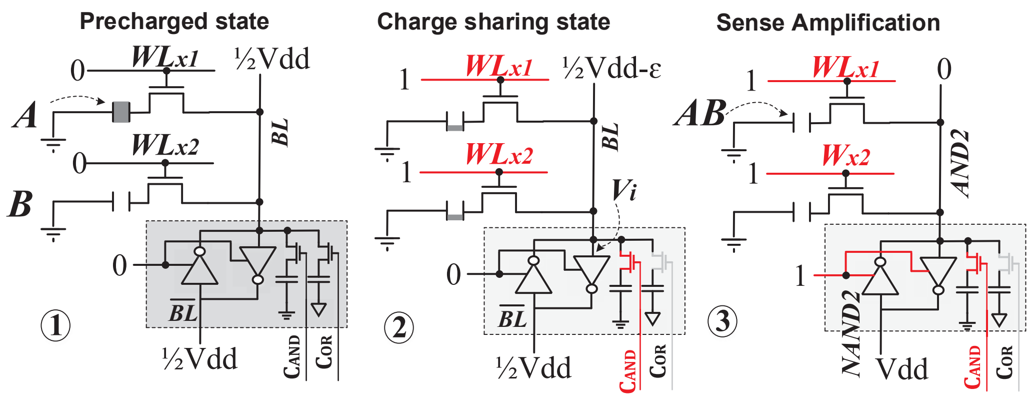


Figure 26: Realization of AND2 function in GraphiDe.

3.2.2 In-memory Adder

GraphiDe proposes a Quintuple Row Activation method (QRA) as an extension for the TRA method, realizing a 5-input (Maj5) operation. In this method, GraphiDe's MRD (Fig. 25b Ⓐ) helps to activate five WLs, simultaneously. During the precharged state as shown in Fig. 27 ①, both BL and $\overline{\text{BL}}$ are connected to $\frac{V_{dd}}{2}$. By activating the five WLs ($WLx1$ to $WLx5$), the memory cells storing input operands start to charge sharing ②. In this case, since three of the five cells are initially in the charged state, charge sharing results in a positive deviation on the BL. Therefore, by activating the Enable (En), such deviation from $\frac{V_{dd}}{2}$ is amplified ③, and the SA drives the BL to Vdd and accordingly, fully charges all the five cells. Based on Maj3 and Maj5 schemes, a new parallel in-DRAM computation and mapping method for addition (add) operation can be presented to accelerate a wide spectrum of big data tasks. Assume D_i, D_j, and D_k as input operands, the carry-out (C_{out}) of the Full-Adder (FA) can be generated through $MAJ3(D_i, D_j, D_k) = D_iD_j + D_iD_k + D_jD_k$ using TRA method. Moreover, the Sum can be readily carried out through $MAJ5(D_i, D_j, D_k, \overline{C_{out}}, \overline{C_{out}})$ with only writing back the $\overline{C_{out}}$ into memory (leveraging two DCC rows) and then applying QRA method.

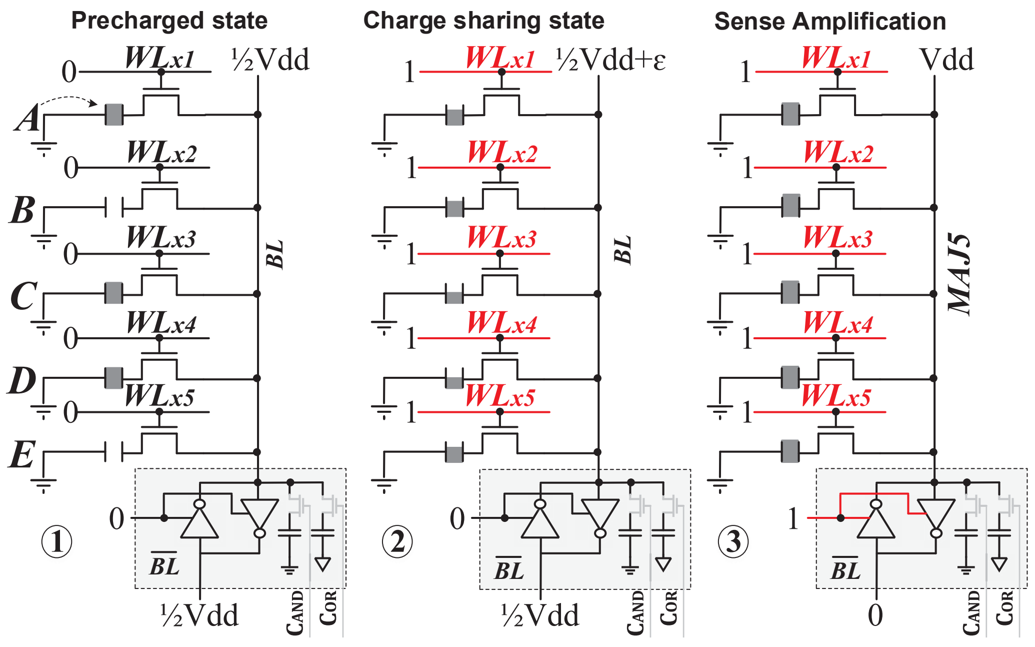


Figure 27: Realization of MAJ5 function in GraphiDe.

3.2.3 ISA Support

While GraphiDe is meant to be an independent high-performance and energy-efficient accelerator, it has to be exposed to programmers and system-level libraries to utilize it. From a programmer perspective, GraphiDe 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 GPUs. Therefore, a virtual machine and ISA for general-purpose parallel thread execution need to be defined similar to PTX [108] for NVIDIA. Accordingly, the programs are translated at install time to the GraphiDe hardware instruction set discussed here to realize the functions tabulated in Table 8. The micro and control transfer instructions are not discussed here.

Table 8: The basic functions supported by GraphiDe.

Function	Operation	Command Sequence	AAP Type
copy	$D_r \leftarrow D_i$	AAP(D_i, D_r)	1
NOT	$D_r \leftarrow \overline{D_i}$	AAP$(D_i, dcc2)$	1
		AAP$(dcc1, D_r)$	1
AND2	$D_r \leftarrow D_i.D_j$	AAP$(D_i, x1)$	1
		AAP$(D_j, x2)$	1
		AAP$(x1, x2, D_r, 0)$	3
OR2	$D_r \leftarrow D_i + D_j$	AAP$(D_i, x1)$	1
		AAP$(D_j, x2)$	1
		AAP$(x1, x2, D_r, 1)$	3
XOR2	$D_r \leftarrow D_i \oplus D_j$	AAP$(D_i, x1, dcc2)$	2
		AAP$(D_j, x2, dcc4)$	2
		AAP$(x1, dcc3, x4, 0)$	3
		AAP$(x2, dcc1, x5, 0)$	3
		AAP$(x4, x5, D_r, 1)$	3
Addition	$Sum \leftarrow D_i \oplus D_j \oplus D_k$	AAP$(D_i, x1)$	1
	$C_{out} \leftarrow MAJ3(D_i, D_j, D_k)$	AAP$(D_j, x2)$	1
		AAP$(D_k, x3)$	1
		AAP$(x1, x2, x3, C_{out})$	4
		AAP$(C_{out}, dcc2, dcc4)$	2
		AAP$(x1, x2, x3, dcc1, dcc2, Sum)$	5

GraphiDe is developed based on ACTIVATE-ACTIVATE-PRECHARGE command referred to as AAP primitives. As thoroughly explained in Ambit [19], most bulk bit-wise operations involve a sequence of AAP commands. There are five types of AAP primitives supported by GraphiDe that only differ from the number of activated source or destination rows, 1- AAP (src, des) that runs the following commands sequence: ACTIVATE source address; ACTIVATE destination address; PRECHARGE. This is manly used for copy and NOT functions as indicated Table 8. 2- AAP (src, des1, des2) that is designed to copy the result of an operation simultaneously to two destination rows. 3- AAP (src1, src2, des, Ctrl) that performs the DRA method by activating two source addresses along with a control input ('0' for C_{AND}/ '1' for C_{OR}) and then writes back the result on the destination address. 4- AAP (src1, src2, src3, des) that performs TRA method by activating three source rows simultaneously and writing back the MAJ3 or MIN3 result on the destination address. 5- AAP (src1, src2,

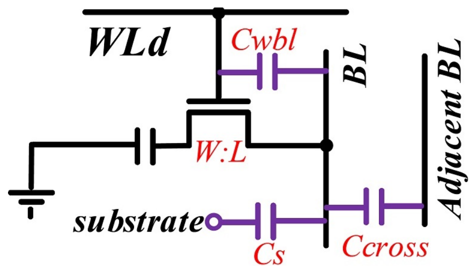


Figure 28: Noise sources in DRAM cell. Glossary: $Cwbl$, Cs, and $Ccross$ are WL-BL, BL-substrate, and BL-BL capacitance, respectively.

src3, src4, src5, des) that performs QRA method on five sources and writes the result back to the destination address.

In order to implement the addition-in-memory, as shown in Table 8, three AAP-type1 commands first copy the three input data rows to computational rows $(x1, x2, x3)$. Then, C_{out} is generated by AAP-type4 and written back to the designated data row. Again, C_{out} row is readout and its inversion is copied to two DCC rows ($dcc2$ and $dcc4$) with AAP-type2. Eventually, AAP-type5 command activates five rows to implement Sum function.

3.2.4 Reliability

An extensive circuit-level simulations following the Ambit's approach [19] is conducted to study the effect of process variation on GraphiDe's DRA and QRA methods considering a worst-case scenario variation in all components (cell/BL/WL capacitance and transistor) as shown in Fig. 28. A Monte-Carlo simulation with 45nm PTM library [109] is conducted (DRAM cell parameters were taken from Rambus [110] model) under 10000 trials. The amount of variation was increased from ±0% to ±20% for each PIM mechanism.

Table 9 shows the percentage of the test error in each variation. It is observed that even considering a significant ±10% [19] variation, the percentage of erroneous DRA or QRA across 10000 trials is just 0.12% and 0.39% which is consistent with what Ambit reports. Therefore, GraphiDe shows acceptable reliability in performing PIM operations. Note that the DRA method is less vulnerable to capacitance variation effects than TRA due to its third fixed-voltage branch. By scaling down the transistor size, the process variation effect is expected to get worse [19], [103]. Since GraphiDe is mainly developed based on existing DRAM structure and operation with slight modifications, different methods currently-used to tackle process variation can also be applied for GraphiDe (e.g., spare rows). Besides, just like Ambit, GraphiDe chips that fail testing due to DRA, TRA, and QRA methods can be considered regular DRAM chips alleviating DRAM yield.

Table 9: Process variation analysis.

Variation	±0%	±5%	±10%	±20%
GraphiDe's DRA	0.00%	0.00%	0.12%	11.43%
GraphiDe's QRA	0.00%	0.08%	0.39%	18.92%

Regarding the error correction, 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 GraphiDe as the data being processed is not visible to the memory controller. Note that this issue is common across all PIM designs. To overcome this issue, GraphiDe can potentially augment each row with additional ECC bits that can be calculated and verified at the memory module level or bank level.

3.2.5 Virtual Memory

GraphiDe has its own ISA with operations that can potentially use virtual addresses. To use virtual addresses, GraphiDe'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 GraphiDe and giving GraphiDe'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 GraphiDe operates within a memory module. Furthermore, page table coherence issues can arise. The other way to implement translation capabilities for GraphiDe is through memory controller pre-processing of instructions being written to GraphiDe instruction registers. For instance, if the programmer writes instruction AAP $add0, add1$, then the memory controller intercepts the virtual addresses and translates them into physical addresses. Note that most systems have near memory controller translation capabilities, mainly to manage IOMMU and DMA accesses from I/O devices. One issue that can arise is that some operations are appropriate only if the resulting physical addresses are within a specific plane, e.g., within the same bank. Accordingly, the compiler and the OS should work together to ensure that the operands of commands will result in physical addresses suitable to the operation type. To avoid the complexity of virtual memory when using GraphiDe, system architects can designate a continuous physical range that GraphiDe and the user/application can use physical addresses for operands. Directly operating on physical addresses can limit multi-tasking on GraphiDe.

3.2.6 Area Overhead

GraphiDe is developed on top of Ambit [19] (with the area overhead of <1%). We have modified the ctrl, MRD, and SAs by adding two reference branches per column. Such enhanced SAs and peripheral circuitry in GraphiDe's sub-array occupy less than 15% of area. Therefore, the overall area overhead of GraphiDe is $\sim$1.3% over the commodity DRAM.

3.3 Design II: ReDRAM

The ReDRAM is proposed in [6] as a generic and independent accelerator based on main memory architecture to support and accelerate a wide variety of big data applications. The main memory organization of ReDRAM is similar to Fig. 25a based on typical DRAM hierarchy. However, we divide the ReDRAM's sub-array row space into two different-sized regions as depicted in Fig. 29a: 1- Data rows (1016 rows out of 1024) connected to an RD, and 2- Computation rows (8-labeled by $x1, ..., x8$), connected to an MRD, which enables dual row activation required for bulk bit-wise in-memory operations between operands. ReDRAM's computational sub-array is developed to perform a full set of bit-wise operations based on the proposed DRA mechanism leveraging charge-sharing among different rows, as discussed below.

3.3.1 ReDRAM's Dual Row Activation Mechanism

The main idea is to perform in-memory logic operations through a DRA mechanism to address all three challenges discussed in Section 3.1.2. To achieve this goal, a

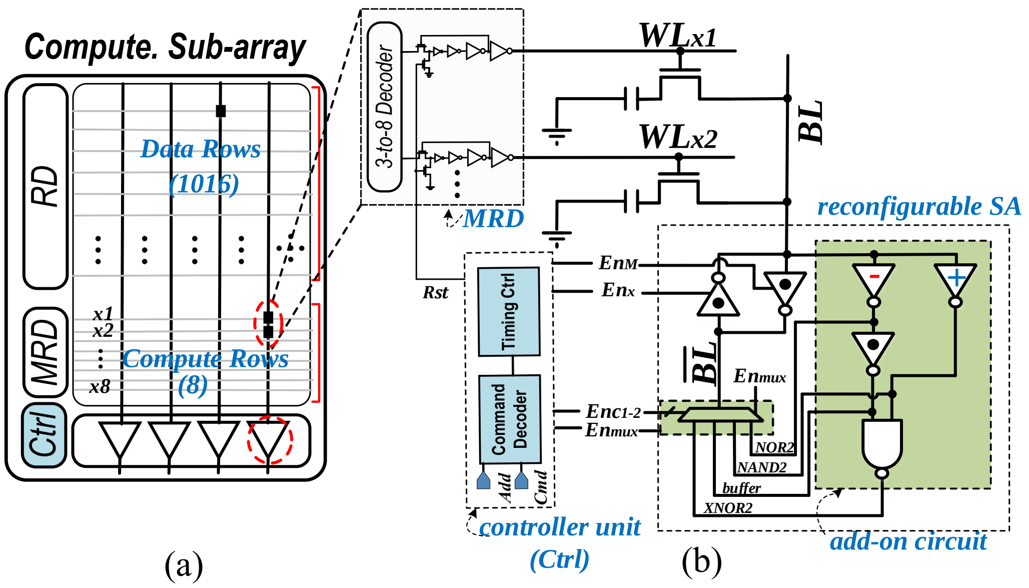
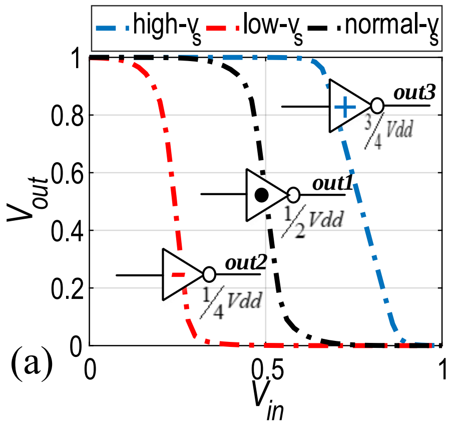


Figure 29: (a) Block level scheme of ReDRAM's computational sub-array, (b) Computational Rows and reconfigurable SA [6].

new reconfigurable SA is developed, as shown in Fig. 29b, on top of existing DRAM circuitry. It consists of a regular DRAM SA equipped with add-on circuits including three inverters, one NAND gate, and one MUX, controlled with five enable signals ($En_M, En_x, En_{mux}, En_{C1}, En_{C2}$). This design leverages the basic charge-sharing feature of the DRAM cell and elevates it to implement NOT/AND2/OR2/XOR2 logic between two selected rows through static capacitive functions in a single cycle. To implement capacitor-based logic, two different skewed inverters with shifted Voltage Transfer Characteristic (VTC) are used as shown in Fig. 30a. In this way, a NAND/NOR logic can be readily carried out based on high switching voltage (V_s)/low-V_s inverters with standard high-V_{th}/low-V_{th} NMOS and low-V_{th}/high-V_{th} PMOS transistors. It is worth mentioning that utilizing low/high-threshold voltage transistors, and normal-threshold transistors have been accomplished in the low-power application, and many circuits have enjoyed this technique in low-power design [97], [111].

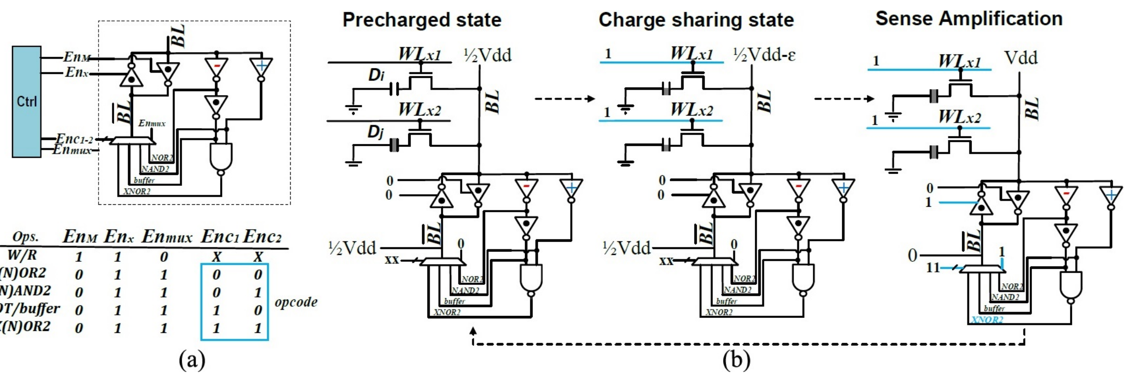
(a) 

(b)

Di	Dj	● out1	- out2	+ out3
0	0	1	1	1
0	1	½ ---→	0	1
1	0	½ ---→	0	1
1	1	0	0	0
			NOR2	*NAND2*

Figure 30: (a) VTC and (b) Truth table of the SA's inverters to realize capacitive NAND2-NOR2 functions.

Fig. 31 gives the detailed control signals of ReDRAM's sub-array to implement different memory and in-memory logic functions (here, e.g., X(N)OR2). ReDRAM's ctrl activates En_M and En_x control-bits simultaneously (when MUX is deactivated-En_{mux}=0) to perform typical memory write/read operation. It is worth noting that in such memory operations, MUX's output voltage is high-z, and $\overline{\text{BL}}$ voltage is solely determined in sense amplification state through two normal-V_s back-to-back inverters, just like normal DRAM's SA mechanism. Therefore, ReDRAM can perform the bulk copy operation based on RowClone mechanism, as discussed earlier. Now, consider D_i and D_j operands (in Fig. 31b) are RowCloned from data rows to $x1$ and $x2$ computational rows and both BL and $\overline{\text{BL}}$ are precharged to $\frac{V_{dd}}{2}$ (Precharged State). ReDRAM's Ctrl first activates two WLs in computational row space (here, $x1$ and $x2$) through the modified decoder for charge-sharing when all the other enable signals are deactivated. During sense amplification state, by setting the proper enable set ($En_M, En_x, En_{mux}, En_{C1}$, En_{C2}), tabulated in Fig. 31a (01111 for X(N)OR2), the input voltage of both low- and high-V_s inverters in the reconfigurable SA can be simply derived as $V_i = \frac{n.V_{dd}}{C}$, where n is the number of DRAM cells storing logic '1' and



Ops.	En_M	En_x	En_{mux}	Enc_1	Enc_2
W/R	1	1	0	X	X
(N)OR2	0	1	1	0	0
(N)AND2	0	1	1	0	1
NOT/buffer	0	1	1	1	0
X(N)OR2	0	1	1	1	1

Figure 31: (a) ReDRAM's control signals and activations in the sense amplification state, (b) Dual Row Activation mechanism. Here, X(N)OR2 is implemented by setting enable set ($En_M, En_x, En_{mux}, En_{C1}, En_{C2}$) to 01111.

C represents the total number of unit capacitors connected to the inverters (i.e., 2 in DRA mechanism). Now, the low-V_s inverter acts as a threshold detector by amplifying deviation from $\frac{1}{4}V_{dd}$ and realizes a NOR2 function, as tabulated in the truth table in Fig. 30b. At the same time, the high-V_s inverter amplifies the deviation from $\frac{3}{4}V_{dd}$ and realizes a NAND2 function. Accordingly, XNOR2 function of input operands can be realized after the CMOS NAND gate. Now, ReDRAM's MUX can be readily reconfigured through the selectors to assign NOR2/NAND2/buffer/XNOR2 value and its complementary logic to $\overline{\mathrm{BL}}$ and BL, respectively. Based on Eq. (3.1), by setting enable set to 01111, XOR2 result can be produced in a single cycle on the BL.

$$\overline{BL} = \overline{(\overline{D_i.D_j}).(D_i + D_j)} = \overline{D_i.\overline{D_j} + D_j.\overline{D_i}} = D_i \odot D_j \Rightarrow BL = D_i \oplus D_j \qquad (3.1)$$

The transient voltage simulation results of ReDRAM's DRA mechanism to realize single-cycle in-memory operations are shown in Fig. 32. We can observe how NOR2/NAND2/XNOR2 function is produced for two inputs (D_i and D_j). In this case, MUX's selectors are configured to set $\overline{\mathrm{BL}}$ voltage with XNOR2 result as in Fig. 31b. Based on this, the target cell's capacitor is charged to V_{dd} (when D_iD_j=10/01) or

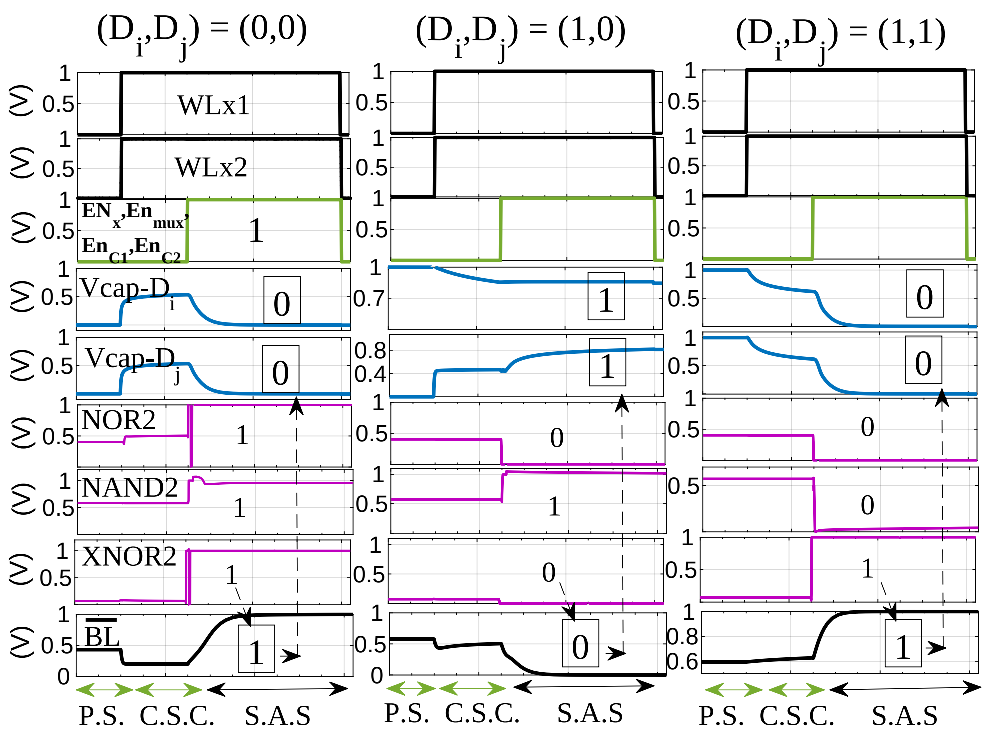


Figure 32: The transient simulation of the internal ReDRAM's sub-array signals in DRA mechanism. Glossary- Vcap-D_i and Vcap-D_J represent the voltage across the two selected DRAM cell's capacitors connected to WLx1 and WLx2. P.S., C.S.S., S.A.S. are short for Precharged State, Charge Sharing State, and Sense Amplification State, respectively.

discharged to GND (when D_iD_j=00/11) during sense amplification state. Therefore, the DRA mechanism can effectively provide single-cycle logic functions (NOT, AND, OR, XOR) and two-cycle complementary logics to address the challenge-2 discussed in Section 3.1.2 by eliminating the need for the row initialization and DCC Rows. Note that, NOT function is readily realized on the BL by selecting the corresponding MUX selectors (01110). In addition, ReDRAM can perform more complex in-memory logic functions (such as XOR2) in a single memory cycle not relying on multiple TRA-based [19] or NOR-based [15] operations.

3.3.2 Software Support

ReDRAM hardware instruction set can be leveraged to realize the functions tabulated in Table 10. Additionally, Table 10 lists the corresponding function implementation in Ambit [19] and DRISA [15] platforms. The micro and control transfer instructions are not discussed here. Similar to the GraphiDe [7], ReDRAM instructions are developed based on AAP primitives. To enable a processor to communicate with ReDRAM efficiently, two types of AAP-based instructions are designed:

1- AAP (src, des, size) that runs the following commands sequence: 1) ACTIVATE a source address (src); 2) ACTIVATE a destination address (des); 3) PRECHARGE to prepare the array for the next access. The size of input vectors for in-memory computation must be a multiple of DRAM row size, otherwise the application must pad it with dummy data. The type-1 instruction is mainly used for *copy* function;

2- AAP (src1, src2, des, opcode, size) that performs DRA method by activating two source addresses (src1 and src2) and then writes back the result on a destination address (des) according to the opcode. Here opcode corresponds to the MUX's selectors (En_{C1} and En_{C2}), as shown in Fig. 31a.

For instance, in order to implement the XOR2-in-memory, as tabulated in Table 10, ReDRAM first copies the input operands from data rows to computational rows in two consecutive cycles using AAP-type-1 and then performs the operation in a single cycle using AAP-type-2 by setting the opcode to 11. A similar operation requires at least 7 consecutive cycles based on the Ambit's TRA mechanism.

Table 10: The basic functions supported by ReDRAM, Ambit and DRISA.

		Command Sequence		
Func.	Operation	ReDRAM	Ambit [19]	DRISA‡ [15]
Copy	$D_r \leftarrow D_i$	AAP(D_i, D_r)†	AAP(D_i, D_r)	AP$(D_i, Latch)$ AP$(Latch, D_r)$
NOT	$D_r \leftarrow \overline{D_i}$	AAP$(D_i, D_r, 10)$	AAP$(D_i, dcc2)$ AAP$(dcc1, D_r)$	AAP$(D_i, dcc2)$ AAP$(dcc1, D_r)$
AND	$D_r \leftarrow D_i.D_j$	AAP$(D_i, x1)$ AAP$(D_j, x2)$ AAP$(x1, x2, D_r, 01)$	AAP$(D_i, x1)$ AAP$(D_j, x2)$ AAP$(0, x3)$* AAP$(x1, x2, x3, D_r)$**	AP$(D_i, Latch)$ AAP$(D_j, x1)$ AAP$(Latch, x1, D_r)$
OR	$D_r \leftarrow D_i + D_j$	AAP$(D_i, x1)$ AAP$(D_j, x2)$ AAP$(x1, x2, D_r, 00)$	AAP$(D_i, x1)$ AAP$(D_j, x2)$ AAP$(1, x3)$* AAP$(x1, x2, x3, D_r)$**	N/A
XOR2	$D_r \leftarrow D_i \oplus D_j$	AAP$(D_i, x1)$ AAP$(D_j, x2)$ AAP$(x1, x2, D_r, 11)$	AAP$(D_i, x1, dcc2)$ AAP$(D_j, x2, dcc4)$ AAP$(0, x3, x4)$* AP$(dcc1, x2, x3, x2)$** AP$(dcc3, x1, x4, x1)$** AAP$(1, x3)$* AAP$(x1, x2, x3, D_r)$**	N/A

† Size of input vectors are not shown here. ‡ DRISA's 1T1C-logic is realized with add-on AND2 gate. Therefore, it can not implement other functions. *Row initialization steps. **TRA steps.

