University of California
Santa Barbara

Memory-Centric Architectures: Bridging the Gap Between Compute and Memory

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Shuangchen Li

Committee in charge:

Professor Yuan Xie, Chair
Professor Margaret Marek-Sadowska
Professor Dmitri Strukov
Professor William Yang Wang
Dr. Hongzhong Zheng

March 2018

The Dissertation of Shuangchen Li is approved.

Professor Margaret Marek-Sadowska

Professor Dmitri Strukov

Professor William Yang Wang

Dr. Hongzhong Zheng

Professor Yuan Xie, Committee Chair

January 2018

Memory-Centric Architectures: Bridging the Gap Between Compute and Memory

by

Shuangchen Li

Acknowledgements

First of all, I sincerely thank my advisor, Prof. Yuan Xie. I have learnt a lot from his board knowledge, wisdom, passionate, skills and I wish I could have learnt them all. His research foresight is what I admire and benefit from the most. This thesis is an example of his great vision. Even more, he almost offers everything and leverages all the resource that he has to make me a better researcher. Words are to pale to express all my gratitude.

Many thanks to Prof. Margaret Marek-Sadowska, Prof. Dmitri Strukov, Prof. William Wang, and Dr. Hongzhong Zheng for serving in my dissertation committee and providing valuable feedback to my qualify exam and this dissertation.

I would like to express my special thank to Prof. Sharon Hu and Prof. Yongpan Liu, who first guided me into the research world during my Master study; Dr. Kaisheng Ma, who taught and helped me a lot for turning into a computer architect when I was in Penn State University. Also, I feel so lucky to have the chance working with Prof. Jishen Zhao and Dr. Cong Xu, who taught me so much knowledge and skills that I will always regard them as my mentors. I would also like to thank Dr. Hang Zhang, who was an alive Google Scholar and offered me lots of help.

I sincerely thank coauthors of my research papers, including but not limited to Dr. Ping Chi, Dr. Kaisheng Ma, Peng Gu, Liu Liu, Yu Ji, Dr. Tao Zhang, and Dr. Krishna Malladi. I also sincerely thank Dr. Dimin Niu, Dr. Hongzhong Zheng, Dr. Niladrish Chatterjee, and Dr. Mike O'Connor for their insightful mentoring during my internships.

I would like to thank all my friends in both PSU and UCSB and all members in the MDL group and the SEAL group, for keeping me accompany in this journey, encouraging and inspiring me, making me all these good memories. Especially Dr. Kaisheng Ma, Ziyang Qi, Xulong Tang, Dr. Cong Xu, Dr. Ping Chi, Dr. Jia Zhan, Hang Zhang, Itir Akgun, Linuo Xue, Liu Liu, Peng Gu, Maohua Zhu, Dylan Stow, Dr. Chao Zhang, and Dr. Xing Hu.

Last but not the least, I would like to thank my parents for their unconditional love and support, and all my friends for making my life vivid.

Curriculum Vitæ
Shuangchen Li

Education

2018	Ph.D. in Electrical and Computer Engineering (Expected), University of California, Santa Barbara.
2014	M.A. in Electrical Engineering, Tsinghua University.
2011	B.S. in Electrical Engineering, Tsinghua University

Publications

[1]. Wei-Hao Chen, Wen-Jang Lin, **Shuangchen Li**, Li-Ya Lai, Jian-Wei Su, Huan-Ting Lin, Chien-Hua Hsu, Heng-Yuan Lee, Yuan Xie, Shyh-Shyuan Sheu, and Meng-Fan Chang "A 16Mb Dual-Mode ReRAM Macro with Sub-14ns Computing-In-Memory and Memory Functions Enabled by Self-Write Termination Scheme." Proc. *IEEE International Electron Devices Meeting (IEDM)*, 2017.

[2]. **Shuangchen Li**, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie "DRISA: A DRAM-based Reconfigurable In-Situ Accelerator." Proc. *International Symposium on Microarchitecture (MICRO)*, 2017.

[3]. Liu Liu, Ping Chi, **Shuangchen Li**, Yuanqing Cheng, and Yuan Xie. "Processing-In-Memory Architecture Design for Accelerating Neuro-Inspired Algorithms." Proc. *Neuro-inspired Computing Using Resistive Synaptic Devices*, 2017.

[4]. Ping Chi, **Shuangchen Li**, and Yuan Xie. "Building Energy-Efficient Multi-Level Cell STT-RAM Caches with Data Compression." Proc. *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017.

[5]. Chi Ping and **Shuangchen Li (Equal Contribution)**, Cong Xu, Tao Zhang, Jishen Zhao, Yu Wang, Yongpan Liu, Yuan Xie "PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory." Proc. *International Symposium on Computer Architecture (ISCA)*, 2016.