3.3.3 Reliability

A comprehensive circuit-level simulation is run to study the effect of process variation on both ReDRAM's DRA and TRA methods considering different noise sources and variation in all components including DRAM cell (BL/WL capacitance and transistor, shown in Fig. 28) and SA (width/length of transistors-V_s). The Monte-Carlo simulation is conducted with 45nm NCSU Product Development Kit (PDK) library [99] in Cadence Spectre (DRAM cell parameters were taken and scaled from Rambus [110]) under 10000 trials by increasing the amount of variation from ±0% to ±30% for each method. Table 11 shows the percentage of the test error in each variation. Again, considering a significant ±10% variation, the percentage of erroneous DRA across 10000 trials is zero, where the TRA method shows a failure with 0.18%. There-

Table 11: Process variation analysis.

Variation	±5%	±10%	±15%	±20%	±30%
ReDRAM's DRA	0.00%	0.00%	1.2%	9.6%	16.4%
Ambit's TRA	0.00%	0.18%	5.5%	17.1%	28.4%

fore, ReDRAM offers a solution to alleviate challenge-3 by showing acceptable voltage margin in performing operations based on the DRA mechanism.

3.3.4 Raw Performance

To assess the performance of ReDRAM as a new PIM platform, a comprehensive circuit-architecture evaluation framework[6] and two in-house simulators are developed. 1- At the circuit level, we developed ReDRAM's sub-array with new peripheral circuity (SA, MRD, etc.) in Cadence Spectre with 45nm NCSU Product Development Kit (PDK) library [99] in to verify the DRA mechanism and achieve the performance parameters. 2- An architectural-level simulator is built on top of Cacti [112]. The circuit-level results were then fed into our simulator. It can change the configuration files corresponding to different array organizations and report performance metrics for AAP-based PIM operations. The memory controller circuits are designed and synthesized by Design Compiler [113] with a 45nm industry library. 3- A behavioral-level simulator is developed in Matlab to calculate the latency and energy that ReDRAM spends on different tasks. Besides, it has a mapping optimization framework to maximize the performance according to the available resources.

[6]The evaluation framework is introduced in Section 4.2.

3.3.4.1 Throughput

The ReDRAM's raw performance is evaluated and compared with various computing units and accelerators, including a Core-i7 Intel CPU [114] and an NVIDIA GTX 1080Ti Pascal GPU [115]. In PIM domain, the comparison shall be restricted to four recent processing-in-DRAM platforms, Ambit [19], DRISA-1T1C [15], DRISA-3T1C [15], and HMC 2.0 [116], to handle four bulk bit-wise operations, i.e., NOT, AND2, OR2, and XOR2. To have a fair comparison, the ReDRAM's and other PIM platforms' raw throughput was reported when implemented with the same 8 banks with 1024$\times$256 computational sub-arrays. The Intel CPU consists of 4 cores and 8 threads working with two 64-bit DDR4-1866/2133 channels. The Pascal GPU has 3584 CUDA cores running at 1.5GHz [115] and 352-bit GDDR5X. The HMC has 32-10 GB/s bandwidth vaults. Accordingly, an in-house micro-benchmark was designed to run the operations repeatedly for $2^{27}/2^{28}/2^{29}$-bit length input vectors and report the throughput of each platform, as shown in Fig. 33a-d.

It can be observed that 1) either the external or internal DRAM bandwidth has limited the throughput of the CPU, GPU, and even HMC platforms. However, HMC outperforms the CPU and GPU with $\sim$25$\times$ and 6.5$\times$ higher performance on average for bulk bit-wise operations. PIM platforms also achieve remarkable throughput compared to von-Neumann computing systems (CPU/GPU) by unblocking the data movement bottleneck. ReDRAM shows on average 54$\times$ and 7.1$\times$ better throughput compared to CPU and GPU, respectively. 2) While the ReDRAM, Ambit, and DRISA platforms achieve almost the same performance on performing bulk bit-wise NOT function, shown in Fig. 33a, ReDRAM outperforms other PIMs in performing AND2, OR2, and XOR2 operations. As for XOR2, our platform improves the

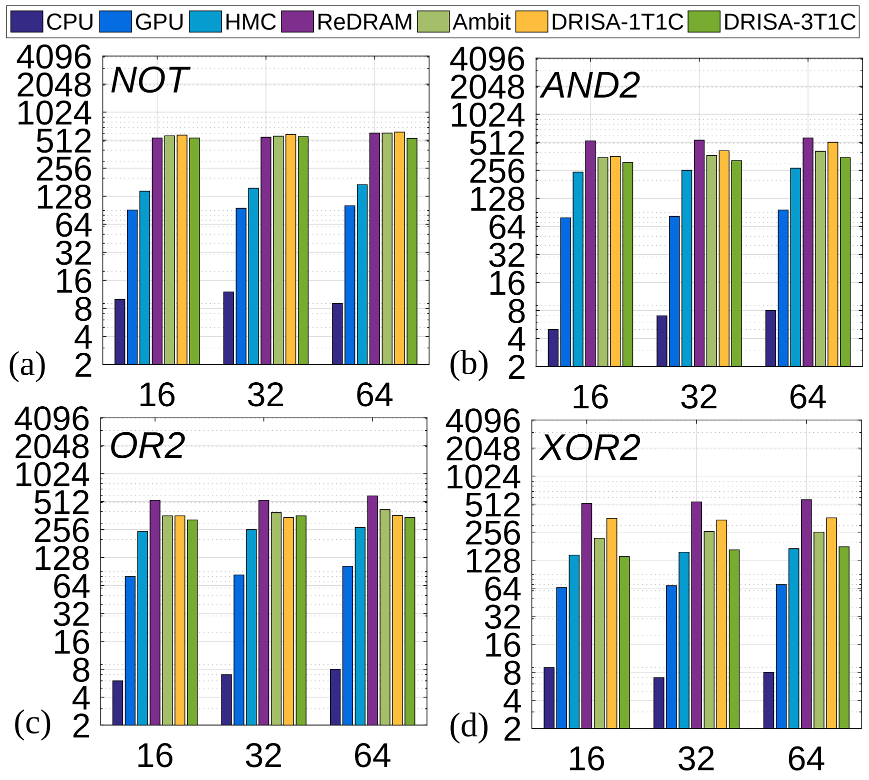


Figure 33: Throughput of (a) NOT, (b) AND2, (c) OR2, and (d) XOR2 operations implemented by different platforms. X-axis: Vector Size (MB) and Y-axis: Log Scaled Throughput (GOps/second).

throughput on average by 2.3×, 1.9×, 3.7× compared with Ambit [19], DRISA-1T1C [15], and DRISA-3T1C [15], respectively. This mainly comes from the DRA mechanism that eliminates the need for row the initialization in Ambit and multi-cycle DRISA mechanism. Note that the add-on logic of DRISA-1T1C is developed with the corresponding logic in the plots [15]; however, in practice, only one single logic can be accelerated with this platform. That is why DRISA-1T1C shows the second-best performance in performing bulk bit-wise XOR2 operation. To sum it up, ReDRAM's DRA mechanism could effectively address challenge-1 by proposing the high-through bulk bit-wise X(N)OR operation.

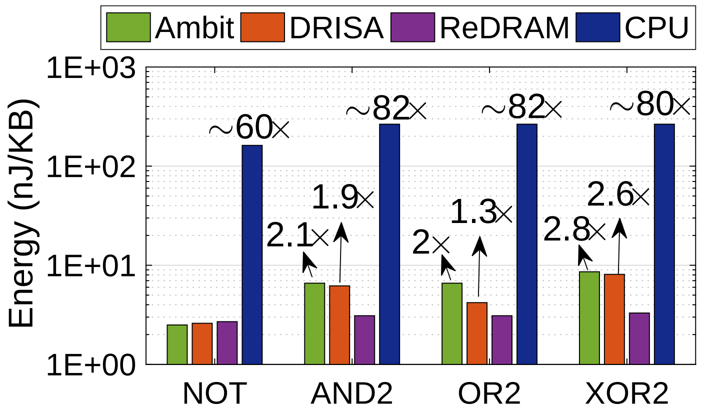


Figure 34: Energy of different platforms (Y-axis: log scale).

3.3.4.2 Energy

The DRAM chip's energy consumption to perform the four bulk bit-wise operations per Kilo-Byte was measured for ReDRAM, Ambit [19], DRISA-3T1C [15], and CPU[7]. Fig. 34 shows that ReDRAM achieves 2.6× and 2.8× energy reduction over Ambit [19] and DRISA-3T1C [15], respectively, to perform bulk bit-wise XOR2 operation. Besides, compared with copying data through the DDR4 interface, ReDRAM reduces the energy by ~80×. As for bit-wise in-memory AND2 operation, ReDRAM outperforms TRA-based Ambit, NOR-based DRISA-3T1C, and CPU, respectively, with ~2.1×, 1.9×, and 82× reduction in energy consumption.

3.3.4.3 Area Overhead

To estimate the area overhead of ReDRAM on top of commodity DRAM chip, three hardware cost sources must be taken into consideration. First, add-on transistors

[7]This energy doesn't involve the energy that processor consumes to perform the operation.

to SAs; each SA requires 30 additional transistors connected to each BL in our design. Second, the 3:8 MRD overhead; the WL driver was modified by adding two more transistors in the typical buffer chain, as depicted in Fig. 29b, so there are only 16 add-on transistors for computational rows. Third, the Ctrl's overhead to control enable bits; ctrl generates the activation bits with MUX units with 6 transistors. To sum it up, ReDRAM imposes 31 DRAM rows (31×256 transistors) per sub-array, at the most, which can be interpreted as $\sim$14% of DRAM chip area. Note that Ambit design requires DCC rows with two WL associated with each; based on the estimation made by [104], each DCC row imposes roughly one transistor over regular DRAM cell to each BL. Besides, DRISA-3T1C [15] requires two add-on transistors per cell, which essentially triple the area overhead.

3.4 Summary

In this chapter, we presented GraphiDe [7] and ReDRAM [6] as two promising PIM platforms based the commodity DRAM chip. Both designs are dedicated to eliminating the need for row initialization and multi-cycle operation in the previous designs by introducing new dual/ quintuple row activation mechanisms. GraphiDe imposes a very low $\sim$1.3% area overhead over the commodity DRAM chip supporting single-cycle AND2/OR2 operations, while ReDRAM shows a $\sim$14% overhead supporting a full-set of 1- and 2-input in-memory operations. ReDRAM can be leveraged to greatly reduce energy consumption and latency of complex in-DRAM logic computations relying on state-of-the-art mechanisms based on triple-row activation, dual-contact cells, row initialization, NOR style, etc. It achieves a considerably higher performance through unblocking the data transfer issue by $54\times$/ $7.1\times$ better throughput as opposed to von-Neumann computing systems, CPU/GPU. Besides, ReDRAM out-performs other PIMs in performing X(N)OR-based operations by up to 3.7$\times$ higher throughput. From the energy consumption perspective, ReDRAM reduces the DRAM chip energy by 2.6$\times$ compared with Ambit [19] and $\sim$80$\times$ compared to data copying via the DDR4 interface.

Chapter 4

PROCESSING-IN-MEMORY ACCELERATION OF DEEP NEURAL NETWORKS

4.1 Introduction

Deep Convolutional Neural Network (DNN/CNN) has achieved world-wide attention due to outstanding performance in image recognition over large-scale datasets such as ImageNet [117]. For instance, ResNet shows a prominent recognition accuracy of 96.43%, which is higher than human beings (94.9%). Following the trend, when going more in-depth in DNNs (e.g., ResNet employs 18-1001 layers), memory/computational resources and their communication have faced inevitable limitations. This can be interpreted as “DNN power and memory wall” [118], leading to the development of different approaches to improve DNN efficiency at either algorithm or hardware level.

Estimation of DNN using shallower models, quantizing parameters [119], [120], compressing pre-trained networks [121], and network binarization [122]–[127] are the most widely explored algorithmic approaches. Recent research efforts have significantly reduced both model size and computing complexity by using low bit-width weights, activations, and gradients [119], [120]. For example, Zhou et al. [119] have shown that low bit-width convolution kernels achieved from their quantization method can accelerate both training and inference with almost comparable prediction accuracy as 32-bit counterparts on ImageNet dataset. A DNN basically consists of multiple stacking layers, namely convolution, activation, and pooling. As depicted in Fig. 35 [117],

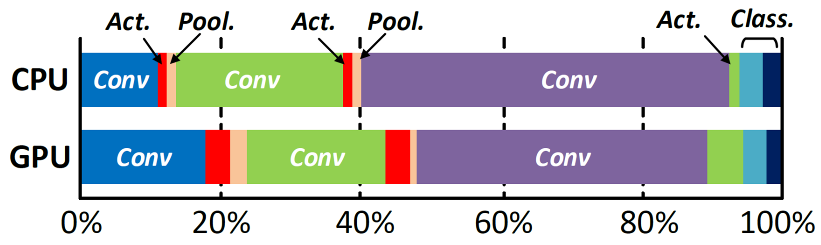


Figure 35: Execution time of a sample DNN for scene labeling on CPU and GPU [117].

the convolutional layer always takes the most fraction (>90%) of execution time and computational sources in both GPU and CPU implementations.

In this chapter, a generic and comprehensive evaluation framework is initially presented to quantitatively analyze the performance of various PIM platforms running big data applications. The framework is then put into the test to quantitatively compare the analog and digital PIM acceleration solutions for DNNs. In addition, the PIM acceleration of DNNs is investigated by proposing in-memory bit-wise adder and in-memory bit-wise convolver schemes based on the MRIMA architecture [1] presented in Chapter 2 to accelerate binary-weight and low bit-width CNNs. Detailed mapping methods are then presented that harness the full potential of PIM capabilities to reduce DNN's data movement overheads. The analysis shows MRIMA is fully capable of realizing DNN-in-memory.

4.1.1 DNN Terminology

DNN is a machine learning classifier that takes an image as input and then computes the probability that image features belong to a sort of output class. Typically, a DNN consists of several convolutional layers and pooling layers followed by Fully-

Connected layers (FC) as depicted in Fig. 36. Note that it has been proven that FC layers could be equivalently implemented by convolutions [119], [122]. Fig. 36 also shows a visualization of the convolutional layer of DNN where each layer receives a set of features organized in multi-channel as input (Input fmaps). It applies kernels (filters) by performing high-dimensional convolutions, i.e., Multiplication-and-Accumulation (MAC) and then produces the features (Output fmaps) for the next layer [128]. The dimensions of both fmaps (input/output) and kernels are 4-D (multiple 3-D structures), and a batch of input fmaps is typically processed by multiple 3-D kernels. After convolution, a non-linear activation function, such as ReLU, will be applied to the results. Considering the shape parameters listed in Table 12, the computation of one convolutional layer can be defined as follow:

$$O[n][k][x][y] = ReLU(B[k] + \sum_{i=0}^{F_h-1} \sum_{j=0}^{F_w-1} \sum_{z=0}^{C-1} I[n][z][U_x + i][U_y + j]W[k][z][i][j]),$$
$$0 \leq n < N, 0 \leq k < K, 0 \leq x < W2, 0 \leq y < H2; \quad (4.1)$$

where O, B, I, and W are the matrices representing output fmaps, Bias, input fmaps, and kernels, respectively. W2/H2 dimensions can be achieved as $W2 = (W1 - F_w + 2P)/S + 1$ and $H2 = (H1 - F_h + 2P)/S + 1$.

Table 12: Shape Parameters of a Convolutional Layer.

Shape Parameter	Description
input fmaps dimension	$W1 \times H1 \times C$
3-D fmaps batch size (input/output)	N
no. of 3-D kernels	K
spatial extent of kernels	$F_w \times F_h \times C$
stride	S
no. of zero padding	P
output fmaps dimension	$W2 \times H2 \times M$

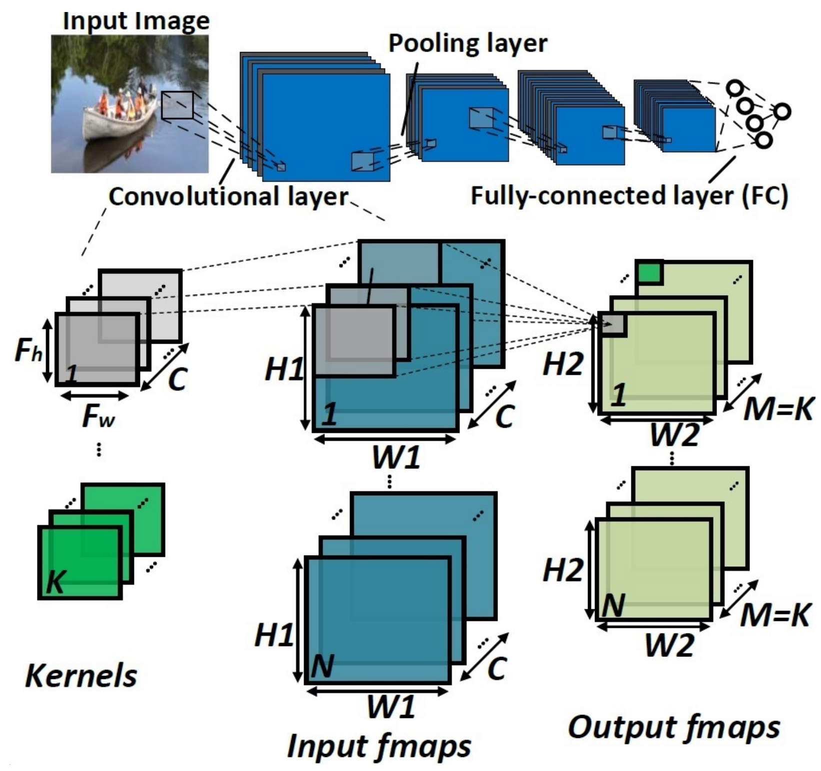


Figure 36: Visualization of inference (a.k.a forward propagation) in DNN.

4.1.2 DNN Acceleration: Analog or Digital PIM Approach?

For DNN acceleration in memory, analog resistive cross-bar memory, as one of the most popular memory array structures, has drawn significant interest due to its high memory accessing bandwidth and *in-situ* computing capability [14], [129]. More importantly, its current-mode weighted summation operation intrinsically matches the dominant MAC in the artificial neural network, making it one of the most promising candidates as the basic computing unit for neural network accelerator design [14]. For example, ISAAC [34] architecture improves throughput and energy by 14.8× and 5.5×, respectively, relative to a well-known ASIC architecture. PipeLayer [85] achieves the speed-up and energy saving of 42.45× and 7.17×, respectively, compared with a

GPU platform on average. However, many non-ideal effects, such as IR-drop (i.e., wire resistance), Stuck-At-Fault (SAF), thermal noise, and random telegraph noise [130], are limiting the progress of hardware implementation of large-scale DNNs on ReRAM crossbar-based accelerators. Many recent works have investigated such issues with either hardware or software solutions [131], [132].

As an alternative solution to realize massive MAC and memory operations in DNN deployments, researchers have come up with quantized/binarized DNNs, through constraining weights and activations of DNN to be quantized/ binarized in forward propagation [119]. These modifications convert the conventional MAC operation to much simpler bulk bit-wise operations (based addition/subtraction [46], [125], [133] or comparison [28]) that can be accelerated in the content of digital memories. For example, Neural Cache [30], as an SRAM-based platform improves inference latency by $18.3\times$ over the state-of-the-art multicore CPU (Xeon E5) and $7.7\times$ over server-class GPU. DRISA [15], as a DRAM-based platform, employs 3T1C- and 1T1C-based computing mechanisms and achieves $7.7\times$ speed-up and $15\times$ better energy-efficiency over GPUs for DNN accelerations. CMP-PIM [28] as an MRAM-based platform achieves $\sim 10\times$ better energy-efficiency compared to DNN-ReRAM accelerators.

While the respective benefits of the aforementioned DNN acceleration-in-memory approaches (i.e., analog and digital) are well known, it still lacks cross-technology comparison and analysis. In this chapter, a comprehensive and universal cross-layer evaluation platform is developed [1], [9], [134] to quantitatively compare and analyze the analog and digital approaches for DNN acceleration-in-memory schemes. The presented framework will be also used to analyze the performance of various big data applications such as bioinformatics DNA alignment and graph processing in Chapters 5 and 6, respectively.

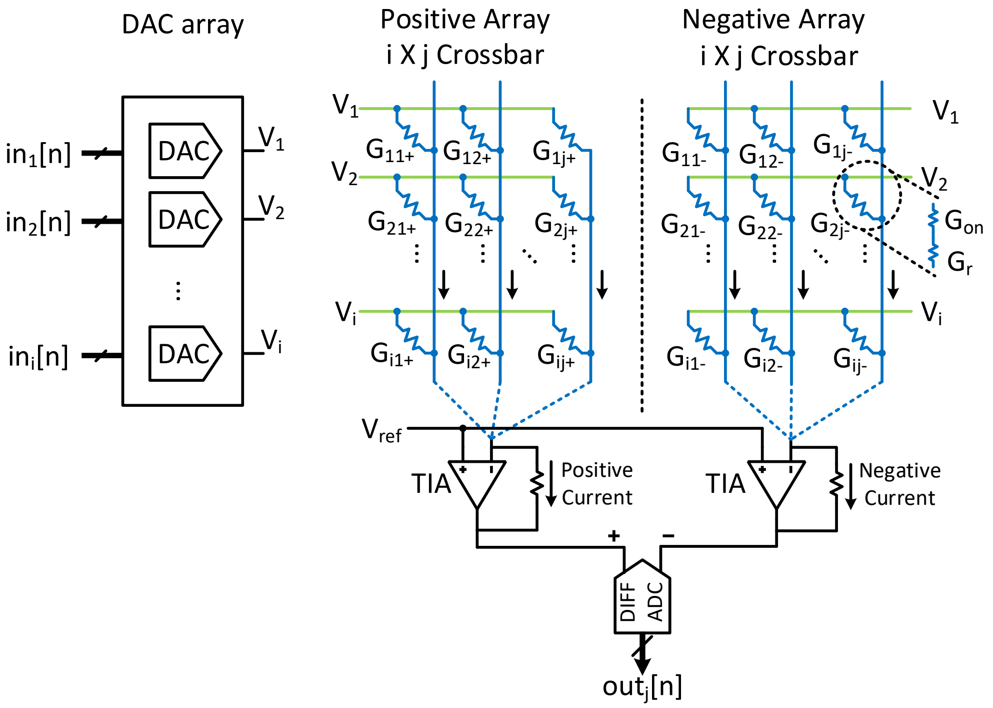


Figure 37: Hardware implementation of a single $M \times M$ ReRAM crossbar array pair (positive and negative array) as an analog dot-product engine [130], [135].

The primary computation performed by an analog ReRAM cross-bar is the current-mode weighted summation operation (i.e., dot-product). The architecture of the cross-bar and its peripheral circuits are described in Fig. 37. Note that the positive and negative array setup is widely used in crossbar-based dot-product engine [84], [130], [136], [137] for performing convolution computation with positive and negative kernel values. As shown in Fig. 37, the n-bit binary bit-strings $in_i[n]$ are the inputs to the crossbar array, which is first converted by the Digital-to-Analog Converter (DAC) array into voltages V_i. Since the reference voltage V_{ref} is set to $\frac{V_{dd}}{2}$, the current forward into the differential ADC in the j-th column pair (i.e., two corresponding columns in the positive and the negative array) can be described as:

$$I_{ADC,j} = \sum_{i=1}^{M} \Big((V_i - V_{ref}) \cdot (G^{+}_{i,j} - G^{-}_{i,j}) \Big) \tag{4.2}$$

where $G^{\pm}_{i,j}$ is the conductance of ReRAM cell indexed by i and j in the positive and negative array respectively. As can be seen, Eq. (4.2) performs the dot-product computation between two vectors $\boldsymbol{V} - V_{ref}$ and $\boldsymbol{G}^{+}_{:,j} - \boldsymbol{G}^{-}_{:,j}$. However, for using the ReRAM crossbar array to accelerate the dot-product computation in DNN, a software-hardware co-design is essential, since mapping the DNN parameters into the crossbar-based accelerator requires a series of signal conversions as introduced in [85], [130].

4.2 Proposed Bottom-up PIM Evaluation Framework

As various data-intensive applications with distinct workload sizes and memory access patterns are expected to benefit from processing-in-memory in both cache and main memory levels, selecting the right design for a particular application is a complex task. Besides, by choosing a PIM design, it is imperative to establish uniform evaluation conditions to make an impartial choice between available design options. To perform the cross-technology comparison among aforementioned PIM techniques, we have developed a comprehensive bottom-up cross-layer framework [9], [41], [134] shown in Fig. 38.

1- For *Device level* modeling, the device parameters are first extracted from different assessments and models. The Non-Equilibrium Green's Function (NEGF), and Landau-Lifshitz-Gilbert (LLG) equations are used as explained in Chapter 2 to model STT-MRAM and SOT-MRAM bitcell (indicated under MRAM in Fig. 38) [13], [74]. Large numbers of physical parameters are integrated into the compact model to achieve a good agreement with experimental measurements. The default ReRAM and SRAM .cell configurations of NVSIM [138] are considered for evaluation. Moreover, DRAM cell parameters are taken from Rambus [110] and scaled.

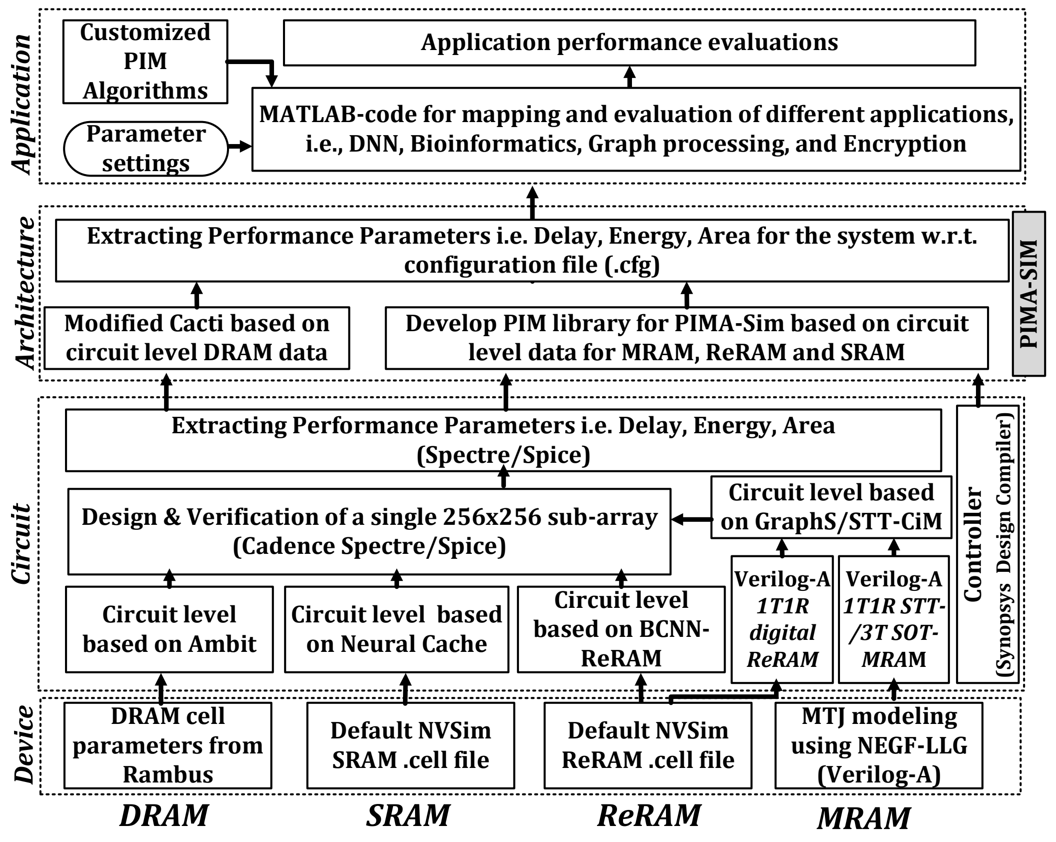
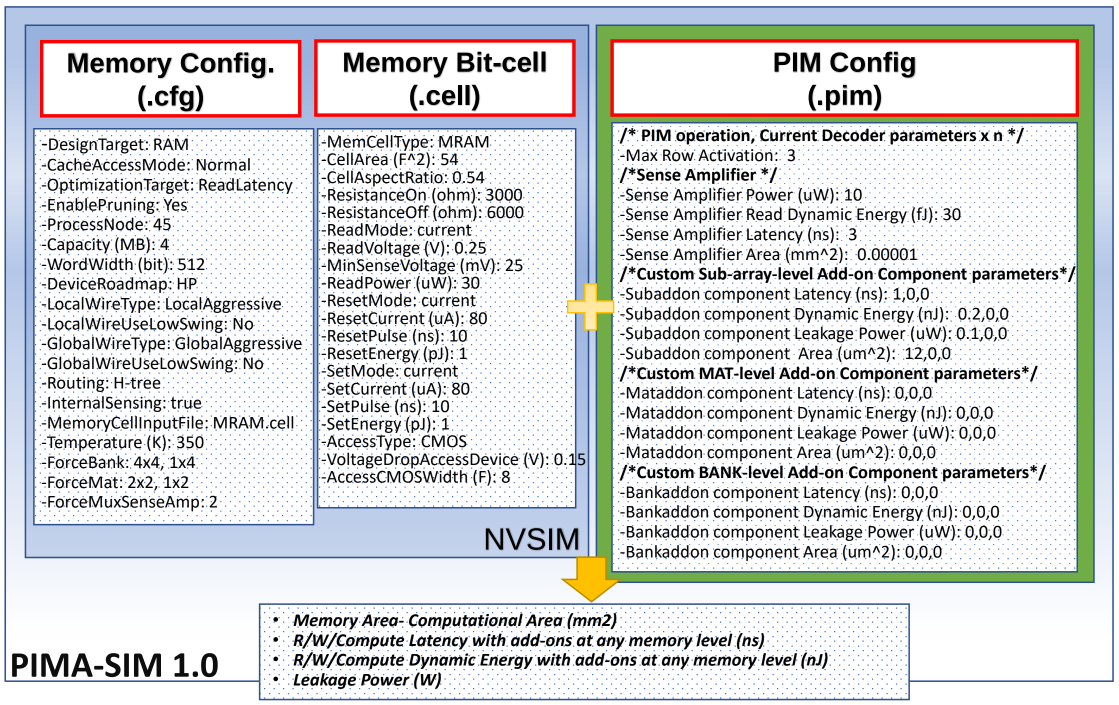

Application
Customized PIM Algorithms
Application performance evaluations
Parameter settings
MATLAB-code for mapping and evaluation of different applications, i.e., DNN, Bioinformatics, Graph processing, and Encryption
Architecture
Extracting Performance Parameters i.e. Delay, Energy, Area for the system w.r.t. configuration file (.cfg)
Modified Cacti based on circuit level DRAM data
Develop PIM library for PIMA-Sim based on circuit level data for MRAM, ReRAM and SRAM
PIMA-SIM
Circuit
Extracting Performance Parameters i.e. Delay, Energy, Area (Spectre/Spice)
Design & Verification of a single 256x256 sub-array (Cadence Spectre/Spice)
Circuit level based on GraphS/STT-CiM
Controller (Synopsys Design Compiler)
Circuit level based on Ambit
Circuit level based on Neural Cache
Circuit level based on BCNN-ReRAM
Verilog-A 1T1R digital ReRAM
Verilog-A 1T1R STT-/3T SOT-MRAM
Device
DRAM cell parameters from Rambus
Default NVSim SRAM .cell file
Default NVSim ReRAM .cell file
MTJ modeling using NEGF-LLG (Verilog-A)
DRAM
SRAM
ReRAM
MRAM

Figure 38: The bottom-up evaluation framework developed for PIM platform evaluation.

2- For *Circuit level* simulation, the memory sub-array with peripheral circuity (SA, MRD, MCD, etc.) could be implemented based on a particular PIM style for each technology on top of the device level data. In this chapter for DNN acceleration, the GraphS [4] PIM style is used for SOT-MRAM and digital ReRAM implementations; STT-CiM [89] as the STT-MRAM design; BCNN-ReRAM [84] design for analog ReRAM crossbar; Neural Cache [30] design for SRAM and Ambit [19] design for DRAM are accordingly used. The memory sub-arrays are simulated in Cadence Spectre with 45nm NCSU Product Development Kit (PDK) library [99] to verify the PIM's circuit functionality and achieve the circuit performance parameters. The memory



```
-DesignTarget: RAM
-CacheAccessMode: Normal
-OptimizationTarget: ReadLatency
-EnablePruning: Yes
-ProcessNode: 45
-Capacity (MB): 4
-WordWidth (bit): 512
-DeviceRoadmap: HP
-LocalWireType: LocalAggressive
-LocalWireUseLowSwing: No
-GlobalWireType: GlobalAggressive
-GlobalWireUseLowSwing: No
-Routing: H-tree
-InternalSensing: true
-MemoryCellInputFile: MRAM.cell
-Temperature (K): 350
-ForceBank: 4x4, 1x4
-ForceMat: 2x2, 1x2
-ForceMuxSenseAmp: 2
```

```
-MemCellType: MRAM
-CellArea (F^2): 54
-CellAspectRatio: 0.54
-ResistanceOn (ohm): 3000
-ResistanceOff (ohm): 6000
-ReadMode: current
-ReadVoltage (V): 0.25
-MinSenseVoltage (mV): 25
-ReadPower (uW): 30
-ResetMode: current
-ResetCurrent (uA): 80
-ResetPulse (ns): 10
-ResetEnergy (pJ): 1
-SetMode: current
-SetCurrent (uA): 80
-SetPulse (ns): 10
-SetEnergy (pJ): 1
-AccessType: CMOS
-VoltageDropAccessDevice (V): 0.15
-AccessCMOSWidth (F): 8
```

```
/* PIM operation, Current Decoder parameters x n */
-Max Row Activation: 3
/*Sense Amplifier */
-Sense Amplifier Power (uW): 10
-Sense Amplifier Read Dynamic Energy (fJ): 30
-Sense Amplifier Latency (ns): 3
-Sense Amplifier Area (mm^2): 0.00001
/*Custom Sub-array-level Add-on Component parameters*/
-Subaddon component Latency (ns): 1,0,0
-Subaddon component Dynamic Energy (nJ): 0.2,0,0
-Subaddon component Leakage Power (uW): 0.1,0,0
-Subaddon component Area (um^2): 12,0,0
/*Custom MAT-level Add-on Component parameters*/
-Mataddon component Latency (ns): 0,0,0
-Mataddon component Dynamic Energy (nJ): 0,0,0
-Mataddon component Leakage Power (uW): 0,0,0
-Mataddon component Area (um^2): 0,0,0
/*Custom BANK-level Add-on Component parameters*/
-Bankaddon component Latency (ns): 0,0,0
-Bankaddon component Dynamic Energy (nJ): 0,0,0
-Bankaddon component Leakage Power (uW): 0,0,0
-Bankaddon component Area (um^2): 0,0,0
```

- *Memory Area- Computational Area (mm2)*
- *R/W/Compute Latency with add-ons at any memory level (ns)*
- *R/W/Compute Dynamic Energy with add-ons at any memory level (nJ)*
- *Leakage Power (W)*

Figure 39: PIMA-SIM as a PIM support evaluation tool developed to model the timing, energy, and area of various PIM technologies.

controller circuits for all platforms are synthesized by Design Compiler [113] with the same 45nm industry library.

3- For *Architecture level*, a PIM support evaluation tool is developed for the NVSIM [138] named PIMA-SIM as shown in Fig. 39 to model the timing, energy, and area of various PIM technologies based on STT-MRAM, STT-MRAM, PCM, ReRAM, and SRAM. This tool offers the same flexibility in memory configuration in terms of bank/mat/subarray organization and peripheral circuitry design as NVSIM, while supporting PIM-level configurations. PIMA-SIM can be configured using three configuration files. At the cell level, it uses NVSIM's .cell file to save the device-circuit level info. At the architecture level, it uses NVSIM's .cfg file to configure the memory organization and optimization target. In addition, as depicted in Fig. 39, at the PIM level, PIMA-SIM's .pim file is designed to save the PIM-level parameters. PIMA-SIM's .pim file gives the following flexibilities to study the PIM behaviors: 1- users can specify the number of row activation for PIM purposes; 2- users can insert their customized sense amplifier designs; and 3- users can add any number of customized add-ons at the sub-array/mat/bank level. The PIM libraries are accordingly developed for each platforms on top of NVSIM [138] and Cacti [139] based on device/circuit level data. Accordingly, the performance data (i.e., latency, energy, and area) could be extracted for different PIM platforms w.r.t. a single input memory configuration file (.cfg).

4- For *Application level* simulations, a behavioral-level simulator is developed in Matlab, taking architecture-level results as well as the proposed customized in-memory algorithm for various big data applications to calculate the latency, energy, and area that different PIM platforms spend on them. It has a mapping optimization framework to maximize the performance w.r.t. the available resources.

4.2.1 Performance Analysis

In this section, two different experiments under ISO-Capacity and ISO-Computation constraints are conducted to quantitatively compare and analyze the analog and digital acceleration-in-memory approaches for DNNs.

4.2.1.1 ISO-Memory-Capacity Comparison

The performance of digital and analog PIM platforms with an ISO-memory-capacity constraint is initially studied. A 32Mb, single bank unit based on digital (SOT-MRAM, STT-MRAM, ReRAM, SRAM, and DRAM) and analog ReRAM cross-bar is developed with the presented bottom-up evaluation framework. Table 13 reports eleven performance parameters for each platform. The observations on this experiment are listed below.

4.2.1.1.1 Area

The area metric was divided into two parts: memory die area (M), and computational area (C) which includes controller, modified decoder, SA, 8-bit ADC for the relevant analog ReRAM crossbar, etc. In terms of memory die area, the digital PIM platforms impose a relatively larger area than analog ReRAM cross-bar except for STT-MRAM design [89]. However, if we take the computational area into account, the ReRAM cross-bar consumes 2.5 mm^2, which is much larger than that of digital counterparts, such as digital ReRAM (0.4 mm^2). Accordingly, a memory to computational area ratio as M/C can be defined. The M/C ratio equals 23.53 for SOT-MRAM-based PIM, while the analog ReRAM cross-bar shows a ratio of 1.33. The low M/C ratio of ReRAM cross-bar is the consequence of large peripheral circuit's overhead, such as buffers and DAC/ADC, which contributes more than 85% of the computational area [14], [84]. Furthermore, according to the results reported in Table 13, the STT-MRAM and SRAM platforms occupy the smallest and the largest overall area, respectively, compared to other PIM counterparts.

4.2.1.1.2 Latency

As listed in Table 13, the analog ReRAM cross-bar achieves the shortest read latency (1.48ns) as compared with digital platforms. Still, it has the longest write latency (20.9ns). The SOT-MRAM platform achieves the shortest write latency compared to

Table 13: Per operation estimation results for different PIM designs. In the Area part, M denotes Memory die area, and C denotes Computation area overhead. (iso-capacity: 32Mbit-single Bank, Data Width: 512-bit).

	Digital					Analog
Metrics	SOT-MRAM†	STT-MRAM‡	ReRAM†	SRAM*	DRAM§	ReRAM**
Non-volatility	Yes	Yes	Yes	No	No	Yes
Area (mm^2)	M: 7.06 C: 0.3	M: 2.14 C: 0.3	M: 3.92 C: 0.4	M: 10.38 C: 0.5	M: 4.53 C: 0.04	M: 3.34 C: 2.5
Read Latency (ns)	2.85	1.90	1.65	2.9	3.4	1.48
Write Latency (ns)	2.59	5.29	19.8	2.7	3.4	20.9
Read Dynamic Energy (nJ)	0.57	0.37	0.76	0.34	0.66	0.38
Write Dynamic Energy (nJ)	0.66	0.67	2.9	0.38	0.66	2.7
(N)AND/(N)OR Computation Energy (nJ)	~0.64	~0.46	~1.13	~0.59	~0.75	1.96 per MAC
Full Adder Computation Energy (nJ)	~1.92	~1.59	~3.4	~1.18	~11.25	
Leakage Power (mW)	550	410.2	362.4	5243	335.5	587.6
Endurance	~10^{10} - 10^{15}	~10^{10} - 10^{15}	~10^{5} - 10^{10}	Unlimited	10^{15}	~ 10^{5} - 10^{10}
Data over-written issue	No	No	No	No	Yes	No

†implemented based on [4]. ‡implemented based on [89]. * implemented based on [30]. §implemented based on [19].

**implemented based on [84].

other technologies and has a higher endurance (10^{10}-10^{15}) compared to ReRAM-based platforms.

4.2.1.1.3 Energy

Based on Table 13, SOT-MRAM and STT-MRAM platforms consume the smallest write dynamic energy among all the NVM platforms due to its intrinsically low-power device operation. At the same time, SRAM achieves the smallest read and write energy compared to all the platforms. The analog ReRAM cross-bar achieves a close-to-SRAM read dynamic energy, but it consumes a large write dynamic energy. In terms of computational energy, for digital platforms, it is measured based on the PIM's capability to perform (N)AND/(N)OR and full adder functions. As seen from Table 13, the STT-MRAM [89] and SRAM [30] PIM respectively consume the smallest computational energy compared to different technologies to perform different operations, where SOT-MRAM stands as the third most energy-efficient platform. Note that, although the DRAM PIM design based on Ambit [19] consumes 0.75 nJ to perform (N)AND/(N)OR based TRA mechanism, it requires over 14 memory cycles to perform the addition operation to avoid overwriting data, which leads to much higher energy consumption compared to other platforms. For the analog cross-bar, the computational energy per MAC was reported, which is comparable with addition operation in digital SOT-MRAM platform. In terms of leakage power consumption, the digital ReRAM and DRAM can be observed as relatively more power-efficient platforms. Moreover, the SRAM platform consumes $\sim$14.5$\times$ and $\sim$9$\times$ more power than digital and analog ReRAM, respectively.

4.2.1.2 ISO-Computation Comparison

The performance of the digital and analog PIM platforms is further explored for DNN acceleration. Hereby, we took the classical LeNet-5 [140] as a simple example to perform the handwritten digit classification task with MNIST dataset. For correctly mapping the target DNN into the PIM, offline training of the LeNet-5 network was conducted with weight and activation quantization, following the methods presented in [119], [123]. A description model of each platform based on the data reported in Table 13 was then employed in the application level DNN simulator. For fair hardware comparison, the bit-width configuration of [1:8] for [Weight: Activation] was selected, although ReRAM cross-bar-based accelerator supports higher weight bit-width (> 1 bit) with better DNN performance (i.e., classification accuracy in the experiment). No quantization was applied in the first and last layer of DNN, and the full-precision computations were also handled by the PIM-based accelerator. For the sake of simplicity, the estimated performance results (area, energy, latency) of convolutional layers are only reported.

4.2.1.2.1 Area

Contrary to the approach used to report the area in Table 13, we leverage the method presented in [28], [84] to report the results. Specifically, we consider the area overhead due to computation by calculating the number of cross-bars or sub-arrays. Table 14 reports the area for digital and analog PIM platforms by dividing it into the memory and logic parts. We observe that the digital ReRAM and STT-MRAM platforms require the smallest area than other platforms, respectively, mainly due to their single transistor cell structure. It is noteworthy that, while the DRAM platform has one of the least die areas due to its single-transistor cell and owns the least computational area under ISO-capacity constraint due to its almost unchanged peripheral circuitry (1% as listed in Table 13), it requires accessing to multiple sub-arrays to avoid overwriting data problem as well as fitting the network at the same time, resulting in a larger area requirement compared to NVMs. As for the analog cross-bar platform, the logic part contributes $\sim 4\times$ more than the memory area. Overall, it imposes a larger area than that of other digital NVM platforms due to matrix splitting and extra-large add-on area overhead [14].

4.2.1.2.2 Latency

Table 14 summarizes each platform's latency required to process the convolutional layers of the DNN. According to the table, the SRAM platform is the fastest one with 0.7ms latency. This mainly comes from its short read and write latency and fast two-cycle addition scheme [30]. Besides, we observe that the SOT-MRAM platform achieves 0.9ms latency and stands as the second-fastest platform. The DRAM platform shows an extremely long latency mainly due to the excessive copy operations needed to avoid overwriting data, as explained in Chapter 3. The analog cross-bar needs 5.8ms to process the convolutional layers.

4.2.1.2.3 Energy

Table 14 also reports the energy consumption of different platforms. It can be seen that SOT-MRAM and STT-MRAM based platforms save $15.8\times$ and $17.3\times$ energy compared to the analog cross-bar. In addition, the volatile digital memories consume

Table 14: Estimated row performance of various PIMs without parallelism techniques.

	Digital					Analog
Parameters	SOT-MRAM	STT-MRAM	ReRAM	SRAM	DRAM	ReRAM
Area (mm^2) (memory + logic)	0.018 ∼(0.0172+0.0008)	∼0.012 ∼(0.011+0.0008)	0.0097 ∼(0.009+0.0007)	0.64 ∼(0.608+0.032)	0.16 ∼(0.158 + 0.002)	0.06 ∼(0.011+0.049)
Energy (μJ) (write-back+read-based Ops)	0.85 ∼(0.31+0.54)	0.78 ∼(0.25+0.53)	1.9 ∼(0.75+1.15)	1.6 ∼(0.42+1.18)	2.1 ∼(0.8+1.3)	13.5 ∼(0+13.5)
Latency (ms)	0.9	1.8	1.3	0.7	13.5	5.8

much smaller energy than that of the analog platform. Therefore, from energy saving standpoint, digital PIM platforms could be a better choice in comparison to the analog cross-bar. Note that, for PIM platforms, all operands are assumed to be stored in memory. Unlike traditional computation, an extra intermediate data write-back is needed, which has a large effect on the overall energy and latency. Based on this, we split the reported energy into write-back and read-based logic operations energy. The write-back energy involves the energy required to write the weights or inputs into PIM plus the energy required write the computation results back to the memory for computation in the next layer. The read-based operation energy involves the read and bit-line computing energy. The analog cross-bar [84] can accomplish the MAC operation without writing back the intermediate data, that's why we omit the write-back energy for this platform.

4.3 MRIMA as a Bit-wise DNN Inference Accelerator

In this section, we demonstrate that one of our previously-discussed PIM platforms, namely MRIMA [1] in Chapter 2, can accelerate Binary-Weight DNNs (BWNNs) and low bit-width DNNs 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 architecture in Fig. 7. In both networks, except for the inception layer, kernels need to be continuously quantized before mapping into computational sub-arrays. However, quantized shared kernels can be utilized for different inputs. The DPU in Fig. 7 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 fea-

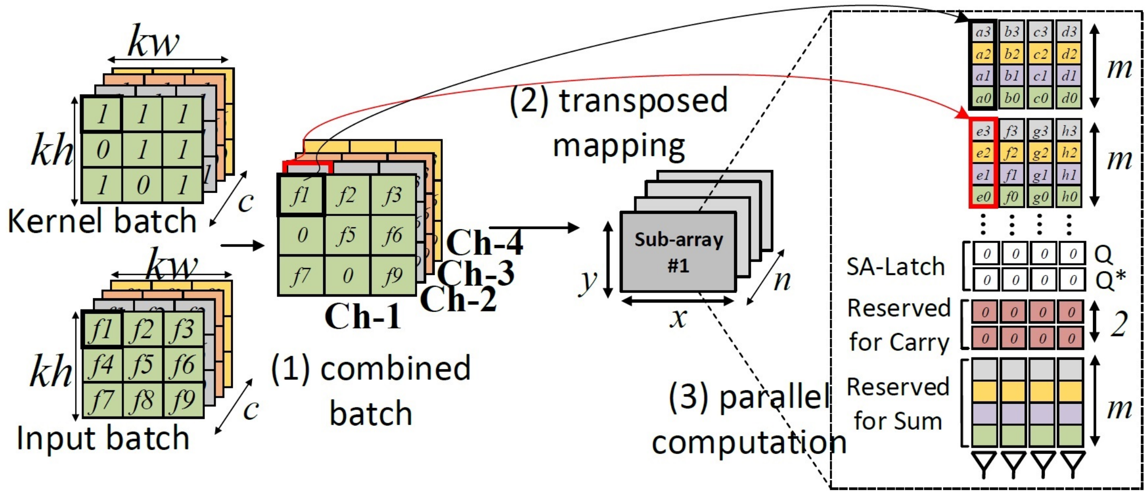


Figure 40: MRIMA's data organization and computation steps of binary-weight layers.

tures using MRIMA's computation methods. Finally, DPU's Active. module activates the generated feature map and completes 3^{rd} step by producing output fmaps.

4.3.1 In-memory Bit-wise Adder

As the main operation of BWNNs, addition (/subtraction) is the most critical unit of the accelerator [84], [122], [141]. This unit must keep high throughput and resource efficiency while handling different input bit-widths at run-time. Therefore, we proposed a parallel in-memory add/sub mapping technique based on MRIMA's 2-cycle in-memory adder to accelerate multi-bit add operation. While there are few designs for in-memory adder/subtractor in literature [15], [38], [49], [88], to the best of our knowledge, MRIMA was the first which presents a fast and fully parallel design in MRAM domain. Fig. 40 shows the requisite data organization and computation steps of binary-weight layers with a straightforward and intuitive example in Fig. 41 only considering add operations. Obviously sub can be implemented based on add.

(1) Initially, c channels (here, 4) in the size of $kh \times kw$ (here, 3×3) are selected from the 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 the MRIMA accelerator operates to produce the output feature maps. The memory sub-array organization for such parallel computation is delineated in Fig. 40 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.

The add operation of two matrices of 4-bit elements (Ch1 and Ch2) is used in Fig. 41 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 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 SAs use the 2 bit cells located in the same bit-lines as input operands for IML23 (see Table 5) and carry latch's data as carry-in to generate sum based on MRIMA's method. During step 2 of Sum (S2), a

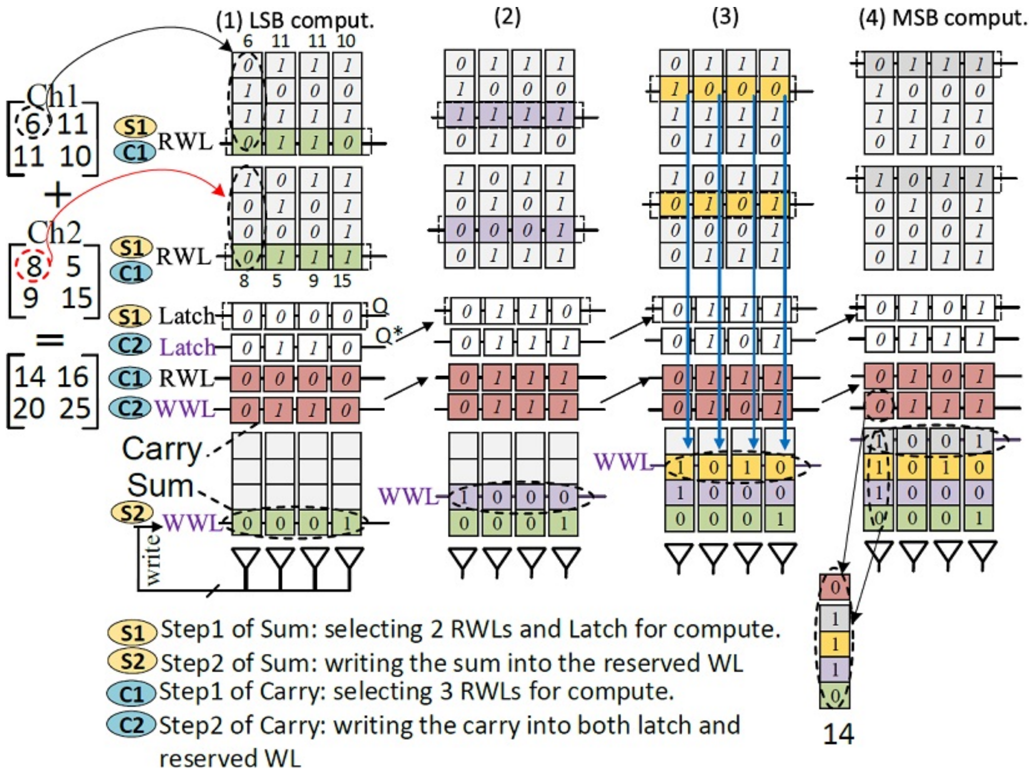

(1) LSB comput.
(2)
(3)
(4) MSB comput.
Ch1
Ch2
RWL
WWL
Latch
Q
Q*
Carry
Sum
write
14
S1 Step1 of Sum: selecting 2 RWLs and Latch for compute.
S2 Step2 of Sum: writing the sum into the reserved WL
C1 Step1 of Carry: selecting 3 RWLs for compute.
C2 Step2 of Carry: writing the carry into both latch and reserved WL

Figure 41: Parallel in-memory addition steps for generating sum and carry-out logic.

WWL is activated to save back the Sum bit using MRIMA's FRC. In step 1 of Carry (C1), the same two operands in conjunction with one of the carry's reserved rows are enabled to generate the carry-out leveraging MRIMA's IML33. During step 2 of Carry (C2), FRC is activated to save back the carry-out bit into a reserved row and 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 the 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>=$< 1{:}m >$).

4.3.2 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 [119]. There are some other layers in DNNs, such as the inception layer (directly taking image as inputs and not necessarily quantized) and FC layer. These layers as discussed, can be equivalently implemented by convolution operations using 1×1 kernels [119]. Thus, all layers could be implemented by convolution computation by exploiting these operations [8], [46], [119]:

$$I * W = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} 2^{m+n} bitcount(AND2(C_n(W), C_m(I))) \tag{4.3}$$

Assume I is a sequence of M-bit input integers (3-bit as an example in Fig. 42) 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 $W = \sum_{n=0}^{N-1} 2^n c_n(W)$.

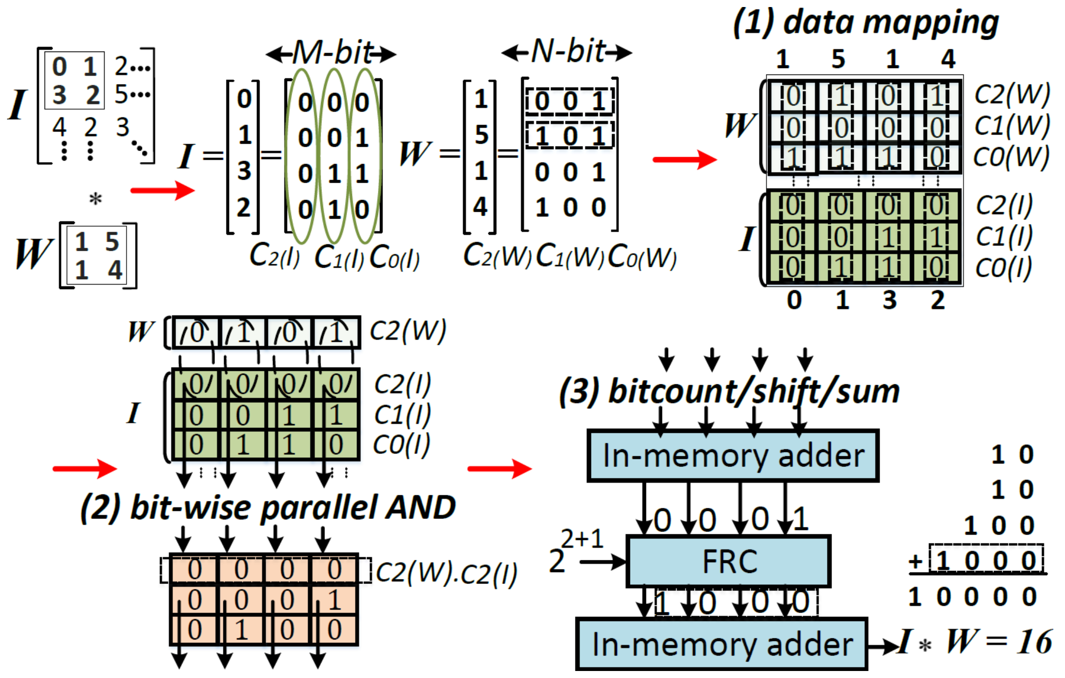


Figure 42: Mapping and computation of MRIMA's bit-wise convolver.

As shown in the data mapping step in Fig. 42, $C_2(W)$-$C_0(W)$ 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. 42. The results of parallel AND operations stored within sub-array will be accordingly processed using bit-counter. Bitcount is translated 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. 42, "0001", produced by in-memory adder is left-shifted by 3-bit ($\times 2^{2+1}$) to "1000". Eventually, an 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>=$< n{:}m >$).

4.3.3 Evaluation

To assess the performance of MRIMA as a new PIM platform, we leveraged the presented comprehensive device-to-architecture evaluation framework in Section 4.2. The device parameters to model STT-MRAM cell are listed in Table 2. At the circuit level, a comprehensive Verilog model for DPU was also developed 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 [142] with case-statement codes. The 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 $y = kx + h$ [124], 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 (see Fig. 7) can fetch each pixel of feature maps, feed 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.

4.3.3.1 Architectural 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.

4.3.3.2 Area and peak performance

The area of MRIMA is 109.6mm^2. The experiments show that, in total, MRIMA imposes 5.6-5.8% area overhead to the memory die, where Pinatubo [23], RIMPA [49], and DRISA [15] 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 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. 43 shows this trade-off considering both computational and power efficiency metrics [34], [85]. With current configuration, the computational efficiency of MRIMA is 1521.83 GOps/s/mm^2 which is higher than PipeLayer-ReRAM [85] (1485), ISAAC-ReRAM [34] (478.9), and DaDianNao-ASIC [143] (63.46). The 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 4.3.3.3 for various under-test platforms.

4.3.3.3 DNN Acceleration Performance

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

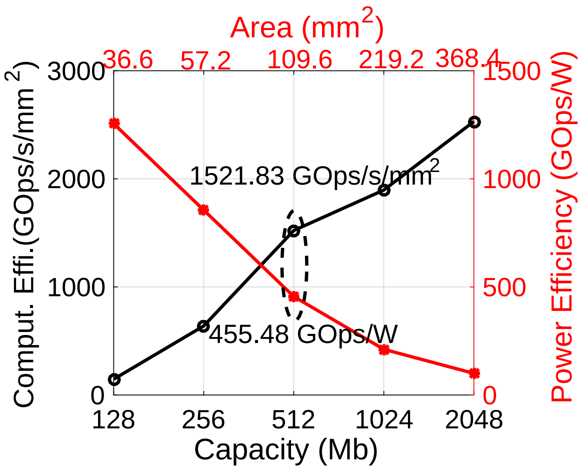


Figure 43: MRIMA's area-peak performance trade-off.

4.3.3.3.1 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 [144] is selected. The images are re-sized to 40×40 and fed to the model. *Model:* A DNN 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:* The open-source algorithm by DoReFa-Net [119] was used where all the operations can be accelerated significantly using the bit-wise convolution of fixed-point integers. The batch normalization and different dropout techniques were adopted to accelerate and avoid over-fitting. The model was trained on TensorFlow [145] with 100 epochs, and the lowest test error of epoch was reported.

4.3.3.3.2 Accelerators' setup

DRAM: A DRISA-like [15] accelerator for binary-weight DNNs was developed. Two different computing methods of DRISA named 3T1C and 1T1C-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 [139] for evaluation of DRAM's solutions. Similar to [15], the controllers and adders were synthesized in Design Compiler [113]. *ReRAM:* A Prime-like [14] 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 sub-arrays, for each mat, there are 256×256 ReRAM cells and eight 6-bit reconfigurable SAs. For evaluation, NVSIM simulator [138] was extensively modified to emulate Prime functionality. Note that the default NVSIM's ReRAM cell file (.cell) was adopted for the assessment. *STT-MRAM:* An STT-CiM-like [89] accelerator was developed with the exact 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 DNNs. *ASIC:* A DaDianNao-like [143] accelerator was developed. To have a fair comparison, two versions with either 8×8 tiles or 16×16 tiles were selected. Accordingly, the designs were synthesized with Design Compiler [113] under 45 nm process node. The eDRAM and SRAM performance was estimated using CACTI [146]. *GPU:* The NVIDIA GTX 1080Ti Pascal GPU with 3584 CUDA cores running at 1.5GHz (11TFLOPs peak performance) was used. The energy consumption was measured with

NVIDIA's system management interface. Similar to [15], we scaled the achieved results by 50% to exclude the energy consumed by cooling, etc. Accordingly, based on the bit-width configuration of <I>, i.e., 1, 2, 4, 8, GPU results were aggressively scaled by $\times 32$, $\times 16$, $\times 8$, and $\times 4$, respectively, to get the peak performance for each quantized networks. Note that GPU doesn't support fixed-point DNN, and the real scale ratio should be less than these numbers [15], [147].

4.3.3.3.3 Accuracy

Fig. 44a tabulates the test error results and relative complexity of the discussed model under various configurations. The 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 [119] that weights, inputs, and gradients are progressively more sensitive to bit-width changes. Fig. 44b depicts the prediction accuracy curve vs. the number of epoch in different configurations. It is observed that the low bit-width networks can keep the accuracy high compared to the original 32-bit case.

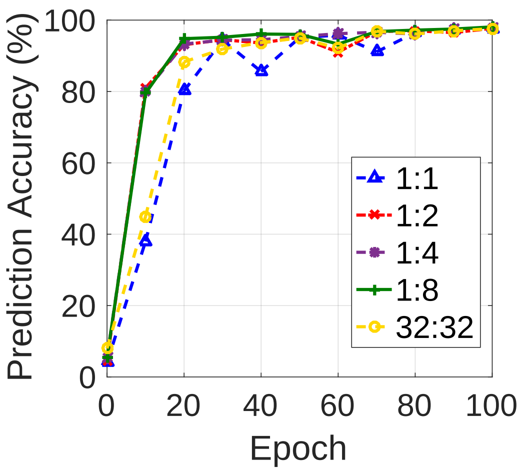
4.3.3.3.4 Energy consumption

Fig. 45 shows the MRIMA's energy-efficiency results on DNN application with a batch size of 8 and 64 in different <W:I>. As shown, the MRIMA solution offers the highest energy-efficiency normalized to the area compared to others owing to its energy-efficient and parallel operations. It can be observed that MRIMA achieves $\sim 1.8\times$ and $2.1\times$ higher energy-efficiency than that of DRAM-3T1C and 1T1C-adder, respectively. The main reason is the energy-efficiency of operations in MRIMA; as dis-

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 [119] and it is already reported in previous works.

(a)



(b)

Figure 44: (a) Test error of the DNN model, (b) Prediction accuracy vs. epoch.

cussed 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. 45 shows that MRIMA solution is 1.7× more energy-efficient than the best ASIC solution. Besides, it shows ∼8.5× saving in energy compared to the ReRAM solution. It is worth pointing out that MRIMA doesn't follow the conventional ReRAM-based cross-bar designs to realize DNN-in-memory, which brings significant energy-efficiency due to eliminating DAC/ADC units. Compared to the STT-CiM counterpart, MRIMA obtains on average 1.4× higher energy-efficiency normalized to the area. Besides, the 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.

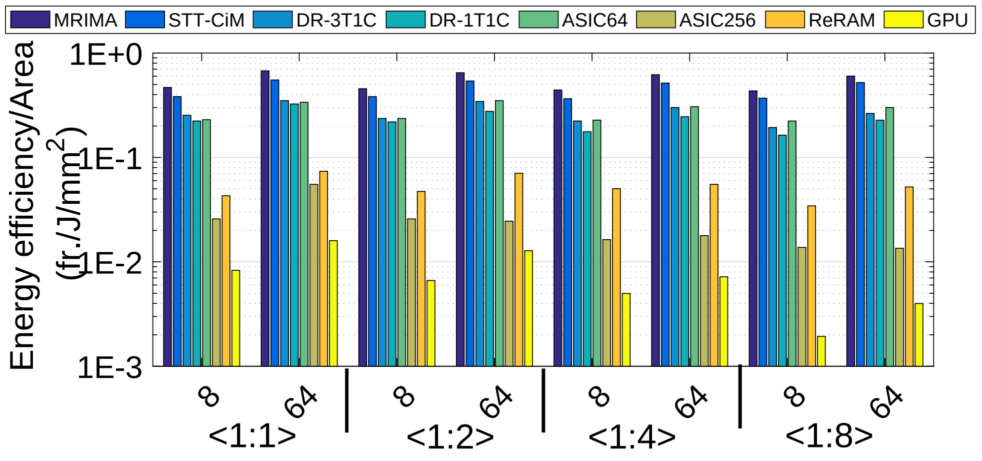


Figure 45: Energy-efficiency of MRIMA vs. different DNN accelerators.

4.3.3.3.5 Performance

Fig. 46 shows the MRIMA's performance results on DNN application in different <W:I>. It shows that the MRIMA solution is 2.4× faster than the best DRAM solution (1T1C-adder) and 11.2× faster than the 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 activation's bit-width is, the higher performance is obtained for MRIMA solution than DRAMs due to its more paralleled computations. Additionally, we see that MRIMA is 8.1× faster than ReRAM solution. ReRAM design employs matrix splitting due to intrinsically limited bit levels of ReRAM device, so multiple sub-arrays should be occupied, besides ReRAM-based cross-bar has a large peripheral circuit's overhead such as buffers and DAC/ADC which contribute more than 85% of area [14]. MRIMA achieves on average 1.5× higher speed-up compared with STT-CiM, with the exact

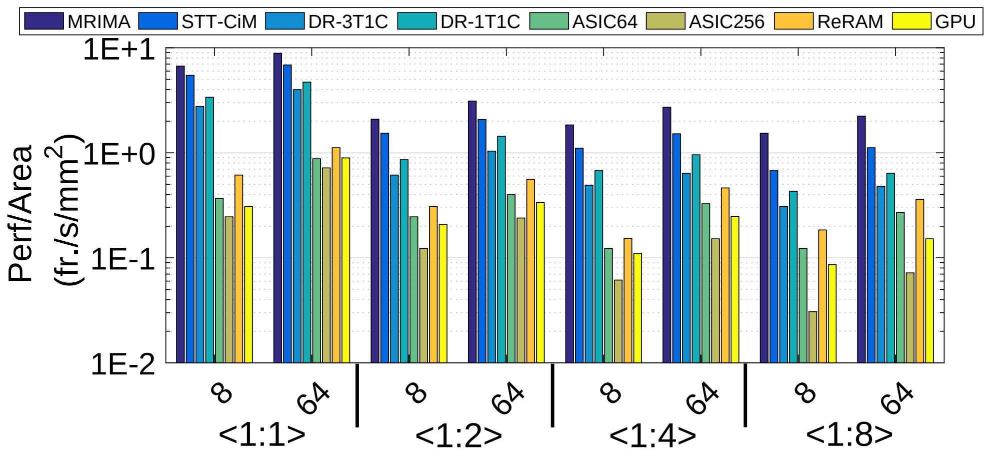


Figure 46: Performance of different MRIMA vs. DNN accelerators.

same memory configuration. This mainly comes from MRIMA's fast and fully parallel operations.

Fig. 47 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 reduced energy and delay. Note that, while the other PIM counterpart designs based on NVMs such as Prime [14], ISAAC [34], etc. implement full bit-wise DNN 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 our experiment, i.e., bit-wise DNN 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.

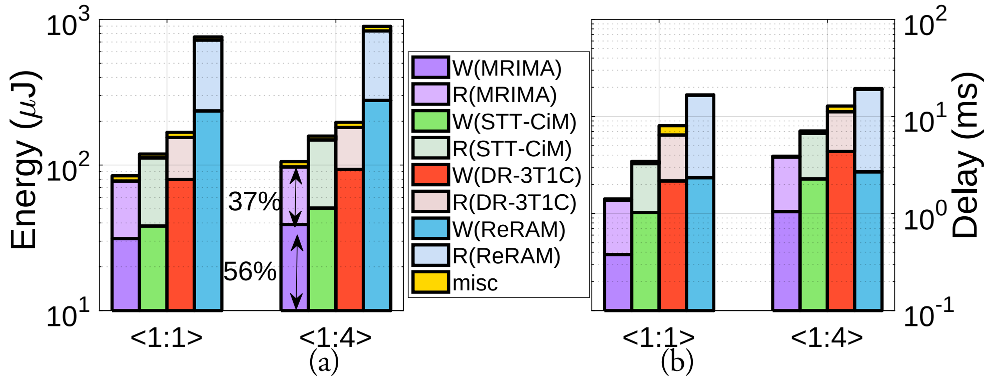


Figure 47: Break-down of (a) Energy and (b) Delay of four PIM platforms.

4.3.3.3.6 Memory wall

Fig. 48 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 performed according to the peak performance and experimentally extracted results for each platform, considering the number of memory access in each bit-width configuration. The results[8] show the MRIMA's favorable solution for solving the 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), the ReRAM solution shows a lower memory bottleneck ratio than MRIMA. This comes from two sources: (1) increased number of computational cycles and (2) unbalanced computation and data movement of MRIMA due to the limitation in the number of activated sub-arrays when operands get larger.

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

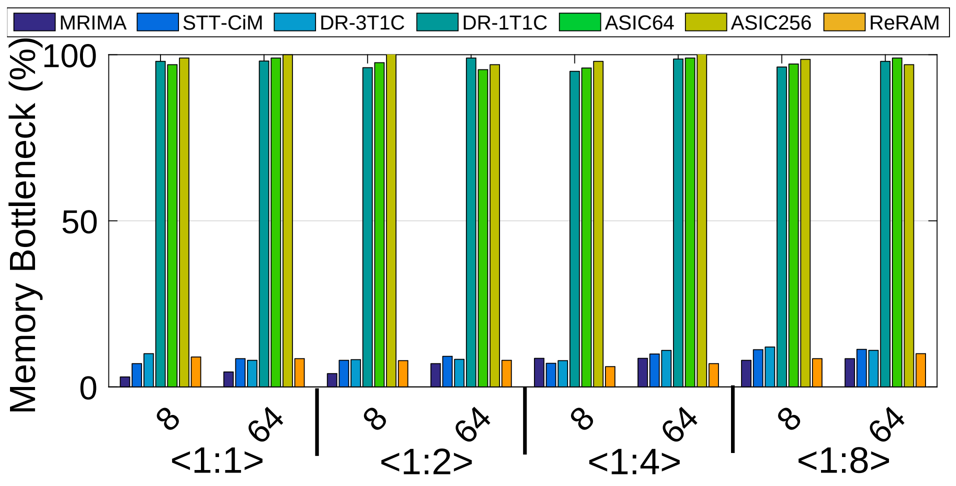


Figure 48: The memory bottleneck ratio.

4.3.3.3.7 Resource utilization

The less memory wall ratio can be interpreted as the higher resource utilization ratio for the accelerators, shown in Fig. 49. 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. 48.

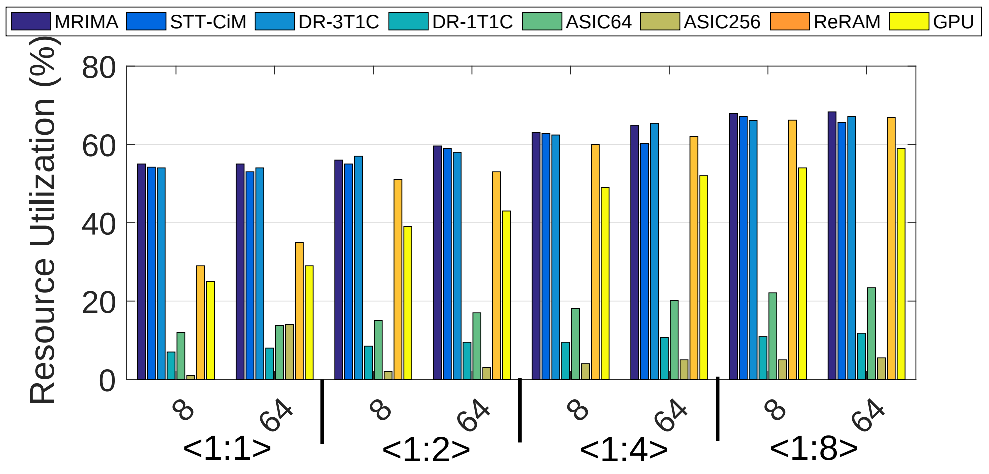


Figure 49: The resource utilization ratio.

4.4 Summary

To accelerate DNN in PIM platforms, the analog current-mode weighted summation operation in resistive memory crossbars intrinsically matches the dominant MAC operation in the DNN. Alternatively, latest algorithmic progression has brought competitive classification accuracy for neural networks despite constraining the network parameters to limited-bit representations, which essentially converts the MAC operation to much simpler bulk bit-wise operations such as addition or comparison that can be accelerated inside existing digital memories (e.g., SRAM, DRAM, MRAM). In this chapter, a generic and comprehensive evaluation framework was initially presented to quantitatively analyze the performance of various PIM platforms running big data applications. The framework was then put into the test to quantitatively compare such analog and digital PIM acceleration solutions for DNNs. The observations was reported considering three key evaluation metrics, i.e., area, energy, latency. In the second part of Chapter 4, a practical DNN case study was presented to demonstrate MRIMA's [1] acceleration for binary-weight and low bit-width convolutional neural networks. The 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.

Chapter 5

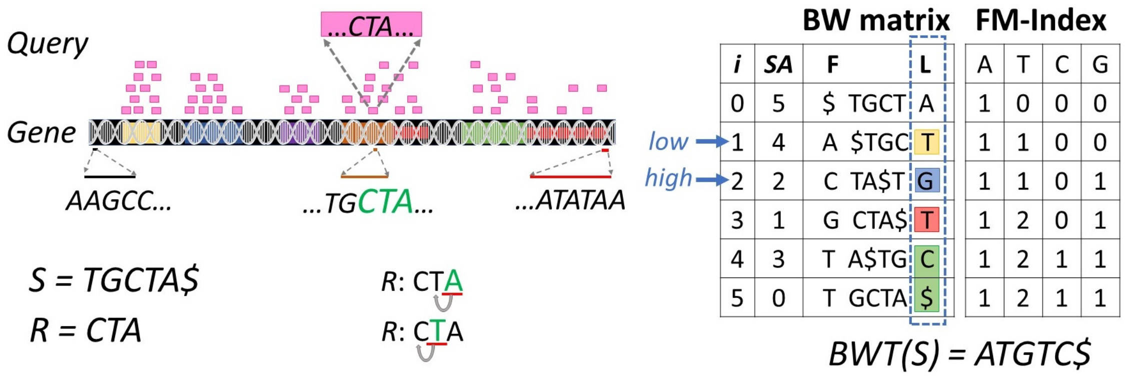
PROCESSING-IN-MEMORY ACCELERATION OF GENOME ANALYSIS

5.1 Introduction

The main focus of chapter 5 is on in-memory acceleration of bioinformatics applications including DNA short read alignment and DNA genome assembly with the presented PIM platforms [2], [10] in Chapter 2. To achieve this goal, the existing bioinformatics algorithms are first reconstructed such that they could be fully implemented in the presented PIM platforms. Then, local data partitioning methodologies, mapping, and pipeline techniques are developed to maximize the parallelism in multiple computational sub-arrays while doing a particular genome analysis task. At the end, the performance and energy-efficiency of the proposed PIM accelerators are extensively assessed and compared with recent genome analysis accelerators based on GPU, ASIC, FPGA, processing-in-ReRAM, etc. with the bottom-up evaluation framework presented in Section 4.2.

5.2 DNA Short Read Alignment

The novel DNA sequencing method on top of the recent high-throughput genomic technologies, is able to analyze and give the accurate order of nucleotides (*nt*) along genomes, and to measure cells' molecular activities. Such advances improves diagnostics of disease and different aspects of medical care, such as prenatal testing and tailoring patient treatment [6], [148]. In general, the generated sequence data of one patient



i	SA	F	L	A	T	C	G
0	5	$ TGCT	A	1	0	0	0
1	4	A $TGC	T	1	1	0	0
2	2	C TA$T	G	1	1	0	1
3	1	G CTA$	T	1	2	0	1
4	3	T A$TG	C	1	2	1	1
5	0	T GCTA	$	1	2	1	1

Figure 50: Short read alignment concept.

sample is composed of tens of millions short DNA sequences (short reads) ranging from 50-500 nucleotide-*nt* in length with no position information. Thus, it is required to determine what part of the chromosome/genome they are from before most genomic analyses can start. This is achieved by aligning the short reads to the reference genome as shown in Fig. 50. The reference genome is really large containing two paired, twisting strands where each strand consists of roughly 3.2 billion *nt* bases (A, T, C, G) in human specifically paired as A-T and C-G [2], [149], [150]. As a result, for a single sample, the DNA short read alignment task is to map the reads (tens of millions) to a reference genome (3.2 billion base pair-*bp*) allowing 1-2 mismatches on each short read. Various alignment algorithms have been developed during the last decade. However, even the efficient algorithms such as Bowtie [151] or BWA [152] based on Burrows-Wheeler Transformation (BWT) seek hours or even days to align the short reads generated by one run (Terabytes of DNA sequence data) from DNA sequencing machine. Therefore, the genomic information achieved from DNA sequencing data cannot be applied for prognosis or disease diagnosis in clinics and hospitals.

Today's sequencing acceleration platforms including CPU, GPU [153], ASIC [148], [154], [155], and FPGA [156] are mostly based on the von-Neumann architecture with separate computing and memory components connecting via buses and inevitably consume a large amount of energy in data movement between them. The most recent ReRAM-based PIM solutions for short read alignment [157], [158] rely on Ternary Content-Addressable Memory (TCAM) arrays that unavoidably impose significant area and energy overheads to the system [149] due to associative processing dealing with Smith-Waterman (SW)-based algorithms that require many write operations and takes 75% of the ReRAM cells to store the intermediate data [159]. Alternatively, RADAR [158], AligneR [149], and FindeR [160] present ReRAM-based PIM architectures that can directly map more efficient algorithms such as BLASTN and FM-index-based search.

5.2.1 BWT-based Read Mapping

Sequence alignment algorithms (e.g., BWA [152] and Bowtie [151]) take all the advantages of BWT and index the large reference genome-S to implement the read alignment efficiently. The BWT is a reversible rearrangement of a character string. Exact alignment finds all occurrences of the short read R (m bp) in the reference genome-S (n bp). Fig. 50 shows an intuitive example of exact alignment of a sample read-$R = CTA$ to a sample reference $S = TGCTA\$$ extracted from a gene, in which \$ denotes the end of a sequence. BW matrix (constructed by lexicographically sorting the strings originated from circulating string S) makes the Suffix Array (S_A) of a reference genome-S a lexicographically-sorted array of the suffixes of S, where each suffix is represented by its position in S. In this way, the last column in the BW matrix is the BWT of reference-S, here, $BWT(S) = ATGTC\$$.

The FM-Index is then built on top of BWT providing the occurrence information of each symbol in it. The S_A interval (low, $high$) covers a range of indices where the suffixes share the same prefix. A backward search of the matched positions in the reference genome-S is then executed for each short read-R starting from the rightmost nucleotide (A in Fig. 50). The matched lower bound (low) and upper bound ($high$) in a S_A of the S for each nucleotide in R are determined based on FM-Index and count function [152]. Thus, S_A interval can represent all the occurrences of the query string. At the end of search, if $low < high$, R has found a match in S. Otherwise, it has failed to find a match. The complexity of this alignment algorithm is linearly proportional to the number of nucleotides in a read ($\mathcal{O}(m)$) in contrast to dynamic programming algorithms such as Smith-Waterman (SW) with $\mathcal{O}(nm)$ complexity [161]. Backtracking can simply extend the exact alignment algorithm to allow mismatches to support inexact alignment. In this approach, the DNA short read is permuted using edit operations (substitutions, insertions or deletions).

5.2.2 Presented PIM Sequencing Algorithm

The DNA alignment-in-memory algorithm consists of two stages: exact alignment and inexact alignment. For most sequencing data, up to ~70% of short reads should be exactly aligned to the reference genome after stage one. The remaining reads are then processed through the stage two. Most genome variations are relatively small, involving only one or two nucleotides. If we only allow exact match between short reads and the reference genome, the reads contain the genome variations from the sample cannot map to the reference genome. In addition, the genome variations (e.g., single nucleotide mutations) cannot be identified based on the exact alignment algorithm. Thus, such potential molecular signatures cannot be applied for disease phenotype prediction. In the following, these two stages are respectively elaborated.

5.2.2.1 Exact Alignment Algorithm

The proposed customized alignment algorithm [2], [10] is developed based on BWT and FM-Index [152], and optimized using the developed PIM's functions (AlignS [2] and PIM-AligneR[9] [10] in Chapter 2). As depicted in Fig. 51, the first step is to store some important pre-computed tables based on reference genome-S. This is only a single-time computation for BWT, S_A, and Marker Table (M_T) to be saved in the memory consuming $\sim$12GB of memory space. We need to reconstruct and store the table data into different memory sub-arrays, banks and chips to provide high-speed memory access and parallel PIM operations. In Fig. 51, the number of nucleotides in the BW matrix's first column that are lexicographically smaller than the nucleotide-nt is represented by $Count(nt)$. So there are four elements for sequence alignment. The FM-index table so-called Occurrence (Occ.) table, is then made based on BWT. In Occ. table, each element-$Occ[i, nt]$ represents the number of occurrences in the position range 0 to $i-1$ for nucleotide-nt in the BWT. We sampled such large Occ. table every d positions (i.e. bucket width) and reconstructed a Sampled Occ. table. Therefore, we are able to diminish the table size by a factor of d. Besides, a M_T was developed by element-wise addition between $Count(nt)$ and sampled Occ. table. Such marker table has the same size with Sampled Occ-Table. M_T basically consists of the matched position of nt in the first column of BWT. Accordingly, both PIM platforms are able to effectively retrieve low and $high$ values in each iteration.

[9] The design is similar to GraphS [4] in Chapter 2.

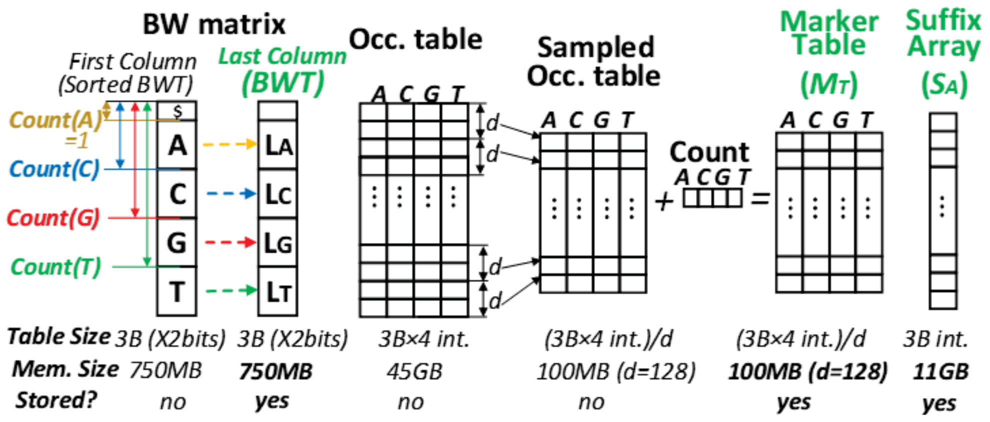

BW matrix
First Column (Sorted BWT)
Last Column (BWT)
Occ. table
Sampled Occ. table
Marker Table (M_T)
Suffix Array (S_A)
Count(A) =1
Count(C)
Count(G)
Count(T)
$
A
C
G
T
LA
LC
LG
LT
A C G T
d
Count
A C G T
Table Size 3B (X2bits) 3B (X2bits) 3B×4 int. (3B×4 int.)/d (3B×4 int.)/d 3B int.
Mem. Size 750MB 750MB 45GB 100MB (d=128) 100MB (d=128) 11GB
Stored? no yes no no yes yes

Figure 51: The required pre-computation in alignment-in-memory algorithm.

In Algorithm 1, the backward search process can be reconstructed with the presented hardware-friendly $Bound(M_T, nt, id)$ procedure (line-9) executed on BWT. This procedure calculates the updated low or $high$ interval's value from M_T with an input index-id considering a bucket width of d. As can be observed, such $Bound$ procedure iteratively performs in every step of 'for' loop. The aforementioned PIM platforms are particularly developed to run such intensive computation via computing two operations, i.e., comparison and addition between the occurrence counting data and 'marker' value for the required nts located between checkpoint positions and remaining positions in BWT. As indicated in Algorithm 1, three in-memory functions, named, MEM (memory read operation), $XNOR\_Match$ (XNOR2), and IM_ADD (add) are used to implement the $Bound$ procedure completely within memory. MEM function is used to access data in the stored S_A or M_T having the index. $XNOR\_Match$ conducts in-memory XNOR2 operation to check if there is a match between the BWT elements saved in the entire word-line and the current input-nt in a single computational cycle. IM_ADD conducts 32-bit integer in-memory addition operation (index range) and compute 'marker+count_match' results without sending it to CPU or other computing units.

Algorithm 1 DNA Exact Alignment-in-Memory.

```
Require: : Pre-Compute and Data Mapping: Partition pre-computed BWT, Marker
    Table ($M_T$) and Suffix Array ($S_A$) into memory chip.
    input: DNA Short Read-$R$
    output: positions of short read-$R$ in reference genome-$S$
    Step-1. Initialization:
 1: $low \leftarrow 0$, $high \leftarrow |S| - 1$
    Step-2. Backward Search:
 2: for $i := |R| - 1$ to $0$ do
 3:     $low \leftarrow \textbf{\textit{Bound}}(M_T[\lfloor low/d \rfloor], R[i], low)$
 4:     $high \leftarrow \textbf{\textit{Bound}}(M_T[\lfloor high/d \rfloor], R[i], high)$
 5:     if $low \geq high$ then
 6:         break & return 0                    ▷ there is no exact alignment
    Step-3. Get matched positions from stored suffix array based on search result:
 7: for $j := low$ to $high - 1$ do
 8:     $positions \leftarrow \textbf{MEM}(S_A[j])$       ▷ Read positions from Suffix Array memory
    Define procedure Bound:
 9: Procedure: $\textbf{\textit{Bound}}(M_T, nt, id)$             ▷ compute matched interval
10: $count\_match \leftarrow 0$
11: for $j := 0$ to $j < (id \bmod d)$ do     ▷ count number of $nt$ within the BWT region
12:     if $\textbf{XNOR\_Match}(nt, BWT[id - (id \bmod d) + j]) == 1$ then
13:         $count\_match = count\_match + 1$
14: $marker \leftarrow \textbf{MEM}(M_T[\lfloor id/d \rfloor], nt])$          ▷ Read Marker Table value
15: return $\textbf{IM-ADD}(marker, count\_match)$
16: end Procedure
```

Two main features can be identified in the modified alignment algorithm that makes it a potential candidate for in-memory implementation: 1) it matches AlignS [2] and PIM-Aligner's [10] logic operations (e.g., comparison and addition) very well and 2) it is memory-bound and parallelizable, not needing any memory access to perform entire read alignment.

5.2.2.2 Extend to Inexact Match

Here the exact alignment algorithm is extended to handle inexact match (mismatch, insertion and deletion) as shown in Algorithm 2. With recursively computing the intervals that match $R[0, i]$, the presented inexact alignment algorithm allows mismatches between read-R and reference genome-S within a tolerance (no more than z differences) with the condition that $R[i+1]$ matches $\{low,\ high\}$. While updating the intervals I, we consider all possible alignments as long as there exists tolerance for differences up to current position i. For the intervals I of position i, we perform union for all match (line 18) and mismatch (line 20) intervals. Accordingly, the algorithm reports the target positions (line 4) in the reference genome, with no more than z mismatches, which the read can be mapped to. We observe that since Algorithm 2 again iteratively exploits the presented *Bound* function, it can be also accelerated by the proposed PIM platforms.

Algorithm 2 DNA Inexact Alignment-in-Memory.

```
Require: : Pre-Compute and Data Mapping: Partition pre-computed BWT, Marker
    Table ($M_T$) and Suffix Array ($S_A$) into memory chip.
    input: DNA Short Read-$R$, $z$ mismatches allowed in the alignment.
    output: positions of short read-$R$ in reference genome-$S$ with up to $z$ mismatches.
    Step-1. Initialization:
 1: $low \leftarrow 0$, $high \leftarrow |S| - 1$
 2: return I= InexactRecursive($R, |R|, low, high, z$):
 3: for $i := |I| - 1$ to 0 do
 4:     $positions \leftarrow$ MEM($S_A[I[i]]$)
    Define procedure InexactRecursive :
 5: Procedure: InexactRecursive($R, i, low, high, z$)        ▷ $z$ is the number of
    mismatches allowed
 6: if $z < 0$ then
 7:     break & return 0
 8: if $i < 0$ then
 9:     return [low,high]
10: $I \leftarrow 0$
11: $I \leftarrow I \cup InexactRecursive(R, i-1, low, high, z-1)$        ▷ Insertion
12: for each $b \in \{A, C, G, T\}$ do
13:     $low \leftarrow$ Bound($M_T[\lfloor low/d \rfloor], R[i], low$)
14:     $high \leftarrow$ Bound($M_T[\lfloor high/d \rfloor], R[i], high$)
15:     if $low < high$ then
16:         $I \leftarrow I \cup InexactRecursive(R, i, low, high, z-1)$        ▷ Deletion
17:         if $b = R[i]$ then
18:             $I \cup InexactRecursive(R, i-1, low, high, z)$        ▷ Exact Match
19:         else
20:             $I \cup InexactRecursive(R, i-1, low, high, z-1)$        ▷ Inexact Match
21: return $I$
22: end Procedure
```

5.2.3 Correlated and Localized Computation

5.2.3.1 Partitioning

The pre-computed tables (BWT, M_T, and S_A) require a large memory space, therefore, to fully leverage AlignS and PIM-Aligner's parallelism, and maximize alignment throughput, a partitioning, mapping and pipeline design was designed. Given a BWT index range, the accessed memory region of M_T and BWT could be readily predicted and computation could be localized if we store such correlated region into the same memory sub-array. The correlated data partitioning and mapping methodology, as shown in Fig. 52a, locally stores correlated regions of BWT and M_T vectors in the same memory sub-array to enable fully local computation (i.e., *XNOR_Match* and *IM_ADD* completely within the same sub-array without inter-bank/chip communication). To do it, each PIM's sub-array (512×256) is spilt into four zones to save four different data types, i.e., BWT, CRef, M_T, and reserved space for *IM_ADD* (Fig. 52a). First, 256 rows are filled with the corresponding BWT, where each row stores up to 128 bps (encoded by 2 bits). Besides, 4 nucleotide computational reference vectors (*CRef*) are initialized, in which each vector gives one type of nucleotide with vector size of number of bits in one word-line. CRef is designed to enable fully parallel match operation- *XNOR_Match*. Next to it, the value of markers (M_T) is check-pointed every d (=128) positions (one row), and vertically stored to keep the size in check within the PIM platforms. Hence, 256 columns are allocated for storing M_T, each storing 4-byte value for bps (128-bit). After partitioning, starting from the rightmost symbol in R, *Bound* procedure runs and returns *low* and *high* for next symbol.

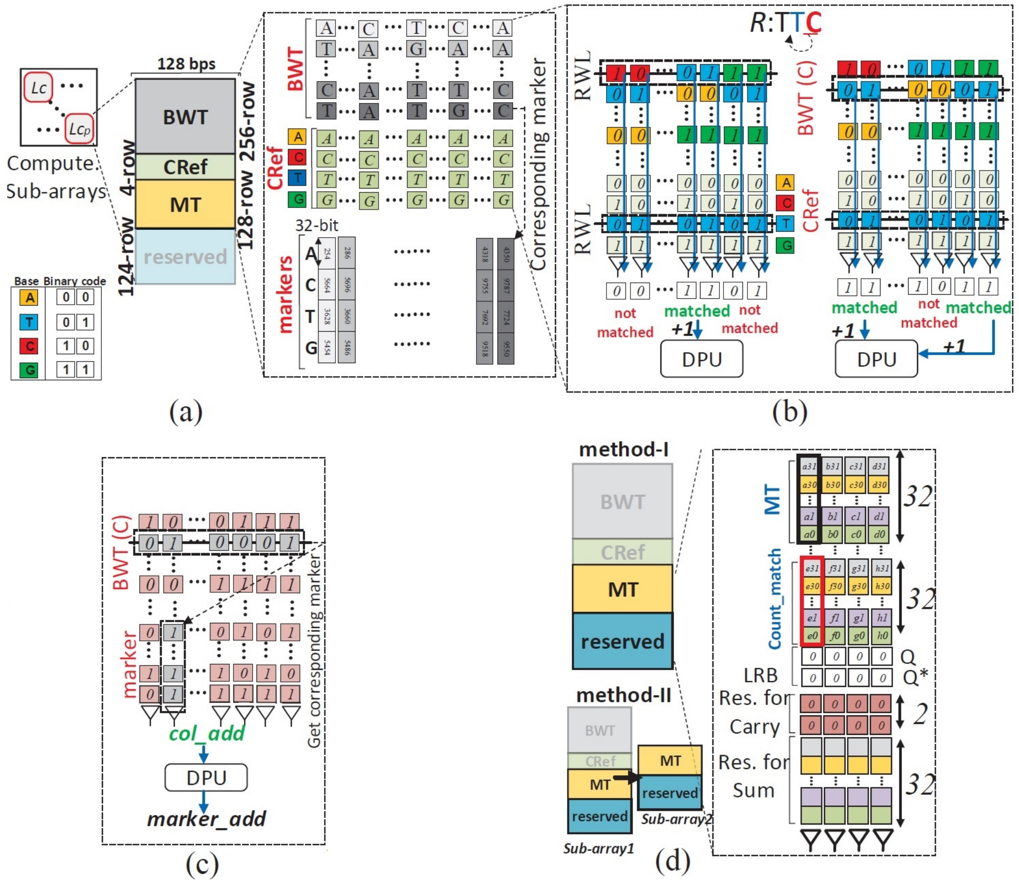


Figure 52: (a) AlignS [2] and PIM-Aligner's [10] sub-array partitioning for comparison and addition operations, (b) Parallel comparison operation (*XNOR_Match*), (c) *MEM* function to retrieve marker_add, (d) *IM_ADD* function with two methods.

5.2.3.2 Mapping and Computation

Considering current input nucleotide is T and input index as id (in Fig. 52b), the presented PIM platforms can convert the BWT index into the corresponding memory WL and BL addresses storing data $BWT[id - (id \bmod d)]$ to $BWT[id]$. Then, such bits and corresponding CRef-T can implement the parallel comparison operations (*XNOR_Match*). If the XNOR output is '1' (a match is found), DPU's embedded counter counts up to eventually compute $count\_match$ for next operation. Fig. 52b intuitively shows the *XNOR_Match* procedure to locate Ts in a sub-array. When counting is done, the sub-array returns the $count\_match$ and marker address ($marker\_add$), shown in Fig. 52c. The correlated data partitioning methodology guarantees the read of $marker$ value (*MEM*) is always a local memory access within the same memory array (Fig. 52c). Now, the $marker$ and just computed and transposed $count\_match$ are buffered in M_T and reserved memory spaces, respectively, as shown in Fig. 52d. To further implement *IM_ADD* function, two distinct hardware-friendly methods are proposed; method-I performs the bit-line addition within the same computational sub-array based on the presented in-memory addition operation though it degrades the system performance as other sub-array resources (*MEM* and *XNOR_Match*) are not used. To alleviate this issue, method-II essentially duplicates the number of sub-arrays, where only in-memory addition computation is transferred to a second sub-array.

5.2.3.3 Pipeline Design

To improve the base-line PIM-Aligner's performance, the processing of multiple reads is considered such that in each pipeline stage a different short read-*R* could be processed. We take the partitioning method-II for pipeline design. With a careful observation of DNA alignment computation phases, it can be realized that the different computing resources of a single sub-array could be set free by copying the sub-array data into a new sub-array. Therefore, we define P_d as parallelism degree (i.e., # of the leveraged sub-arrays) to control the trade-off of resources and performance metrics. For instance, comparison resources of a particular sub-array can be set free after duplicating (P_d=2) that sub-array (method-II). This pipeline technique is intuitively shown in Fig. 53 for a sample 3 reads; when the $R1$ is being processed for *IM_ADD* in the second

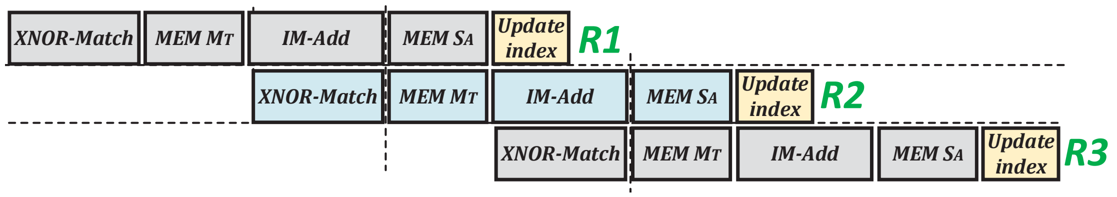


Figure 53: The pipeline technique with P_d=2 for PIM-Aligner.

sub-array, $R2$ can exploits the parallel *XNOR_Match* resources in the first sub-array to increase the parallelism. This can be generalized to more number of sub-arrays where more than two sub-arrays contribute to the computation at the cost of a higher energy consumption.

5.2.4 Evaluation

5.2.4.1 Counterpart Computing Platforms

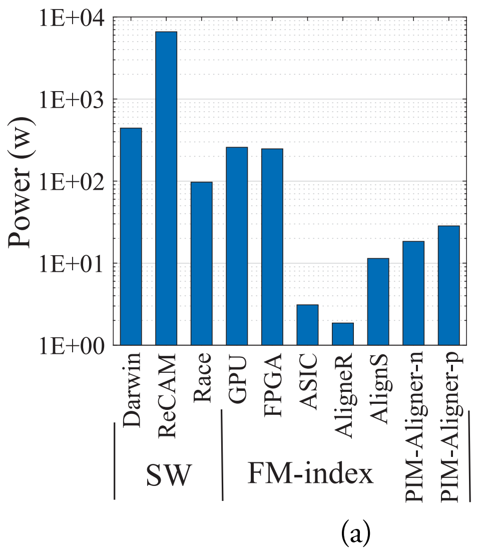
To evaluate the performance of the presented PIM platforms, i.e., accelerating DNA short read alignment task, the comprehensive device-to-architecture evaluation framework in Section 4.2 was exploited. We perform an extensive comparison with the counterpart computing platforms, including SW-based Darwin [148], ReCAM [162] and RaceLogic [154], as well as FM-Index-based platforms including Soap3-dp [153] on GPU, FPGA [156], ASIC [155], AlignS [2], and AligneR [149]. We refer the readership to the abovementioned papers for the detailed configuration of each accelerator. Note that, to perform short read alignment on GPU, the Soap3 [153] was used considering only reads with $\leq$2 mismatches. The ReRAM-, SOT-MRAM, and CAMs were re-implemented with PIMA-SIM. For evaluation, 10 millions 100-bps short read queries were generated via ART simulator [163] and aligned to the human genome

Hg19 with different computing platforms. Note that the population variation and genome error rate were set to 0.1% and 0.2%.

5.2.4.2 Power & Throughput

The power consumption of the DNA alignment task for different accelerators is calculated and shown in Fig. 54a. The AlignS, the baseline (PIM-Aligner-n) and the pipe-lined PIM-Aligner (P_d=2, PIM-Aligner-p) are accordingly implemented. The first observation is that SW-based platforms (except for RaceLogic [154]) require a larger power-budget as we expected, compared with FM-index-based designs. Besides, among FM-index-based platforms, the PIMs generally show less power consumption. ReRAM-based AligneR [149], ASIC [155], and SOT-MRAM-based AlignS [2] respectively consume the least power. PIM-Aligner-n stands as the fourth power-efficient design. It is noteworthy that PIM-Aligner uses three SAs per bit-line to perform the computation in a single cycle, while the AlignS design has two SAs and a two-cycle addition scheme as discussed in Chapter 2. That is why the PIM-Aligner consumes more power than the SOT-MRAM counterpart.

The throughput results for different platforms are reported in Fig. 54b. We observe that PIM-Aligner-p shows the highest throughput compared with other platforms except RaceLogic due to its massively-parallel and local computational scheme. Based on this plot, the pipeline technique with P_d=2 has improved the performance by $\sim$40% compared to the baseline design.



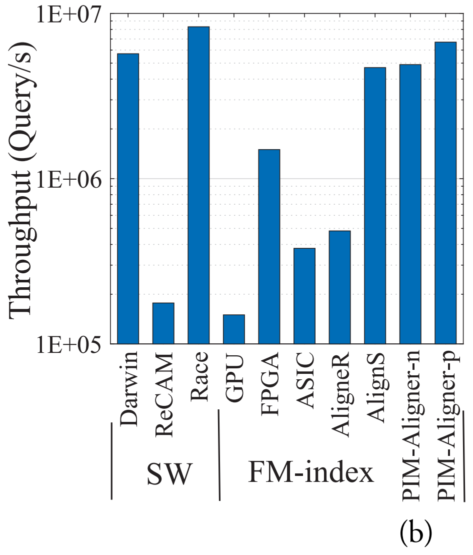


Figure 54: (a) Power consumption and (b) Throughput of different accelerators compared to AlignS and PIM-Aligner (Y-axis:Log scale)

5.2.4.3 Trade-off

The performance/power trade-off can be better explained by correlated parameters, as plotted in Fig. 55a-b. We observe that SOT-MRAM-AlignS achieves the highest throughput per Watt compared to other platforms. Where PIM-Aligner-n stands as the second most efficient design. The PIM-Aligner improves the short read alignment's performance by 3.1× over the RaceLogic [154], the best SW-based accelerator, and ∼2×, 43.8×, 458× over ASIC [155], FPGA [156], and GPU [153] platforms, respectively. Fig. 55b takes estimated area of the chips into account. Considering the area factor, we observe that PIM-Aligner improves read alignment performance significantly over all the other solutions, e.g., by ∼9× and 1.9× compared to FM-index-based ASIC and processing-in-ReRAM designs, respectively. Fig. 55c shows the trade-off between power and throughput w.r.t. parallelism degree. We can see that by increasing the P_d,

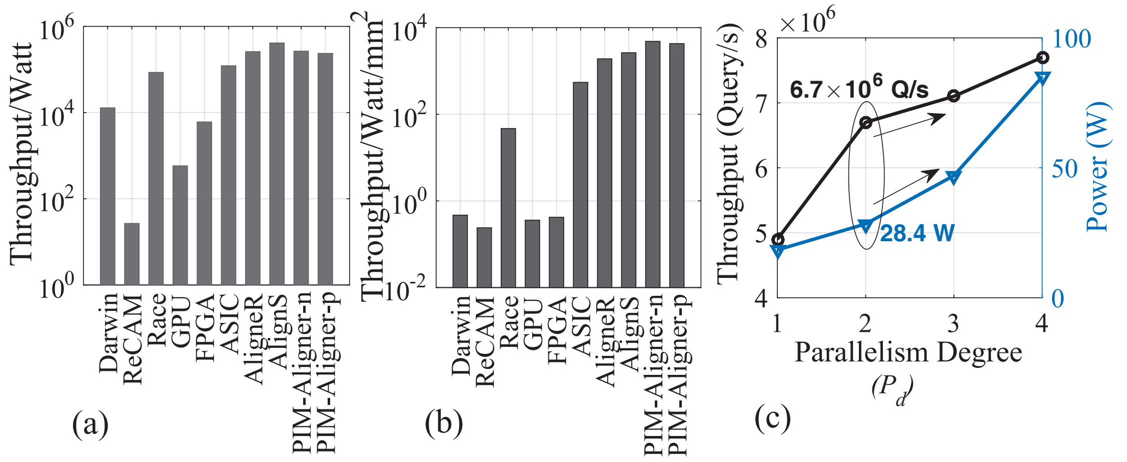


Figure 55: (a) Throughput/Watt, (b) Throughput/Watt/Area, and (c) Power-throughput trade-off w.r.t. P_d.

both power consumption and throughput will increase. Therefore, P_d can be tailored according to the system constraints to provide the best solution.

5.2.4.4 Off-Chip Memory Access

Figure 56a shows the required off-chip memory access for different accelerators. We observe that FM-index-based GPU[153] and FPGA [156] platforms heavily rely on off-chip memory consuming humongous energy to fetch data from stored tables and queries. Note that, ASIC design performs the alignment with only 1GB off-chip memory after compression. Figure 56b reports the Memory Bottleneck Ratio (MBR). Based on this, PIM-Aligner spends less than ∼18% time for memory access and data transfer. It is worth pointing out that other PIM platforms also spend less than 25% time waiting for the loading data. AligneR [149] solution shows higher memory bottleneck ratio than PIM-Aligner due to its unbalanced computation and data movement. The less MBR can be translated as the higher Resource Utilization Ratio (RUR) for the

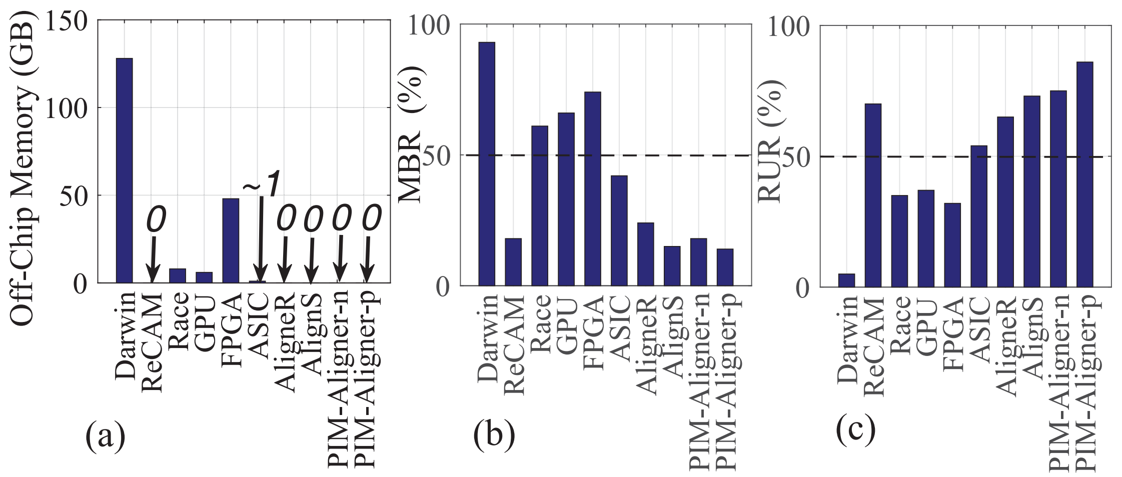


Figure 56: (a) Off-chip memory, (b) Memory Bottleneck Ratio, (c) Resource Utilization Ratio for different accelerators.

computing platforms, shown in Fig. 56c. We can see that PIM-Aligner-p shows the highest resource utilization with up to ∼86%.

5.3 DNA Genome Assembly

With the advent of high-throughput second generation parallel sequencing technologies, the process of generating fast and accurate large-scale genomics data has become a significant advancement. Such data can enable us to measure the molecular activities in cells more accurately by analyzing the genomics activities, including mRNA quantification, genetic variants detection, and differential gene expression analysis. Thus, by understanding the transcriptomic diversity, we can improve phenotype predictions and provide more accurate disease diagnostics [164]. However, the reconstruction of the full-length transcripts considering sequencing errors is a challenging task in terms of computation and time. Since the current cDNA sequencing technology cannot read whole genomes in one step [165], the data produced by the sequencer

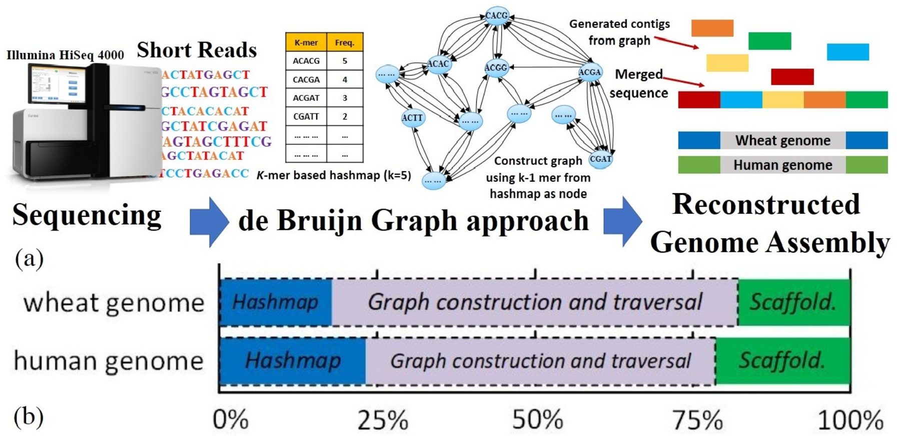


Figure 57: (a) The de Bruijn graph-based genome assembly process, (b) Break down of execution time of Meraculous genome assembler for human and wheat data-set [165], [166].

is extensively fragmented due to the presence of repeated chunks of sequences, duplicated reads, and large gaps. Thus, the goal of genome assembly process is to combine these large number of fragmented short reads and merge them into long contiguous pieces of sequence (i.e., contigs), to reconstruct the original chromosome from which the DNA is originated as shown in Fig. 57a.

In bioinformatics hardware acceleration domain, most CPU [167]-/ GPU [168]-/ FPGA [169]- and even PIM [2], [149]-based efforts have only focused on the DNA short read alignment problem, while the de novo genome assembly problem still relies mostly on CPU-based solutions [170]. De novo assemblers are categorized into Overlap Layout Consensus (OLC), greedy, and de Bruijn graph-based designs. Recently, de Bruijn graph-based assemblers have gained much more attention as they are able to solve the problem using Euler path in a polynomial time rather than finding Hamiltonian path in OLC-based assemblers as an NP hard problem [171]. There are multiple CPU-based genome assemblers implementing the bi-directed de Bruijn graph model, such as Velvet [172], Trinity [173], etc. However, only a few GPU-accelerated assemblers have been presented such as GPU-Euler [170], [174], [175]. This mainly comes from the nature of the assembly workload that is not only compute-intensive but also extremely data-intensive requiring very large working memories. Therefore adapting such problem to use GPUs with their limited memory capacities has brought many challenges [176]. A graph-based genome assembly process, shown in Fig. 57a, basically consists of multiple stages, i.e., *k*-mer analysis for creating a Hashmap, graph construction and traversal, and scaffolding and gap closing. Fig. 57b depicts the breakdown of execution time for the well-known Meraculous assembler [166] for the human and wheat data sets. We observe that Hashmap and graph construction/ traversal are the two most expensive components, which together take over 80% of the total run time.

5.3.1 Presented PIM Assembly Algorithm and Mapping

The genome assembly algorithm consists of three main stages visualized in Fig. 58. First, creating a hash table out of chopped short reads (*k*-mers) and keeping a count of each distinct *k*-mer; second, constructing a de Bruijn Graph with Hashmap; third, traversing through de Bruijn Graph for Euler Path[10]. There is a final stage called scaffolding to close the gaps between contigs, which is the result of the denovo assembly [165]. The first three stages always take most fraction of execute time and computational resources (over 80%) in both CPU and GPU implementations [165]. To effectively handle the huge number of short reads, the assembly algorithm was modularized by focusing on parallelizing the main steps by loading only the necessary data at each stage into the PIM platform, and leave stage-4 as our future work. A variation of

[10]The stage II and III are so-called contig. generation

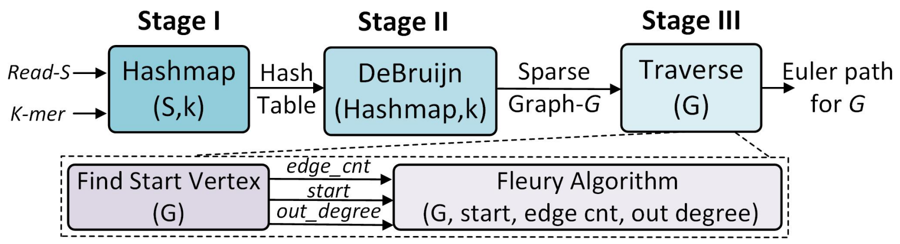


Figure 58: The genome assembly stages.

GraphS [4] PIM platform, called PANDA [51] developed on top SOT-MRAM is selected in this section for implementation. The PANDA can execute not only efficient bulk bit-wise X(N)OR-based comparison/addition operations heavily required for the genome assembly task but a full-set of 2-/3-input logic operations inside the MRAM chip.

5.3.1.1 Stage One: Hash Table

Algorithm 3 shows the reconstructed *Hashmap(S,k)* procedure in which the algorithm takes *k*-mer from the original sequence (*S*) in each iteration, creates a hash table entry (key) for that, and assigns its frequency (value) to 1. This step is visualized in Fig. 59. If the *k*-mer is already in the table, it will calculate a new frequency (New_frq) by adding the previous frequency by one and update the value. As indicated, Hashmap procedure can be implemented through *PANDA_Cmp* (comparison), *PANDA_Add* (addition), and *PANDA_Mem_insert* (memory W/R) in-memory operations. Such functions are iteratively used in every step of 'for' loop and PANDA is specially designed to handle such computation-intensive load through performing comparison, summing, and copying operations.

Algorithm 3 Procedure Hashmap(S, k)

```
   Step-1. Initialization:
1: hashtable named Hashmap = {}
   Step-2. Fill out the table:
2: for $i$ := 0 to length(S)-k+1 do
3:     $k\_mer \leftarrow S[i : i + k]$        ▷ copy values of $S[i$ to $i + k]$ into variable $k\_mer$
4:     if PANDA_Cmp($k\_mer, Hashmap$) == 0 then
5:         PANDA_Mem_insert($k\_mer, 1$)
6:     else
7:         $New\_frq \leftarrow$ PANDA_Add($k\_mer, 1$)        ▷ increment frq by 1
8:         PANDA_Mem_insert($k\_mer, New\_frq$)        ▷ insert into Hashmap again
9: return Hashmap
```

Considering the fact that the number of different keys in Hash table is almost comparable to the genome size G, the memory space requirement to save the hash is given by $\sim 2 \times G \times (k + 1)$ bits (The factor of 2 is given to represent 2 bits per nucleotide). For instance, storing Hash table for human genome with $G \sim 3\times10^9$ and k=32 requires ∼23GB mostly associated with storing the key. Due to very large memory space requirement of hash table for assembly-in-memory algorithm [165], we partition these tables into multiple sub-arrays to fully leverage PANDA's parallelism, and to maximize computation throughput. Obviously, larger memory units [177] and distributed memory schemes [165], [178] are preferable.

The proposed correlated partitioning and mapping methodology, as shown in Fig. 60a, locally stores correlated regions of *k*-mer (980 rows) vectors, where each row stores up to 128 bps (*A,C,G,T* encoded by 2 bits) and value (32 rows) vectors in the same sub-array. For counting the frequencies of each distinct *k*-mer, the ctrl first reads and parses the short reads from the original sequence bank to the specific sub-array. As depicted in Fig. 60a, assuming *S=CGTGTGCA* as the short read, the *k*-mers- k_i-k_{i+n} are extracted and written into the consecutive memory rows of *k*-mer region. However, when a new query such as k_{i+3} arrives (while k_i-k_{i+2} are already in the memory), it will be first written to the temp region. A parallel in-memory comparison operation

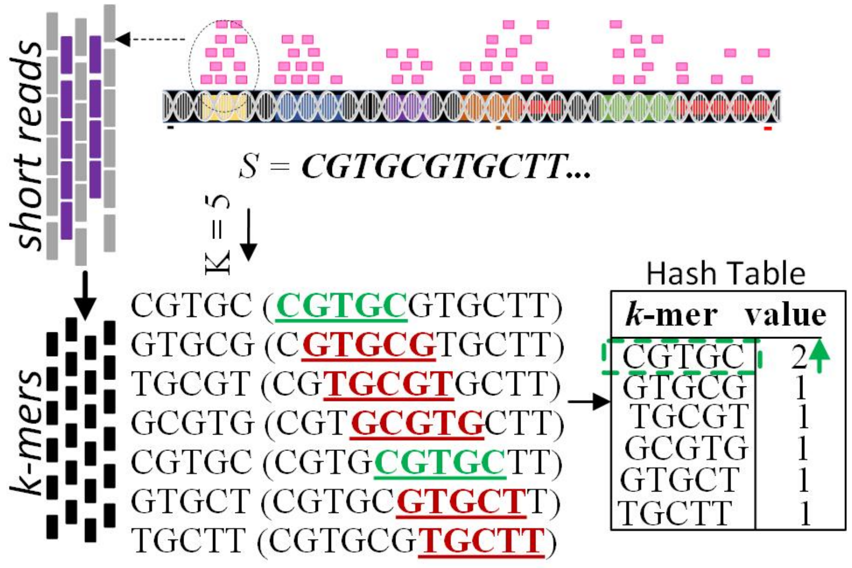


Figure 59: The hash table generation out of k-mers.

(*PANDA_Cmp*) will be performed between temp data and already-stored *k*-mers. Fig. 60b intuitively shows *PANDA_Cmp* procedure, where entire temp row can be compared with a previous *k*-mer row in a single cycle. Then, a built-in ctrl's AND unit in DPU readily takes all the results to determine the next memory operation according to the algorithm. To increase the frequency of a specific *k*-mer, *PANDA_Add* is leveraged to perform in-memory addition without sending data to off-chip processor.

5.3.1.2 Stage Two: Graph Construction

The next step is to construct and access a de Bruijn graph based on the Hash structure to rapidly lookup of a 'value' associated with each *k*-mer. For each entry (of length k) in the Hashmap, we will make two nodes, one with the prefix of length k-1 and other with the suffix of length k-1 (e.g., CGTGC $\rightarrow$ CGTG and GTGC), and connect an edge between them. For each Hash table entry with n as the frequency, n edges is then

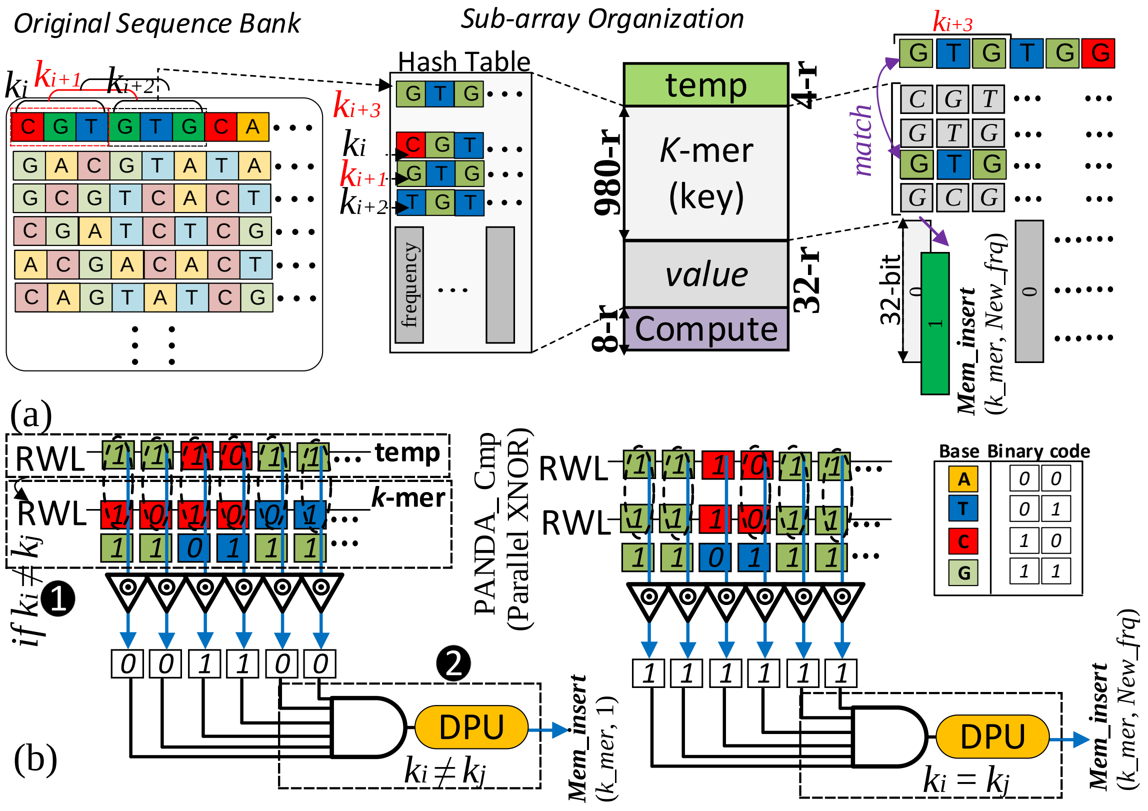


Figure 60: (a) The proposed correlated data partitioning and mapping methodology for creating hash table, (b) Realization of parallel in-memory comparator (*PANDA_Cmp*) between *k*-mers in a computational sub-array.

added between the two nodes. The de Bruijn graph *G* for the sample Hash table in Fig. 59 is constructed in Fig. 61 (step 1). Algorithm 4 shows the reconstructed de Bruijn procedure for PANDA taking Hashmap data and *k* as input returning matrix *G*. For each key within Hash table, *PANDA_Mem_insert* instruction creates an entry in *G* for node1 and node2s.

Leveraging adjacency matrix representation for direct mapping of such humongous sparse graph into memory comes at a cost of significantly increased memory requirement and run time. The size of adjacency matrix will be V×V for any graph with V nodes, where sparse matrix could be represented by a 3×E matrix, where E is the total

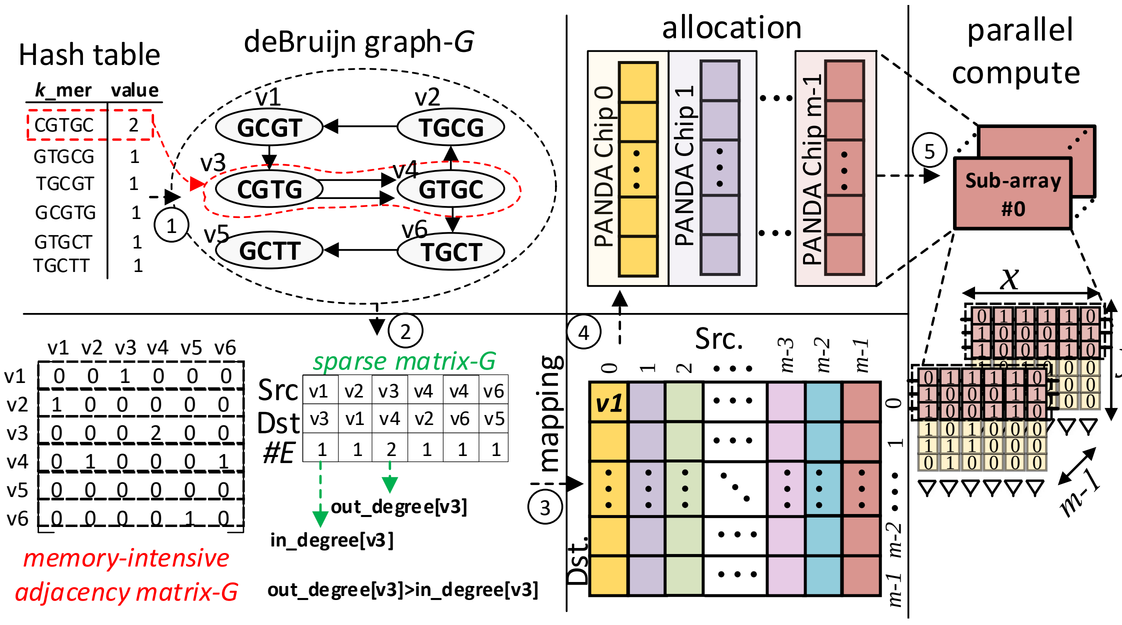


Figure 61: Graph construction with sparse matrix with partitioning, allocation and parallel computation.

number of edges in the graph. PANDA utilizes sparse matrix representation shown in Fig. 61 (step 2) for mapping purpose. Each entry in the 3^{rd} row of the sparse matrix represents the number of connections between two nodes in 1^{st} and 2^{nd} rows.

Algorithm 4 Procedure DeBruijn(Hashmap, k)

```
   Step-1. Initialization:
1: G=[], Nodes_List=[], i=1
   Step-2. Sparse Graph Construction:
2: for $\forall k\_mer \in Hashmap.keys(), i++$ do
3:     $node\_1 \leftarrow k\_mer[0 : k-2]$
4:     $node\_2 \leftarrow k\_mer[1 : k-1]$
5:     $\textbf{PANDA\_Mem\_insert}(G[1][i], node\_1)$
6:     $\textbf{PANDA\_Mem\_insert}(G[2][i], node\_2)$
7:     $\textbf{PANDA\_Mem\_insert}(G[3][i], Hashmap[k\_mer])$
8: return G
```

To balance workloads of each PANDA's chip and maximize parallelism, the interval-block partitioning method was leveraged. We used the hash-based approach [179] by splitting the vertices into M intervals and then divided edges into M^2 blocks as shown Fig. 61 (step 3: mapping). Then each block is allocated to a chip (step 4: allocation) and mapped to its sub-arrays. Having an m-vertex sub-graph with N_s activated sub-arrays (size=$x \times y$), each sub-array can process n vertices ($n \leq f | n \in N, f = min(x, y)$) (step 5: parallel computation). In this way, the number of processing sub-arrays for an N-vertex sub-graph can be formulated as, $N_s = \left\lceil \frac{N}{f} \right\rceil$.

After graph construction, it is possible to perform a round of simplification on the sparse graph stored in PANDA without loss of information to avoid fragmentation of the graph. As a matter of fact, the blocks are broken up each time a short read starts or ends leading to linear connected subgraphs [172]. This fragmentation imposes longer execution time and larger memory space. The simplification process easily merges two nodes within memory if a node-A has only one out-going edge directed to node-B with only one in-going edge.

5.3.1.3 Stage Three: Traversal for Euler Path

The input of this stage will be a sparse representation of graph G. For traversing all the edges, we will use Fleury's algorithm to find the Euler path of that graph (a path which traverses all edges of a graph). Basically, a directed graph has a Euler path if the in_degree and out_degree[11] of every vertex is same or, there are exactly two vertices which have |in_degree - out_degree|= 1. Finding the starting vertex is very important to generate the Eulerian path and we cannot consider any vertex as a starting vertex. The reconstructed PIM-friendly algorithm for finding the start vertex in graph-G is shown in Algorithm 5. For each node, this stage deals with massive number of iteratively-used *PANDA_Add* to calculate the number of in_degree, out_degree and edge_cnt

[11]The in_degree[i] shows how many edges are coming into a vertex-i and out_degree[i] means how many out-going edges vertex-i has.

(total number of edges). Moreover, in order to check the condition (|out_degree = in_degree|+ 1), parallel *PANDA_Cmp* operation is required.

Algorithm 5 Procedure Find Start Vertex(G)

```
    Step-1. Initialization:
 1: $start \leftarrow 0$, $end \leftarrow 0$
 2: $edge\_cnt \leftarrow 0$                        ▷ For counting number of edges in $G$
 3: $Len \leftarrow size(G)$
    Step-2. Find the start vertex:
 4: for $n$ in Nodes do
 5:     $in\_degree[i] \leftarrow 0$
 6:     $out\_degree[i] \leftarrow 0$
 7: for $n$ in Nodes do
 8:     for $k$ :=1 to $Len$ do
 9:         if PANDA_Cmp($G[1][k], n$) then            ▷ node n has an out-going edge
10:             $out\_degree[n] \leftarrow$ PANDA_Add($out\_degree[n], int(G[3][k])$)
11:             $in\_degree[int(G[2][k])] \leftarrow$ PANDA_Add($in\_degree[int(G[2][k])], int(G[3][k])$)
12:             $edge\_cnt \leftarrow$ PANDA_Add($edge\_cnt, int(G[3][k])$)
13:     if PANDA_Cmp($out\_degree[n], in\_degree[n] + 1$) then
14:         $start \leftarrow n$
15:     else
16:         $start \leftarrow first\_node$
17: return start & edge_cnt & out_degree
```

After finding the start node, PANDA has to traverse through the length of sparse matrix G from the starting vertex and check two conditions for each edge and accordingly add qualified edges to the Eulerian Path. The reconstructed Fleury algorithm is shown in Algorithm 6. If an edge *is not a bridge* and *is not the last edge of the graph*, we will add $(start, v)$ in the Eulerian path and remove that edge. $isValidNextEdge()$ function will check if the edge (u, v) is valid to be included into our Euler path. If v is the only adjacent vertex remaining for u, it means that, we have traversed all other adjacent vertices, so we will take this edge, otherwise we won't. The second condition counts the number of reachable nodes from u before and after removing the edge. If the number changes/decreases, it means that, the edge was a bridge (removing it will disconnect the graph into two parts). If it is a bridge, we cannot remove the edge from the graph; otherwise we will remove the edge and add it into Euler path.

Algorithm 6 Procedure Fleury(G, node, edge_count, out_degree)

```
1: for $v := 0$ to $N$ do
2:     if $G[1][k] == start$ then
3:         $v \leftarrow G[2][k]$
4:         if $isValidNextEdge(v)$ then
5:             $\boldsymbol{PANDA\_Mem\_insert}(v)$    ▷ add ($start$, $v$) in the Eulerian path
6:             $\boldsymbol{PANDA\_Add}(out\_degree[start], -1)$
7:             $\boldsymbol{PANDA\_Add}(G[3][k], -1)$    ▷ remove one edge from the graph
8:             $\boldsymbol{PANDA\_Add}(edge\_cnt, -1)$
9:     Fleury(G, v, edge_count, out_degree[])    ▷ run Fleury again for the next node $v$
```

The out_/in_degree and edge_cnt mapping and computation are shown in the PANDA platform in Fig. 62, which basically sums up all the entries of a particular node *i* of valid links connected to a vertex to find the start vertex. As can be seen, the sparse matrix representation is used to store the matrix-*G*. In the proposed mapping technique, each column is assigned to a distinct source vertex in the graph and then filled out with the number of edges (#E) only linked to existing destination vertices in a vertical fashion. Therefore, we do not assign destination vertices to the memory rows as in direct adjacency matrix mapping. Here, a 4-bit representation is considered for the simplicity. For example, v4 has out-going edges to v2 and v6 that are stored vertically in a sub-array. PANDA could perform parallel in-memory addition to calculate the total number of out_ degree for all nodes in parallel. For this task, two rows in the sub-array are initialized to zero as Carry reserved rows such that they can be selected along with two operands (here v4→v2 data (0001) and v4→v6 data (0001)) to perform parallel in-memory addition. To perform parallel addition operation and generate initial Carry and Sum bits, PANDA takes every three rows to perform a parallel in-memory addition. The results are written back to the memory reserved space (Resv.). Then, next step only deals with multi-bit addition of resultant data starting bit-by-bit from the LSBs of the two words continuing towards MSBs. Then PANDA is able to perform comparison between number of out_degree and in_degree for each node in parallel to determine the start node. After finding the start node as shown in

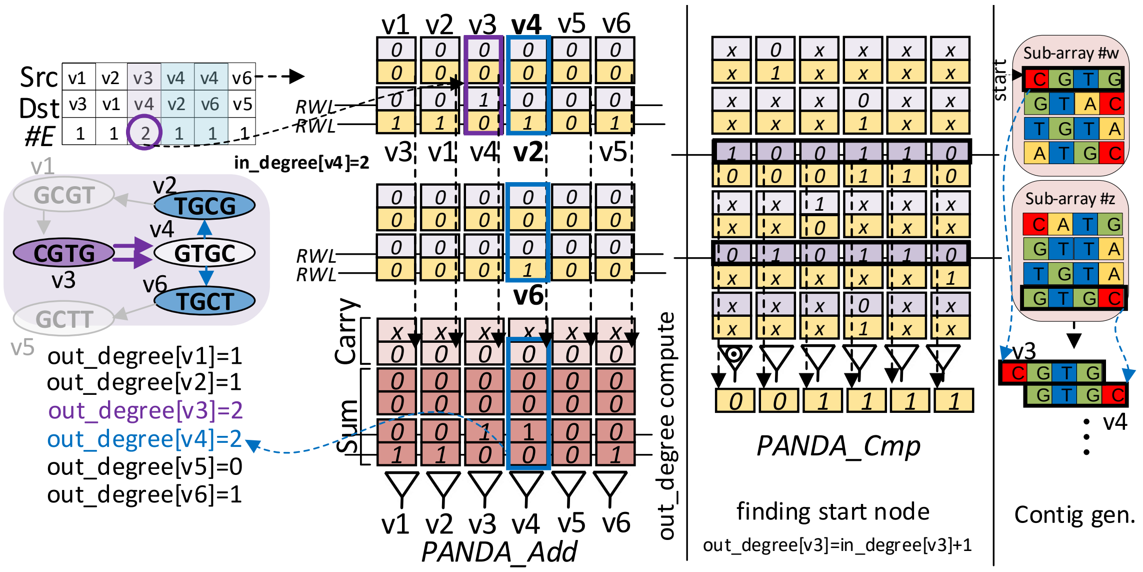


Figure 62: PANDA in-memory addition and comparison scheme for finding the start vertex.

Fig. 62, contig. generation can be readily accomplished through finding the Eulerian path and putting together each vertex data from different sub-arrays.

5.3.2 Evaluation

5.3.2.1 Counterpart Computing Platforms

To the best of our knowledge, PANDA is the first to explore the performance of a PIM platform for genome assembly problem, therefore, the evaluation test bed was developed from scratch to have an impartial comparison with both von-Neumann and non-von-Neumann architectures. The PANDA's computational memory sub-array was configured with 1024 rows and 256 columns, 4×4 memory matrix (with 1/1 as row/column activation) per bank organized in H-tree routing manner, 16×16 banks (with 1/1 as row/column activation) in each memory chip. For comparison, we consider five computing platforms: 1) A general purpose processor (GPP): a Quad Core Intel Core i7-7700 CPU @ 3.60GHz processor with 8192MB DIMM DDR4 1600MHz RAM and 8192KB Cache; 2) A processing-in-STT-MRAM platform capable of performing bulk bit-wise operations [180]; 3) The developed processing-in-SOT-MRAM platform, i.e., AlignS for DNA sequence alignment optimized to perform comparison-intensive operations [2]; 4) A processing-in-ReRAM accelerator designed for accelerating bulk bit-wise operations [37]; 5) A processing-in-DRAM accelerator based on Ambit [19] working with triple row activation mechanism to implement various functions. All PIM platforms have an identical physical memory configuration as PANDA.

The presented cross-layer simulation framework in Section 4.2 was then used starting from device-level simulation all the way to circuit- and architectural level. To evaluate the CPU performance, we used Trinity-v2.8.5 [173] which was shown to be sensitive and efficient in recovering full-length transcripts. Trinity constructs de Bruijn graph from short-read sequences and employs an enumeration algorithm to score all branches, and keeps possible ones as isoforms/transcripts.

In the experiment, 60952 short reads were created through Trinity sample genome bank with 519771 unique *k*-mers. The *k*-mer length, *k*, was initially set to default 25, and then changed to 22, 27, and 32 as typical values for most genome assemblers. To clarify, the CPU executes the *Inchworm*, *Chrysalis*, and *Butterfly* steps in Trinity, while PIM platforms run three main procedures in genome assembly shown in Fig. 58, i.e., Hashmap, DeBruijn, and Traverse for under-test PIM platforms. Trinity's power consumption and execution time are then compared to that of other PIM assemblers by several measures. To have a fair comparison with such a comprehensive assembler (that performs full genome assembly task with scaffolding step), we penalized the PIM platforms with ~25% excessive time and power. This could provide a more realistic comparison with a von-Neumann architecture-based assembler.

5.3.2.2 Run Time

The execution time of genome assembly task for different platforms is reported in Fig. 63. For *k*=25, the CPU platform executes the *Inchworm*, *Chrysalis*, and *Butterfly* steps [173] of Trinity in ~32s, where Chrysalis for clustering the contigs and constructing complete de Bruijn graph takes the largest fraction of the run time (28s) as expected. However, the comparison operation-intensive Hashmap procedure for *k*-mer analysis takes the largest fraction of execution time in all PIM platforms (over 40% of total run time). Larger *k*-mer length typically diminishes the de Bruijn graph connectivity by simultaneously reducing the number of ambiguous repeats in the graph and chance of overlap between two reads. That is why run time for all platforms reduces with increase of *k*-mer length.

We can observe that PIM platforms reduce the run time remarkably compared to the CPU. As shown, PANDA reduces the run time by ~18× compared to the CPU platform for *k*=25 (18.8× on average over 4 different *k*-mer lengths). The PANDA platform essentially accelerates the graph construction and traversal stages by ~21.5× compared with CPU platform. Now, by increasing the *k*-length to 32, the higher speed-up is even achievable. Compared with counterpart PIM platforms, our X(N)OR-friendly design reduces the run time on average by 4.2×, 2.5×, compared to STT-PIM [180], and SOT-PIM [2] platforms as the fastest counterparts, respectively. This comes from the fact that under-test PIM platforms require multi-cycle operations to implement addition operation. Besides, the SOT-based device intrinsically shows higher write

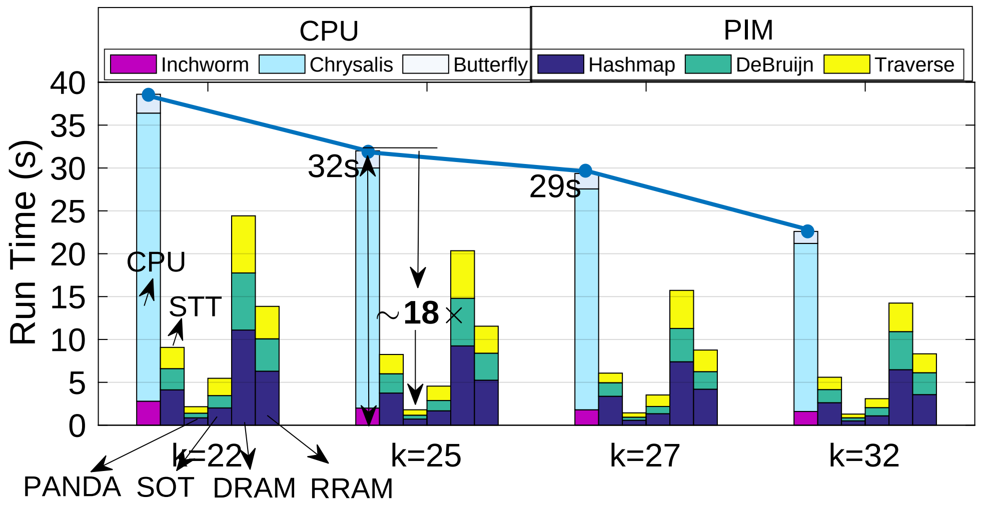


Figure 63: The breakdown of run time for under-test platforms running different k-mer-length genome assembly task. In each bar group from left to right: CPU, processing-in-STT-MRAM [180], PANDA, processing-in-SOT-MRAM/AlignS [2], processing-in-DRAM [19], and processing-in-RRAM [37].

speed compared to STT devices. Compared to DRAM and RRAM platforms, PANDA achieves on average 10.9× and 6× speed-up for various length k-mer processing. It is worth pointing out that the processing-in-DRAM platforms possess a destructive computing operation and require multiple memory cycle to copy the operands to particular rows before computation. As for Ambit [19], seven memory cycles are needed to implement in-memory-X(N)OR function.

5.3.2.3 Power Consumption

The power consumption of different PIM platforms was estimated for running different length k-mers compared to the CPU as shown in Fig. 64. Based on our results, a significant reduction in power consumption can be reported for all under-test PIM platforms compared with the CPU. The breakdown of energy consumption is also shown for the PIM platforms, however this couldn't be accurately achieved for the CPU and overall power consumption is reported. In the experiment, the processing-in-SOT-MRAM design/AlignS [2] achieves the smallest power consumption (on average) to run the three main procedures, as compared with the CPU and other PIM platforms. The PANDA platform stands as the second most power-efficient design. This is mainly due to the three-SA based bit-line computing scheme in PANDA compared with two-SA per bit-line technique in the counterpart design. While the proposed scheme brings more speed-up compared with the design in [2], it requires relatively more power. The PANDA reduces the power consumption by $\sim$9.2$\times$ on average compared with the CPU platform over different length *k*-mers. Besides, it reduces the power consumption by $\sim$18% compared with STT-MRAM [180] platform. The main reason behind this improvement is more efficient addition operation in PANDA. Addition operation requires additional memory cycles in the STT-MRAM [180] platform to save carry bit back to the memory and use it again for the computation of next bits. Compared to DRAM and RRAM platforms, PANDA obtains on average 2.11$\times$ and 55% power reduction for various length k-mer processing.

5.3.2.4 Speed-up/Power-Efficiency Trade-off

We investigate the power-efficiency and speed-up of three best under-test PIM platforms, based on the run time and power consumption results in the previous subsections, by tuning the number of active sub-arrays (N_s) associated with the comparison and addition operations. A parallelism degree (P_d) can be then defined as the number of replicated sub-arrays to boost the performance of the PIM platforms through parallel processing as shown in prior works [2], [23]. For example, when P_d is set

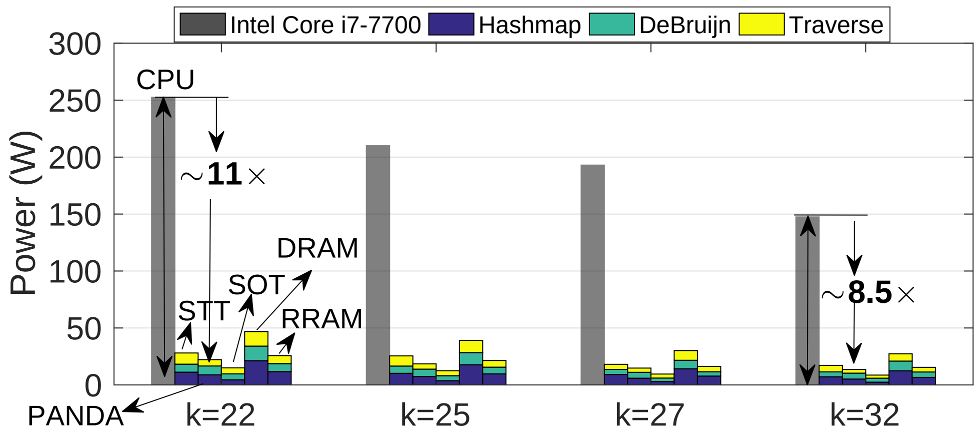


Figure 64: The breakdown of power consumption for PIM platforms running different *k*-mer-length genome assembly task compared to CPU. In each bar group from left to right: CPU, processing-in-STT-MRAM [180], PANDA, processing-in-SOT-MRAM/AlignS [2], processing-in-DRAM [19], and processing-in-RRAM [37].

to 2, two parallel sub-arrays are undertaken to process the in-memory operations, simultaneously. Such parallelism is expected to improve the performance of genome assembly at the cost of sacrificing the power consumption and area. Fig. 65 plots the existing trade-off between run time and power consumption vs. P_d for k= 25. The estimated CPU power budget required to execute Trinity is also shown. It can be seen that for all platforms the run time reduces by increasing the parallelism. For example for PANDA platform, in an extreme case, increasing P_d from 1 to 8 increases the power consumption from $\sim$19W to 128W ($\sim$7$\times$) and reduces the execution time by a factor of 3, which might not be a favorable case. Therefore, a user can meticulously tailor the PANDA performance to meet the system/application constraints. Here, the optimum theoretical performance of PANDA and other PIM platforms could be identified by pinpointing the intersection between power and run time curves in Fig. 65. We observe that PANDA achieves the smallest run time and power consumption task with a P_d $\sim$2 compared with the others.

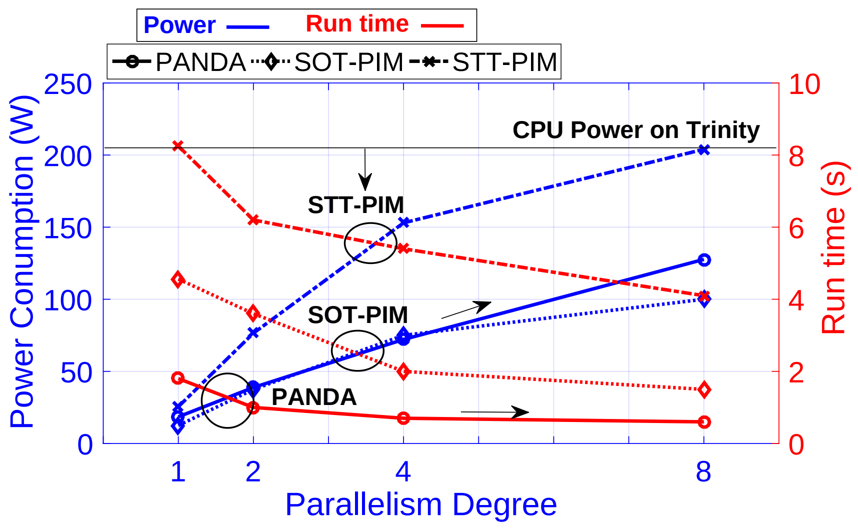


Figure 65: Trade-off between power consumption and run-time w.r.t. parallelism degree in *k*=25.

5.3.2.5 Memory Wall

The power-efficiency and speed-up of PIM platforms against the von-Neumann architecture-based CPU was discussed in prior subsections. Here, we further explore the reasons behind the numbers reported by considering two new measures, i.e., Memory Bottleneck Ratio (MBR) and Resource Utilization Ratio (RUR). We define MBR as the time fraction needed for data transfer from/to on-chip or off-chip, when computation has to wait for data, i.e., memory wall happens. We also define RUR as the time fraction in which the computation resources are loaded with data. The memory wall is considered as the main bottleneck that brings large power consumption and lengthen execution time in CPU.

The MBR is reported in Fig. 66a. The peak throughput for each design in four distinct *k*-mer lengths is taken into account for performing the evaluation. This evaluation mainly considers the number of memory access. As shown, the PANDA uses less than

∼17% time for data transfer due to the PIM acceleration schemes, while CPU's MBR increases to 65% when k=25. Besides, all the other PIM platforms except DRAM also spend less than ∼17% time for data communication. The smaller MBR can be translated as the higher RUR for the accelerators plotted in Fig. 66b. The less MBR can be understood as a higher RUR. With up to ∼82%, PANDA achieves the highest RUR. Taking everything into account, PIM acceleration schemes offer a high utilization ratio (>60% excluding DRAM) confirming the conclusion drawn in Fig. 66a. The memory wall evaluation shows the efficiency of the PANDA platform for solving memory wall challenge.

5.4 Summary

Chapter 5 presents in-memory acceleration schemes for two bioinformatics applications, 1-DNA short read alignment and 2-DNA genome assembly, based on the presented PIM platforms in Chapter 2. For the first application, by selecting AlignS [2] and PIM-Aligner [10] platforms, local data partitioning, mapping, and pipeline

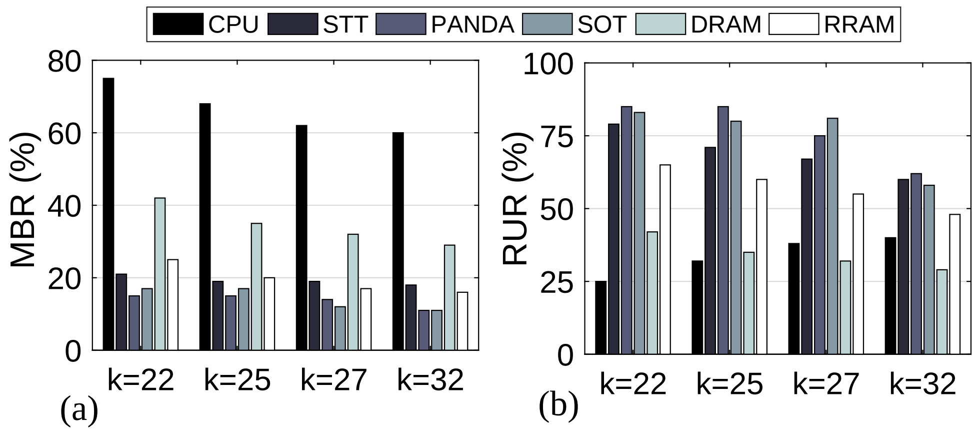


Figure 66: (a) Memory bottleneck ratio and (b) Resource utilization ratio for CPU and three under-test PIM platforms for running genome assembly task.

techniques were presented to maximize the parallelism in multiple computational sub-arrays while conducting the alignment task. The simulation results showed that PIM-Aligner outperforms recent platforms based on dynamic programming with $\sim$3.1$\times$ higher throughput per Watt. Besides, PIM-Aligner improves the short read alignment throughput per Watt per mm^2 by $\sim$9$\times$ and 1.9$\times$ compared to FM-index-based ASIC and processing-in-ReRAM designs, respectively.

For the second application, a highly parallel and step-by-step hardware-friendly DNA assembly algorithm was developed tailored for PANDA platform [51] that only requires the developed in-memory logic operations. The platform was then configured with a novel data partitioning and mapping technique that provides local storage and processing to utilize the algorithm-level's parallelism fully. The cross-layer simulation results demonstrated that PANDA platform reduces the run time and power, respectively, by a factor of 18 and 11 compared with CPU. Besides, speed-ups of up-to 2.5-10$\times$ can be obtained over other recent PIM platforms to perform the same task, like STT-MRAM, ReRAM, and DRAM.

Chapter 6

PROCESSING-IN-MEMORY ACCELERATION OF DATA ENCRYPTION AND GRAPH PROCESSING APPLICATIONS

6.1 Introduction

This chapter focuses on the PIM acceleration of data encryption and graph processing applications. First, the Advanced Encryption Standard (AES) algorithm is selected as a case study to elucidate the mapping of its transformations in ReDRAM platform [6] presented in Chapter 3, which reveals its benefits of energy-efficiency and high-throughput for in-memory data encryption applications. Then, to show the ReDRAM's efficacy in accelerating graph processing workloads, the matching-index task is selected and the required graph partitioning method is discussed. At the end, the performance and energy-efficiency of the proposed PIM accelerator is extensively assessed and compared with GPU, ASIC, and processing-in-DRAM counterparts with the bottom-up evaluation framework presented in Section 4.2.

6.2 Data Encryption

While the processor is typically the trust base, it is possible to rely on memory logic to do encryption in high-assurance computing systems in which the memory logic is attested and verified. In such a design, 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 a 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. 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 three different key lengths (128, 192, and 256 bits) [86]. For 128-bit key length, AES encrypts the input data after 10 rounds of consecutive transformations enumerated as SubBytes, ShiftRows, MixColumns, and AddRoundKey in Fig. 67a.

6.2.1 Mapping and Computation

To facilitate working with input data, each input byte data is distributed into 8-bit such that eight memory sub-arrays are filled by 4×4 bit-matrices, as shown in Fig. 67b. After mapping, the ReDRAM can support the required AES bulk bit-wise operations to accelerate each transformations inside the memory. As shown in [86], all transformations are mainly based on (N)AND and XOR operations.

SubBytes. In the SubBytes stage, each byte of S_M will undergo a Look-up table (LUT) based transformation using S-box and will be independently updated by a non-linear transformation f ($S_{i,j} \leftarrow f(S_{i,j})$). As simply depicted in Fig. 67d, the input of S-Box LUT (16 × 16 memory array) is essentially a Byte which is divided into two 4-bit data patterns. Each pattern yields a row or a column index for the decoders reaching target cell in S-box. Then, the addressed data byte in S-Box is written back to

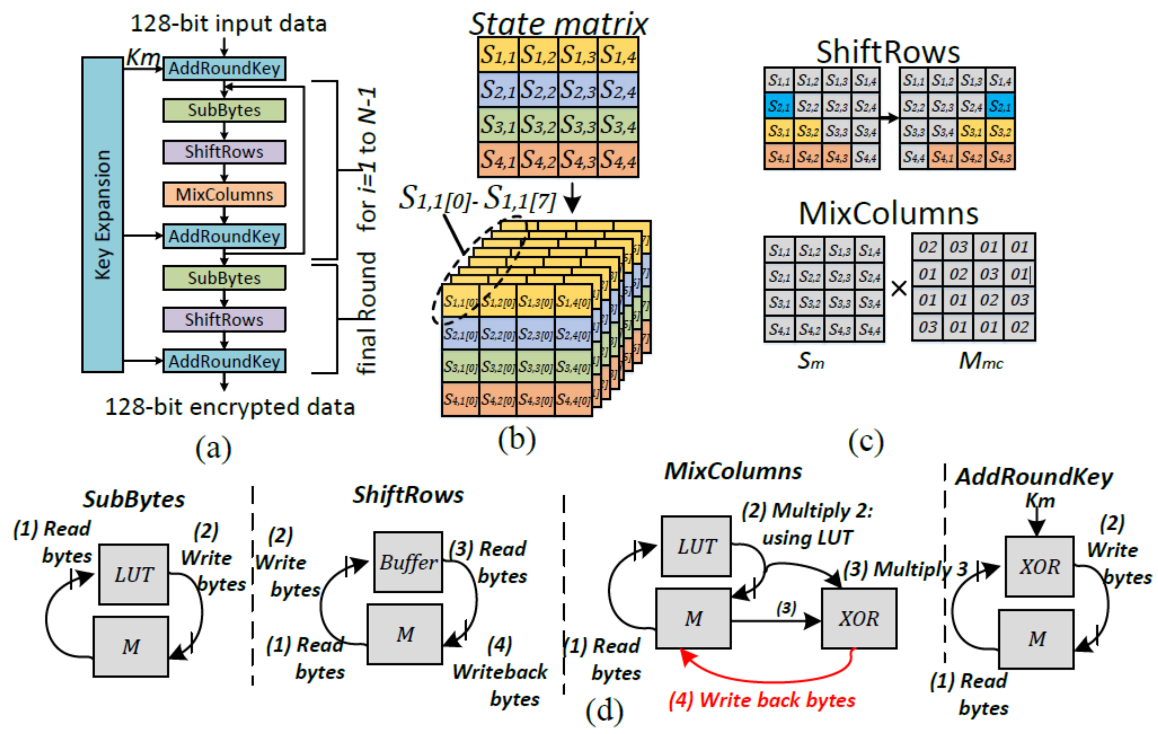


Figure 67: (a) AES block diagram, (b) State matrix partitioning, (c) Schematic representation of ShiftRows and MixColumns transformations, (d) The required computation of each transformation.

the memory unit and substitutes the original data. The S-box data are conventionally stored using SRAM leading to significant leakage power. However, it can be readily implemented within the ReDRAM (shown by LUT) leading to a much more efficient design. To maximize the parallelism in each level, eight ReDRAM's SA are used.

ShiftRows. In the ShiftRows stage, S_M will undergo a cyclically shift operation by a certain offset as shown in Fig. 67c. Algorithmically, the i-th row of S_M will be cyclically left shifted by i-1 bytes. Accordingly, the first row of state matrix is left unchanged. For the second to fourth rows, each byte is shifted by offsets of one to three, respectively. To perform the shift operation, one of the ReDRAM units is considered as a buffer to temporary save the readout data. In this way, after reading the data from the second to fourth row (3 rows), they can be easily rewritten to the memory with desired order.

MixColumns. In the MixColumns stage, the state matrix will be multiplied by a preset matrix depicted in Fig. 67c. The four bytes of each column of S_M are combined using an invertible linear transformation ($S_{(:),j} \leftarrow M_{mc} \times S_{(:),j}$). The prerequisite operations for this stage are addition, multiplication by two (xtime2), and multiplication by three (xtime3). The addition could be implemented by ReDRAM as discussed in Chapter 3; xtime2 can be implemented through shifting followed by a conditional bit-wise XOR with 0x1B; the xtime3 operation is defined as xtime2 result XOR with the original value. As discussed in Chapter 3, the bit-wise XOR, as the basic operations of MixColumns stage, can be efficiently executed by ReDRAM. As shown in Fig. 67d, to maximize the efficiency of AES performance, similar to the design in [86], a LUT-based transformation is used, followed by XOR operations.

AddRoundKey. In the AddRoundKey stage, the subkey is combined with the state matrix. For each round, key expansion unit produces a subkey derived from the main key using Rijndael's key schedule [181]. The 16-byte round keys are organized in a similar 4×4 array (K_M) as the state matrix with $K_{i,j}$ as matrix entry. In this process, each byte of state matrix will be replaced by bit-wise XOR result of $S_{i,j}$ and $K_{i,j}$ (subkey's corresponding bit). This stage can be easily performed using the in-memory XOR unit.

6.2.2 Experiment and Results

The performance of 128-bit AES implemented by a General Purpose Processor (GPP), ASIC, CMOL [182], Ambit [19], DRISA-3T1C [15], and ReDRAM, is measured in terms of energy consumption and number of cycles required for the process.

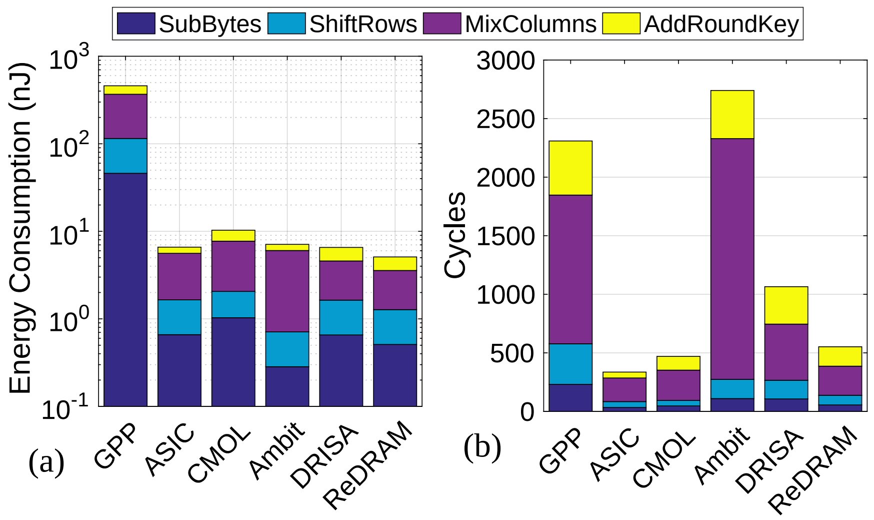


Figure 68: Breakdown of (a) Energy consumption and (b) Delay of different AES implementations.

For evaluation of AES performance in the GPP, we followed the method presented in [86] at 2GHz. AES C code is taken from [183] and compiled, then cycle-accurate gem5 simulator [184] is used to take AES binary and accordingly system level processor power evaluating tool McPAT [35] is used to estimate power dissipation. For evaluation of AES in CMOS ASIC (1.133GHz), Synopsys Design Compiler [113] tool is used.

Fig. 68a and Fig. 68b show the breakdown of energy[12] and number of cycles required for different AES transformations after mapping to the different platforms, respectively. The results show the ReDRAM's energy-efficiency (Fig. 68a) compared to other platforms. The ReDRAM reduces the energy consumption by ~23% compared to the CMOS-ASIC. From number of cycles stand point, we observe that MixColumns consumes the most clock cycles as well as energy due to the high number of resources (memory and in-memory XOR2) that it takes during operation. In some of the XOR-

[12]Y-axis in Fig. 68a: Log scale.

unfriendly platforms such as Ambit [19], MixColumns contributes to more than 70% of the energy consumption and number of cycles. Overall, ReDRAM requires the least number of cycles compared with other processing-in-DRAM platforms and GPP. However, ASIC (with 336 cycles) and CMOL (470) designs show better performance compared to ReDRAM (552).

6.3 Graph Processing

From graph processing algorithm perspective, network topology analysis can help us better understand the intricate connectivity of complex networks in practical problems. For instance, degree centrality is often used to measure the importance of a vertex. In social networks, people with more connections tend to have more significant influence in the community. The matching index is another basic topology parameter characterizes the similarity between two vertices in a network. It measures the ratio of common neighbors for pair of vertices. Evaluation of these network properties plays an essential part in potential applications, such as social network analysis and traffic flow control. The main goal of this section is to provide case studies of how important graph processing workloads can be partitioned and mapped to the ReDRAM array and how they can benefit from the PIM concept.

6.3.1 Mapping and Computation

Real world graph consists of millions of vertices and edges that need to be processed. To efficiently map such graphs into ReDRAM architecture, graph partitioning methods are used. Here, the interval-block partitioning method was adopted to balance work-

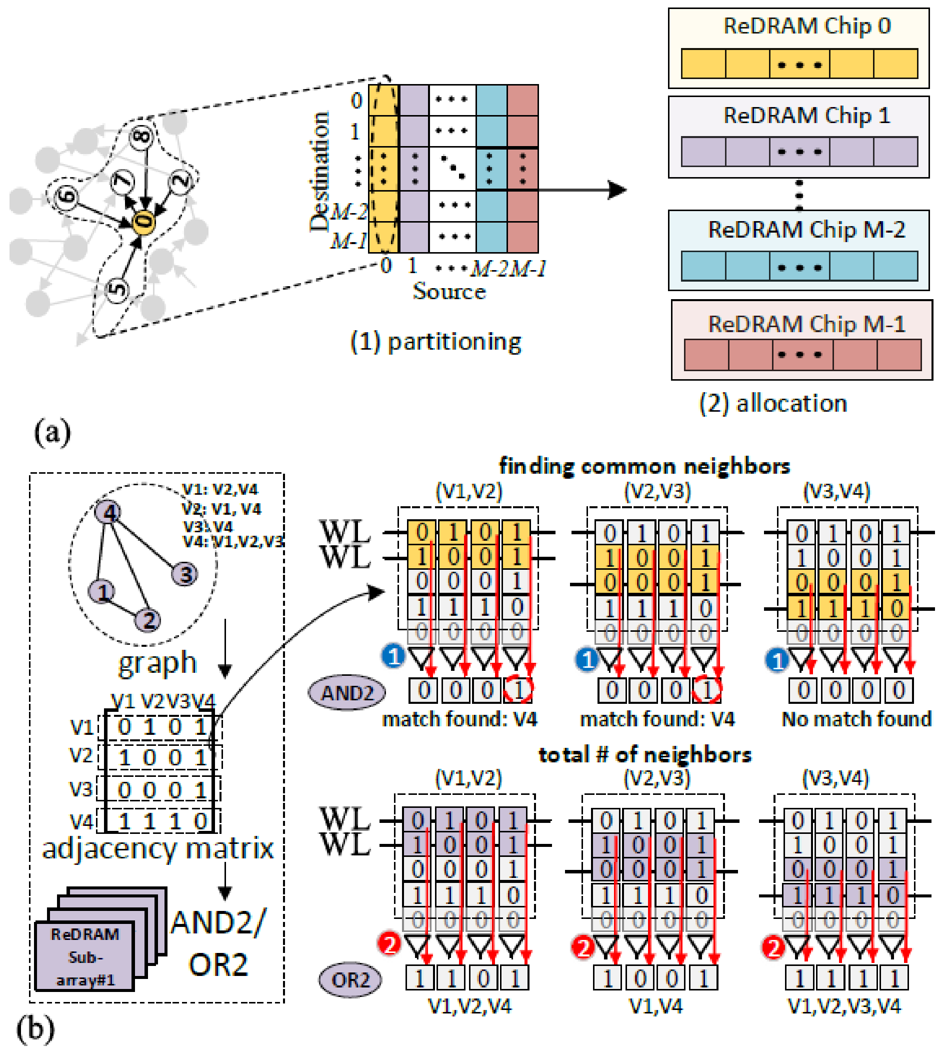


Figure 69: (a) Data partitioning and allocation in chip level, (b) ReDRAM's mapping and acceleration for finding matching index in sub-array level.

loads of each ReDRAM's chip and maximize parallelism. We use hash-based method [179] to split the vertices into M intervals and then divide edges into M^2 blocks as shown in Fig. 69a. The matching index $M_{i,j}$ quantifies the *similarity* between two vertices (V_i and V_j) based on the number of common neighbors shared by vertices as $(\frac{\sum \text{common neighbors}}{\sum \text{total number of neighbors}})$. The main task here is to find the common and total number of neighbors which can be implemented and accelerated by ReDRAM. Fig. 69b provides a straightforward example to elucidate the mapping and acceleration method of Re-

DRAM. After partitioning and allocation, the sample four-vertex network is converted to adjacency matrix and stored in 4 consecutive rows of sub-array. To find the common neighbors of two particular vertices (e.g., V1, V2), ReDRAM performs parallel AND2 on the rows and SA's outputs determine the matches (here, V4). In addition, the total number of neighbors is found by performing OR2 operation on the same rows. Then, ReDRAM can readily process the summation operation based on the ISA.

6.3.2 Experiment and Results

The ReDRAM's memory sub-array was configured with 1024 rows and 256 columns, 4$\times$4 mats (with 1/1 as row/column activation) per bank organized in H-tree routing manner, 16$\times$16 banks (with 1/1 as row/column activation) and 1024Mb total capacity. Therefore, an identical physical memory size (1024Mb) is considered for all PIM implementations henceforth exploiting the presented bottom-up cross-layer evaluation framework in Section 4.2. We developed an Ambit-like [19] accelerator for graph processing. Besides, a conventional architecture presented in [185] using HMC as main memory was selected without instruction offloading functionality. We also used the NVIDIA GTX 1080Ti Pascal GPU. The energy was measured with NVIDIA's system management interface. The achieved GPU results were scaled by 50% to exclude the energy consumed by cooling, etc. To estimate the performance of the accelerators, three social network data-sets were considered, as tabulated in Table 15.

Table 15: Social Network data-sets.

Dataset	Nodes	Edges	Graph Information
ego-Facebook	4,039	88,234	profiles & friends lists from Facebook
dblp-2010	326,186	1,615,400	scientific collaboration network
amazon-2008	735,323	5,158,388	similarity among books reported by Amazon store

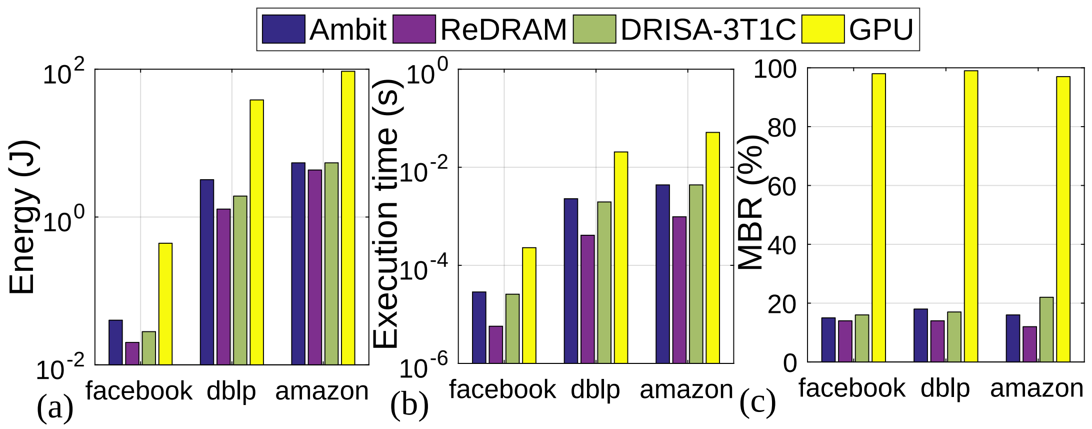


Figure 70: (a) Normalized energy consumption, (b) Execution time, (c) Memory bottleneck ratio of the accelerators.

Fig. 70a depicts the energy that four accelerators (Ambit [19], ReDRAM, DRISA-3T1C [15], and GPU) consume to perform matching-index task on different data-sets. The ReDRAM obtains the highest energy-efficiency compared to others due to the DRA mechanism. The ReDRAM consumes on average 2.5× less energy than that of Ambit accelerator. Compared to GPU, it reduces the energy consumption by ∼21×. Fig. 70b plots the execution time of the ReDRAM and other accelerators. We observe that ReDRAM solution is on average 5× faster than that of Ambit solution and 49× faster than GPU. This is mainly because of fast and parallel in-memory operations of ReDRAM, specifically for implementing AND2-OR2 operations. Fig. 70c also reports the Memory Bottleneck Ratio (MBR), which is the time fraction at which the computation has to wait for data and on-/off-chip data transfer obstructs its performance (memory wall happens) running matching index task on three data-sets. The experiment is performed according to the peak throughput for each platform considering number of memory access. The results reemphasize the PIM platform's efficiency for solving memory wall issue. We observe that ReDRAM along with other PIM solutions spend less than $\sim$22% time for memory access and data transfer. However, GPU accelerator spends more than 90% time waiting for the loading data.

6.4 Summary

This chapter discusses the PIM acceleration of data encryption and graph processing applications with mapping and partitioning, leveraging one of the presented PIM designs in Chapter 3. It is shown how the ReDRAM can be leveraged to greatly reduce energy consumption and latency of complex in-DRAM logic computations relying on state-of-the-art mechanisms based on triple-row activation, dual-contact cells, row initialization, NOR style, etc. As a graph processing accelerator, ReDRAM reduces energy consumption and execution time $\sim$21$\times$ and 49$\times$, respectively, compared with GPUs. As for data-encryption based on AES algorithm, it achieves 23% lower energy consumption compared to CMOS-ASIC implementation.

Chapter 7

CONCLUSIONS AND OUTLOOK

My doctoral research and dissertation have mainly focused on hardware and software co-design of energy-efficient and high-performance PIM platforms for big data applications and IoT. Leveraging innovations from both device and architecture, this dissertation tries to integrate memory and logic to break the existing memory and power walls. Generally, there are two high-level challenges and multiple sub-challenges that it aims to solve. First, the well-known *memory cost-sensitivity vs. reconfigurability* challenge is discussed. In this direction, we designed reconfigurable and low-overhead in-memory computing components on top of the existing NVM/VM circuit and architecture to make them simultaneously work as a memory and as a parallel, fast, reconfigurable PIM to process data within memory directly. For instance, the proposed ReDRAM platform in Chapter 3 uses the intrinsic analog operation of DRAM sub-arrays and elevates it to implement a full set of 1- and 2-input bulk bit-wise operations (NOT, (N)AND, (N)OR, and even X(N)OR) between operands stored in the same bit-line, in a single memory cycle, based on a new dual-row activation mechanism with a modest change to peripheral circuits such as sense amplifiers. Besides, such a platform can be leveraged to reduce energy consumption greatly and latency of complex in-DRAM logic computations relying on state-of-the-art mechanisms based on triple-row activation, dual-contact cells, row initialization, NOR style, etc. Second, the existing big data processing algorithms for deep neural networks, bioinformatics applications such as DNA alignment and assembly, data encryption, and graph processing tasks, are not essentially developed to work with non-Von-Neumann computing architectures. Such algorithms will impose a massive number of write-back operations that eventually may even fade the PIM benefits. Therefore, new customized in-memory computing algorithms and mapping methods were developed to convert the crucial iteratively-used functions to bit-wise PIM-supported functions. For instance, as shown in Chapter 4, MRIMA's in-memory AND-based bit-wise convolver and XOR-based adder schemes outperform recent PIM platforms in terms of number write-back operations leading to reduced energy and delay. Moreover, to quantitatively analyze the performance of various PIM platforms running big data applications, a generic and comprehensive evaluation framework was also presented. The overall system computing performance (throughput, latency, energy efficiency) for each application was then explored through the developed framework.

As future directions for this dissertation, I am interested in addressing the known and anticipating issues in the PIM's circuit and architecture. For the circuit level, the non-ideal sense margin that might cause incorrect output under the presence of wire resistance, process variation, etc., are the main objectives. An enhanced sense amplifier with a larger sense margin, error correction functionality, and other memory peripheral circuits are expected to design. Besides, as discussed, leveraging a PIM platform introduces new challenges for system programmers to circumvent. I will divide such architecture research into three sub-objectives: *(1) Programming model:* there is a great need to investigate how to integrate PIM instructions within a compiler to reduce the load on the programmer by activating smooth instruction offloading or library calls. Therefore, there are needs for designing and examining various compiler-based mechanisms to determine what portions of code should be offloaded to PIM platforms in a transparent fashion to the system programmer. *(2) Supporting Address Translation:* As discussed, the PIM platforms have their ISA with operations that can potentially use virtual addresses. To use virtual addresses, PIM's controller 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 PIM and giving the memory controller 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 PIM operates within a memory module. Therefore, exploring and designing efficient mechanisms for PIM-based virtual-to-physical address translation and access protection for the generality of applications is of great interest. *(3) Cache Coherence:* One primary concern that is common across most off-chip accelerators is cache coherence. When PIM 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 PIM will process later, PIM could use wrong/stale values. The exploration of various ways to solve cache coherence in accelerators is another architectural challenge that needs to be addressed.

REFERENCES

[1] S. Angizi, Z. He, A. Awad, and D. Fan, “Mrima: An mram-based in-memory accelerator”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 5, pp. 1123–1136, 2019.

[2] S. Angizi, J. Sun, W. Zhang, and D. Fan, “Aligns: A processing-in-memory accelerator for dna short read alignment leveraging sot-mram”, in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2019, pp. 1–6.

[3] Z. He, Y. Zhang, S. Angizi, B. Gong, and D. Fan, “Exploring a sot-mram based in-memory computing for data processing”, *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, pp. 676–685, 2018.

[4] S. Angizi, J. Sun, W. Zhang, and D. Fan, “Graphs: A graph processing accelerator leveraging sot-mram”, in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2019, pp. 378–383.

[5] 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.

[6] S. Angizi and D. Fan, “Redram: A reconfigurable processing-in-dram platform for accelerating bulk bit-wise operations”, in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2019, pp. 1–8.

[7] S. Angizi and D. Fan, “Graphide: A graph processing accelerator leveraging in-dram-computing”, in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 45–50.

[8] 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.

[9] S. Angizi, Z. He, D. Reis, X. S. Hu, W. Tsai, S. J. Lin, and D. Fan, “Accelerating deep neural networks in processing-in-memory platforms: Analog or digital approach?”, in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2019, pp. 197–202.

[10] S. Angizi, J. Sun, W. Zhang, and D. Fan, "Pim-aligner: A processing-in-mram platform for biological sequence alignment", in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2020, pp. 1265–1270.

[11] 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.

[12] *Fact sheet: Big data across the federal government (2012)*. [Online]. Available: http://%20www.whitehouse.gov/sites/default/files/microsites/ostp/big%20data%20fact%20sheet%203%2029%202012.pdf.

[13] 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.

[14] 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*, IEEE Press, vol. 44, 2016, pp. 27–39.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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 *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2017, pp. 273–287.

[20] 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.

[21] M. Imani, “Machine learning in iot systems: From deep learning to hyperdimensional computing”, PhD thesis, UC San Diego, 2020.

[22] S. Gupta, “Processing in memory using emerging memory technologies”, PhD thesis, UC San Diego, 2018.

[23] 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.

[24] Z. He, S. Angizi, F. Parveen, and D. Fan, “Leveraging dual-mode magnetic crossbar for ultra-low energy in-memory data encryption”, in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 83–88.

[25] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, “Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks”, *IEEE Journal of Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, 2020.

[26] Z. Jiang, S. Yin, J.-S. Seo, and M. Seok, “C3sram: An in-memory-computing sram macro based on robust capacitive coupling computing mechanism”, *IEEE Journal of Solid-State Circuits*, vol. 55, no. 7, pp. 1888–1897, 2020.

[27] X. Sun, S. Yin, X. Peng, R. Liu, J.-s. Seo, and S. Yu, “Xnor-rram: A scalable and parallel resistive synaptic architecture for binary neural networks”, in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2018, pp. 1423–1428.

[28] 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.

[29] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches", in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2017, pp. 481–492.

[30] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks", pp. 383–396, 2018.

[31] 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*, ACM, vol. 37, 2009, pp. 2–13.

[32] 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, pp. 1–18, 2018.

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

[34] 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.

[35] 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.

[36] S. Li, "Memory-centric architectures: Bridging the gap between compute and memory", PhD thesis, UC Santa Barbara, 2018.

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

[38] 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.

[39] S. Angizi, H. Jiang, R. F. DeMara, J. Han, and D. Fan, “Majority-based spin-cmos primitives for approximate computing”, *IEEE Transactions on Nanotechnology*, vol. 17, no. 4, pp. 795–806, 2018.

[40] S. Angizi, Z. He, Y. Bai, J. Han, M. Lin, R. F. DeMara, and D. Fan, “Leveraging spintronic devices for efficient approximate logic and stochastic neural networks”, in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, 2018, pp. 397–402.

[41] S. Angizi and D. Fan, “Deep neural network acceleration in non-volatile memory: A digital approach”, in *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, IEEE, 2019, pp. 1–6.

[42] H. Jiang, S. Angizi, D. Fan, J. Han, and L. Liu, “Non-volatile approximate arithmetic circuits using scalable hybrid spin-cmos majority gates”, *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 3, pp. 1217–1230, 2021.

[43] S. Angizi, Z. He, R. F. DeMara, and D. Fan, “Composite spintronic accuracy-configurable adder for low power digital signal processing”, in *2017 18th International Symposium on Quality Electronic Design (ISQED)*, IEEE, 2017, pp. 391–396.

[44] S. Angizi, Z. He, and D. Fan, “Energy efficient in-memory computing platform based on 4-terminal spin hall effect-driven domain wall motion devices”, in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 77–82.

[45] D. Fan and S. Angizi, “Energy efficient in-memory binary deep neural network accelerator with dual-mode sot-mram”, in *2017 IEEE International Conference on Computer Design (ICCD)*, IEEE, 2017, pp. 609–612.

[46] S. Angizi, Z. He, and D. Fan, “Dima: A depthwise cnn in-memory accelerator”, in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2018, pp. 1–8.

[47] A. Roohi, S. Angizi, D. Fan, and R. F. DeMara, “Processing-in-memory acceleration of convolutional neural networks for energy-effciency, and power-intermittency resilience”, in *20th International Symposium on Quality Electronic Design (ISQED)*, IEEE, 2019, pp. 8–13.

[48] S. Angizi, N. A. Fahmi, W. Zhang, and D. Fan, “Pim-assembler: A processing-in-memory platform for genome assembly”, in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6.

[49] 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 *2017 IEEE Computer Society annual symposium on VLSI (ISVLSI)*, IEEE, 2017, pp. 45–50.

[50] S. Angizi, Z. He, A. Chen, and D. Fan, "Hybrid spin-cmos polymorphic logic gate with application in in-memory computing", *IEEE Transactions on Magnetics*, vol. 56, no. 2, pp. 1–15, 2020.

[51] S. Angizi, N. A. Fahmi, W. Zhang, and D. Fan, "Panda: Processing-in-mram accelerated de bruijn graph based dna assembly", *arXiv preprint arXiv:2008.06177*, 2020.

[52] 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. 025 001, 2011.

[53] 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.

[54] 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. 102 509, 2011.

[55] 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.

[56] 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.

[57] F. Parveen, Z. He, S. Angizi, and D. Fan, "Hybrid polymorphic logic gate with 5-terminal magnetic domain wall motion device", in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2017, pp. 152–157.

[58] D. Fan, Z. He, and S. Angizi, "Leveraging spintronic devices for ultra-low power in-memory computing: Logic and neural network", in *2017 IEEE 60th*

International Midwest Symposium on Circuits and Systems (MWSCAS), IEEE, 2017, pp. 1109–1112.

[59] 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.

[60] A. S. Rakin, S. Angizi, Z. He, and D. Fan, "Pim-tgan: A processing-in-memory accelerator for ternary generative adversarial networks", in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, IEEE, 2018, pp. 266–273.

[61] Z. He, S. Angizi, and D. Fan, "Current-induced dynamics of multiple skyrmions with domain-wall pair and skyrmion-based majority gate design", *IEEE Magnetics Letters*, vol. 8, pp. 1–5, 2017.

[62] 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.

[63] S.-W. Chung, T. Kishi, J. Park, M. Yoshikawa, K. Park, T. Nagase, K. Sunouchi, H. Kanaya, G. Kim, K. Noma, *et al.*, "4gbit density stt-mram using perpendicular mtj realized with compact cell structure", in *2016 IEEE International Electron Devices Meeting (IEDM)*, IEEE, 2016, pp. 27–1.

[64] K. Garello, F. Yasin, H. Hody, S. Couet, L. Souriau, S. Sharifi, J. Swerts, R. Carpenter, S. Rao, W. Kim, *et al.*, "Manufacturable 300mm platform solution for field-free switching sot-mram", in *2019 Symposium on VLSI Circuits*, IEEE, 2019, T194–T195.

[65] M. Natsui, A. Tamakoshi, H. Honjo, T. Watanabe, T. Nasuno, C. Zhang, T. Tanigawa, H. Inoue, M. Niwa, T. Yoshiduka, *et al.*, "Dual-port field-free sot-mram achieving 90-mhz read and 60-mhz write operations under 55-nm cmos technology and 1.2-v supply voltage", in *2020 IEEE Symposium on VLSI Circuits*, IEEE, 2020, pp. 1–2.

[66] M. Natsui, A. Tamakoshi, H. Honjo, T. Watanabe, T. Nasuno, C. Zhang, T. Tanigawa, H. Inoue, M. Niwa, T. Yoshiduka, *et al.*, "Dual-port sot-mram achieving 90-mhz read and 60-mhz write operations under field-assistance-free condition", *IEEE Journal of Solid-State Circuits*, 2020.

[67] 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 perpen-

dicular 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.

[68] G. Autes, J. Mathon, and A. Umerski, "Strong enhancement of the tunneling magnetoresistance by electron filtering in an fe/mgo/fe/gaas (001) junction", *Physical review letters*, p. 217 202, 2010.

[69] P. Mavropoulos, M. Levzaic, and S. Blugel, "Half-metallic ferromagnets for magnetic tunnel junctions by ab initio calculations", *Physical Review B*, vol. 72, no. 17, p. 174 428, 2005.

[70] M. Bowen, M. Bibes, A. Barthelemy, J.-P. Contour, A. Anane, Y. Lemaitre, 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.

[71] D. Fan, S. Angizi, and Z. He, "In-memory computing with spintronic devices", in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2017, pp. 683–688.

[72] J. Hayakawa, S. Ikeda, F. Matsukura, H. Takahashi, and H. Ohno, "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.

[73] M. J. Donahue, "Oommf user's guide, version 1.0", *-6376*, 1999.

[74] 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*, 2011, pp. 51–54.

[75] Z. He, S. Angizi, and D. Fan, "Exploring stt-mram based in-memory computing paradigm with application of image edge extraction", in *2017 IEEE International Conference on Computer Design (ICCD)*, IEEE, 2017, pp. 439–446.

[76] D. Fan, S. Maji, K. Yogendra, M. Sharad, and K. Roy, "Injection-locked spin hall-induced coupled-oscillators for energy efficient associative computing", *IEEE Transactions on Nanotechnology*, vol. 14, no. 6, pp. 1083–1093, 2015.

[77] G. Panagopoulos, C. Augustine, and K. Roy, “A framework for simulating hybrid mtj/cmos circuits: Atoms to system approach”, in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2012, pp. 1443–1446.

[78] Y. Huai, “Spin-transfer torque mram (stt-mram): Challenges and prospects”, *AAPPS bulletin*, vol. 18, no. 6, pp. 33–40, 2008.

[79] D. Fan, “Boolean and brain-inspired computing using spin-transfer torque devices”, 2015.

[80] L. Liu, T. Moriyama, D. Ralph, and R. Buhrman, “Spin-torque ferromagnetic resonance induced by the spin hall effect”, *Physical review letters*, vol. 106, no. 3, p. 036 601, 2011.

[81] L. Liu, C.-F. Pai, Y. Li, H. Tseng, D. Ralph, and R. Buhrman, “Spin-torque switching with the giant spin hall effect of tantalum”, *Science*, vol. 336, no. 6081, pp. 555–558, 2012.

[82] C.-F. Pai, L. Liu, Y. Li, H. Tseng, D. Ralph, and R. Buhrman, “Spin transfer torque devices utilizing the giant spin hall effect of tungsten”, *Applied Physics Letters*, vol. 101, no. 12, p. 122 404, 2012.

[83] Y. Niimi, Y. Kawanishi, D. Wei, C. Deranlot, H. Yang, M. Chshiev, T. Valet, A. Fert, and Y. Otani, “Giant spin hall effect induced by skew scattering from bismuth impurities inside thin film cubi alloys”, *Physical review letters*, vol. 109, no. 15, p. 156 602, 2012.

[84] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, “Binary convolutional neural network on rram”, in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2017, pp. 782–787.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] M. Imani, S. Gupta, Y. Kim, and T. Rosing, “Floatpim: In-memory acceleration of deep neural network training with high precision”, in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2019, pp. 802–815.

[91] S. Angizi and D. Fan, “Imc: Energy-efficient in-memory convolver for accelerating binarized deep neural network”, in *Proceedings of the Neuromorphic Computing Symposium*, 2017, pp. 1–8.

[92] Z. He, S. Angizi, F. Parveen, and D. Fan, “High performance and energy-efficient in-memory computing architecture based on sot-mram”, in *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, IEEE, 2017, pp. 97–102.

[93] 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.

[94] 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.

[95] S. V. Kosonocky, M. Immediato, P. Cottrell, T. Hook, R. Mann, and J. Brown, “Enchanced multi-threshold (mtcmos) circuits using variable well bias”, in *Proceedings of the 2001 international symposium on Low power electronics and design*, 2001, pp. 165–169.

[96] H. Ozdemir, A. Kepkep, B. Pamir, Y. Leblebici, and U. Cilingiroglu, “A capacitive threshold-logic gate”, *IEEE Journal of Solid-State Circuits*, vol. 31, no. 8, pp. 1141–1150, 1996.

[97] K. Navi, V. Foroutan, M. R. Azghadi, M. Maeen, M. Ebrahimpour, M. Kaveh, and O. Kavehei, "A novel low-power full-adder cell with new technique in designing logical gates based on static cmos inverter", *Microelectronics Journal*, vol. 40, no. 10, pp. 1441–1448, 2009.

[98] R. Zhang, K. Walus, W. Wang, and G. A. Jullien, "A method of majority logic reduction for quantum cellular automata", *IEEE Transactions on Nanotechnology*, vol. 3, no. 4, pp. 443–450, 2004.

[99] (2011). Ncsu eda freepdk45, [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents.

[100] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator", *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2016.

[101] S. Angizi and D. Fan, "Accelerating bulk bit-wise x(n)or operation in processing-in-dram platform", *arXiv preprint arXiv:1904.05782*, 2019.

[102] V. Seshadri, K. Hsieh, A. Boroum, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast bulk bitwise and and or in dram", *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 127–131, 2015.

[103] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, *et al.*, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization", in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2013, pp. 185–197.

[104] H. B. Kang and S. K. Hong, *One-transistor type dram*, US Patent 7,701,751, 2010.

[105] S.-L. Lu, Y.-C. Lin, and C.-L. Yang, "Improving dram latency with dynamic asymmetric subarray", in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2015, pp. 255–266.

[106] G. Sideris, "Intel 1103-mos memory that defied cores", *Electronics*, vol. 46, no. 9, pp. 108–113, 1973.

[107] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "Dracc: A dram based accelerator for accurate cnn inference", in *Proceedings of the 55th Annual Design Automation Conference*, ACM, 2018, p. 168.

[108] (2018). Parallel thread execution isa version 6.1, [Online]. Available: http://docs.nvidia.com/cuda/parallel-thread-execution/index.html.

[109] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm design exploration", in *ISQED*, IEEE Computer Society, 2006, pp. 585–590.

[110] 2. DRAM Power Model. https://www.rambus.com/energy/.

[111] M. W. Allam, M. H. Anis, and M. I. Elmasry, "High-speed dynamic logic styles for scaled-down cmos and mtcmos technologies", in *Proceedings of the 2000 international symposium on Low power electronics and design*, ACM, 2000, pp. 155–160.

[112] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "Cacti 5.1", Technical Report HPL-2008-20, HP Labs, Tech. Rep., 2008.

[113] Synopsys, Inc., *Synopsys design compiler, product version 14.9.2014*, version 14.9.2014, 2014.

[114] *6th generation intel core processor family datasheet*. [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/core/core-vpro/i7-6700.html.

[115] *Geforce gtx 1080 ti*. [Online]. Available: https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080-ti/.

[116] *Hybrid memory cube speci!cation 2.0*. [Online]. Available: http://www.hybridmemorycube.org/files/SiteDownloads/HMC-30G-VSR_HMCC_Specification_Rev2.0_Public.pdf..

[117] L. Cavigelli, M. Magno, and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks", in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[118] 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.

[119] 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.

[120] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications", *arXiv preprint arXiv:1412.7024*, 2014.

[121] S. Lin, "Platform-specific model compression for deep neural networks with joint methods", PhD thesis, Northeastern University, 2020.

[122] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks", in *European conference on computer vision*, Springer, 2016, pp. 525–542.

[123] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations", *arXiv preprint arXiv:1511.00363*, 2015.

[124] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas", in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ACM, 2017, pp. 15–24.

[125] Z. He, S. Angizi, A. S. Rakin, and D. Fan, "Bd-net: A multiplication-less dnn with binarized depthwise separable convolution", in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2018, pp. 130–135.

[126] Z. He, L. Yang, S. Angizi, A. S. Rakin, and D. Fan, "Sparse bd-net: A multiplication-less dnn with sparse binarized depth-wise separable convolution", *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 2, pp. 1–24, 2020.

[127] A. Roohi, S. Sheikhfaal, S. Angizi, D. Fan, and R. F. DeMara, "Apgan: Approximate gan for robust low energy learning from imprecise components", *IEEE Transactions on Computers*, vol. 69, no. 3, pp. 349–360, 2019.

[128] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks", *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[129] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, "A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications", *Nano letters*, vol. 12, no. 1, pp. 389–395, 2012.

[130] Z. He, J. Lin, R. Ewetz, J.-S. Yuan, and D. Fan, “Noise injection adaption: End-to-end reram crossbar non-ideal effect adaption for neural network mapping”, in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[131] I. Chakraborty, D. Roy, and K. Roy, “Technology aware training in memristive neuromorphic systems for nonideal synaptic crossbars”, *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 5, pp. 335–344, 2018.

[132] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, “Rx-caffe: Framework for evaluating and training deep neural networks on resistive crossbars”, *arXiv preprint arXiv:1809.00072*, 2018.

[133] Z. He and D. Fan, “Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 438–11 446.

[134] D. Reis, D. Gao, S. Angizi, X. Yin, D. Fan, M. Niemier, C. Zhuo, and X. S. Hu, “Modeling and benchmarking computing-in-memory for design space exploration”, in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 39–44.

[135] Z. He, “Efficient and secure deep learning inference system: A software and hardware co-design perspective”, PhD thesis, Arizona State University, 2020.

[136] T.-h. Yang and M.-f. Chang, *Sense amplifier of resistive memory and operating method thereof*, US Patent App. 15/939,262, 2019.

[137] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, “Memristor crossbar-based neuromorphic computing system: A case study”, *IEEE transactions on neural networks and learning systems*, vol. 25, no. 10, pp. 1864–1878, 2014.

[138] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, “Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[139] 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.

[140] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[141] Z. He, S. Angizi, and D. Fan, "Accelerating low bit-width deep convolution neural network in mram", in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, 2018, pp. 533–538.

[142] 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.

[143] 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.

[144] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning", in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, 2011, p. 5.

[145] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning", in *12th USENIX symposium on operating systems design and implementation*, 2016, pp. 265–283.

[146] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches", *HP Laboratories*, pp. 22–31, 2009.

[147] 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.

[148] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000 x acceleration on long read assembly", in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 2018, pp. 199–213.

[149] F. Zokaee, H. R. Zarandi, and L. Jiang, "Aligner: A process-in-memory architecture for short read alignment in rerams", *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 237–240, 2018.

[150] S. Angizi, W. Zhang, and D. Fan, "Exploring dna alignment-in-memory leveraging emerging sot-mram", in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 277–282.

[151] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short dna sequences to the human genome", *Genome biology*, vol. 10, no. 3, pp. 1–10, 2009.

[152] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform", *bioinformatics*, vol. 25, pp. 1754–1760, 2009.

[153] R. Luo, T. Wong, J. Zhu, C.-M. Liu, X. Zhu, E. Wu, L.-K. Lee, H. Lin, W. Zhu, D. W. Cheung, *et al.*, "Soap3-dp: Fast, accurate and sensitive gpu-based short read aligner", *PloS one*, vol. 8, no. 5, e65632, 2013.

[154] A. Madhavan, T. Sherwood, and D. Strukov, "Race logic: A hardware acceleration for dynamic programming algorithms", in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, IEEE, 2014, pp. 517–528.

[155] Y.-C. Wu, C.-H. Chang, J.-H. Hung, and C.-H. Yang, "A 135-mw fully integrated data processor for next-generation sequencing", *IEEE transactions on biomedical circuits and systems*, vol. 11, no. 6, pp. 1216–1225, 2017.

[156] J. Arram, T. Kaplan, W. Luk, and P. Jiang, "Leveraging fpgas for accelerating short read alignment", *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 14, no. 3, pp. 668–677, 2017.

[157] S. K. Khatamifard, Z. Chowdhury, N. Pande, M. Razaviyayn, C. Kim, and U. R. Karpuzcu, "A non-volatile near-memory read mapping accelerator", *arXiv preprint arXiv:1709.02381*, 2017.

[158] W. Huangfu, S. Li, X. Hu, and Y. Xie, "Radar: A 3d-reram based dna alignment accelerator architecture", in *55th DAC*, ACM, 2018, p. 59.

[159] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar, "Resistive associative processor", *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 148–151, 2014.

[160] F. Zokaee, M. Zhang, and L. Jiang, "Finder: Accelerating fm-index-based exact pattern matching in genomic sequences through reram technology", in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, IEEE, 2019, pp. 284–295.

[161] S. Canzar and S. L. Salzberg, “Short read mapping: An algorithmic tour”, *Proceedings of the IEEE*, vol. 105, no. 3, pp. 436–458, 2017.

[162] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser, “A resistive cam processing-in-storage architecture for dna sequence alignment”, *IEEE Micro*, vol. 37, no. 4, pp. 20–28, 2017.

[163] W. Huang, L. Li, J. R. Myers, and G. T. Marth, “Art: A next-generation sequencing read simulator”, *Bioinformatics*, vol. 28, no. 4, pp. 593–594, 2012.

[164] H. Li and N. Homer, “A survey of sequence alignment algorithms for next-generation sequencing”, *Briefings in bioinformatics*, vol. 11, no. 5, pp. 473–483, 2010.

[165] E. Georganas, A. Bulucc, J. Chapman, L. Oliker, D. Rokhsar, and K. Yelick, “Parallel de bruijn graph construction and traversal for de novo genome assembly”, in *SC’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, 2014, pp. 437–448.

[166] J. A. Chapman, I. Ho, S. Sunkara, S. Luo, G. P. Schroth, and D. S. Rokhsar, “Meraculous: De novo genome assembly with short paired-end reads”, *PloS one*, vol. 6, no. 8, e23501, 2011.

[167] R. Li, C. Yu, Y. Li, T.-W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang, “Soap2: An improved ultrafast tool for short read alignment”, *Bioinformatics*, vol. 25, no. 15, pp. 1966–1967, 2009.

[168] C.-M. Liu, T. Wong, E. Wu, R. Luo, S.-M. Yiu, Y. Li, B. Wang, C. Yu, X. Chu, K. Zhao, R. Li, and T.-W. Lam, “Soap3: Ultra-fast gpu-based parallel alignment tool for short reads”, *Bioinformatics*, vol. 28, no. 6, pp. 878–879, 2012.

[169] J. Arram, T. Kaplan, W. Luk, and P. Jiang, “Leveraging fpgas for accelerating short read alignment”, *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 14, no. 3, pp. 668–677, 2016.

[170] S. F. Mahmood and H. Rangwala, “Gpu-euler: Sequence assembly using gpgpu”, in *2011 IEEE International Conference on High Performance Computing and Communications*, IEEE, 2011, pp. 153–160.

[171] B. S. C. Varma, K. Paul, and M. Balakrishnan, “Fpga-based acceleration of de novo genome assembly”, in *Architecture Exploration of FPGA Based Accelerators for BioInformatics Applications*, Springer, 2016, pp. 55–79.

[172] D. R. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de bruijn graphs", *Genome research*, vol. 18, pp. 821–829, 2008.

[173] M. G. Grabherr, B. J. Haas, M. Yassour, J. Z. Levin, D. A. Thompson, I. Amit, X. Adiconis, L. Fan, R. Raychowdhury, Q. Zeng, and Z. Chen, "Full-length transcriptome assembly from rna-seq data without a reference genome", *Nature biotechnology*, vol. 29, no. 7, pp. 644–652, 2011.

[174] S. Goswami, K. Lee, S. Shams, and S.-J. Park, "Gpu-accelerated large-scale genome assembly", in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2018, pp. 814–824.

[175] S. Ren, N. Ahmed, K. Bertels, and Z. Al-Ars, "An efficient gpu-based de bruijn graph construction algorithm for micro-assembly", in *2018 IEEE 18th International Conference on Bioinformatics and Bioengineering (BIBE)*, IEEE, 2018, pp. 67–72.

[176] M. Lu, Q. Luo, B. Wang, J. Wu, and J. Zhao, "Gpu-accelerated bidirected de bruijn graph construction for genome assembly", in *Asia-Pacific Web Conference*, Springer, 2013, pp. 51–62.

[177] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, J. Wang, and J. Wang, "De novo assembly of human genomes with massively parallel short read sequencing", *Genome research*, vol. 20, no. 2, pp. 265–272, 2010.

[178] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, "Abyss: A parallel assembler for short read sequence data", *Genome research*, vol. 19, no. 6, pp. 1117–1123, 2009.

[179] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, and H. Yang, "Graphh: A processing-in-memory architecture for large-scale graph processing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 640–653, 2018.

[180] 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*, vol. 26, no. 3, pp. 470–483, 2017.

[181] N.-F. Standard, "Announcing the advanced encryption standard (aes)", *FIPSP*, vol. 197, 2001.

[182] 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.

[183] K. Malbrain, *Byte-oriented-aes: A public domain byte-oriented implementation of aes in c*, 2009.

[184] 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.

[185] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, “Graphpim: Enabling instruction-level pim offloading in graph computing frameworks”, in *2017 IEEE International symposium on high performance computer architecture (HPCA)*, IEEE, 2017, pp. 457–468.

BIOGRAPHY

Shaahin Angizi received his Ph.D. degree in Electrical Engineering at the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, in 2021, under the supervision of Dr. Deliang Fan. He received his B.Sc. and M.Sc. in Computer Engineering from South Tehran Branch and Science and Research Branch of Azad University, Tehran, Iran, in 2012 and 2014, respectively. He has authored and co-authored +70 research papers in top-ranked international journals such as IEEE TNANO, IEEE TCAD, IEEE TC, IEEE TCASI, IEEE TETC, IEEE TMAG, etc., and top-tier EDA conferences including, DAC, DATE, ICCAD, ASP-DAC, ICCD, etc. As a graduate student, he has received prestigious Fellowship and Scholarship awards during his Ph.D. He is also the recipient of the Best Ph.D. Research Award (1st-place) of 2018 Ph.D. Forum at Design Automation Conference (DAC), two Best Paper Awards of IEEE Computer Society Annual Symposium on VLSI (ISVLSI) in 2017 and 2018, and the Best Paper Award of ACM Great Lakes Symposium on VLSI (GLSVLSI) in 2019. His primary research interests include In-Memory Computing Based on Volatile & Non-Volatile Memories, Accelerator Design for Deep Neural Networks, Bioinformatics, Graph Processing, and Low Power and Area-efficient In-Sensor Computing for IoT.