[6]. Yu Ji, Youhui Zhang, **Shuangchen Li**, Ping Chi, Cihang Jiang, Peng Qu, Yuan Xie, and Wenguang Chen "NEUTRAMS: Neural Network Transformation and Co-design under Neuromorphic Hardware Constraints." Proc. *International Symposium on Microarchitecture (MICRO)*, 2016.

[7]. **Shuangchen Li**, Cong Xu, Jishen Zhao, Lu Yu, Yuan Xie "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations on Emergying Non-volatile Memories." Proc. *the 53nd Annual Design Automation Conference (DAC)*, 2016.

[8]. **Shuangchen Li**, Liu Liu, Peng Gu, Cong Xu, and Yuan Xie "NVSim-CAM: A Circuit-Level Simulator for Emerging Nonvolatile Memory based Content-Addressable Memory." Proc. *International Comference on Computer-Aided Design (ICCAD)*, 2016.

[9]. Peng Gu, **Shuangchen Li**, Dylan Stow, Russell Barnes, Liu Liu, Yuan Xie, and Eren Kursun "Leveraging 3D Technologies for Hardware Security: Opportunities and Challenges." Proc. *Great Lakes Symposium on VLSI (GLVLSI)*, 2016.

[10]. Ping Chi, **Shuangchen Li**, Yuanqing Cheng, Yu Lu, Seung H. Kang, Yuan Xie. "Architecture Design with STT-RAM: Opportunities and Challenges." Proc. *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016. (Invited)

[11]. Kaisheng Ma, Yang Zheng, **Shuangchen Li**, Karthik Swaminathan, Xueqing Li, Yongpan Liu, John Sampson, Yuan Xie, and Vijaykrishnan Narayanan. "Architecture Exploration for Ambient Energy Harvesting Nonvolatile Processors." Proc. *International Symposium On High Performance Computer Architecture (HPCA)*, 2015. **(Best Paper Award, IEEE Micro Top Picks 2016)**

[12]. Kaisheng Ma, Xueqing Li, **Shuangchen Li**, Yongpan Liu, John Sampson, Yuan Xie, and Vijaykrishnan Narayanan. "Nonvolatile Processor Architecture Exploration For Energy-Harvesting Applications." *IEEE MICRO magazine*, 2015.

[13]. **Shuangchen Li**, Ping Chi, Jishen Zhao, K.T. Tim Cheng, and Yuan Xie. "Leveraging Nonvolatility for Architecture Design with Emerging NVM." Proc. *Non-Volatile Memory System and Applications Symposium (NVMSA)*, 2015. (Invited)

[14]. **Shuangchen Li**, Ang Li, Yuan Zhe, Yongpan Liu, Peng Li, Guanyu Sun, Yu Wang, Huazhong Yang, Yuan Xie. "Leveraging Emerging Nonvolatile Memory in High-Level Synthesis with Loop Transformations." Proc. *International Symposium on Low Power Electronics and Design (ISLPED)*, 2015.

[15]. Yongpan Liu, Zhewei Li, Hehe Li, Yiqun Wang, Xueqing Li, Kaisheng Ma, **Shuangchen Li**, Mei-Fang Chiang, John Sampson, Yuan Xie, Jiwu Shu, and Huazhong Yang. "Ambient Energy Harvesting Nonvolatile Processors: From Circuit To System." Proc. *the 52nd Annual Design Automation Conference (DAC)*, 2015. (Invited)

[16]. **Shuangchen Li**, Ang Li, Yongpan Liu, Yuan Xie, and Huazhong Yang. "Nonvolatile Memory Allocation and Hierarchy Optimization for High-Level Synthesis." Proc. *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2015.

Abstract

Memory-Centric Architectures: Bridging the Gap Between Compute and Memory

by

Shuangchen Li

While the compute part keeping scaling for decades, it becomes more and more difficult for the memory part to catch up. This mismatch raises two grand challenges. One is referred to as the "Memory Wall", in which case the memory latency and bandwidth turn to be the bottleneck, slowing down the system no matter how computing resource improves. The other one is referred to as the "Power Wall", which demands high power efficiency due to a limited power budget, whereas the energy spent on the memory accesses dominates the total energy consumption.

To address those challenges, this dissertation focuses on designing memory-centric architectures to bridge the gap between compute and memory. Two types of memory-centric architecture have been investigated. The first one is the compute-capable memory architecture, which moves computing resources to the memory side. The in-memory computing scheme explores larger bandwidth and reduces data movement overhead. The second one is the memory-rich accelerator architecture, which is designed with tightly coupled high performance computing resource and large-capacity on-die memory. The in-situ computing design provides benefits as a non *Von Neumann* architecture. This dissertation has proposed five architectures, which cover both compute-capable memory and memory-rich accelerator architectures, both the offshore DRAM and emerging non-volatile memory technologies, and a large range of the important applications, such as deep learning, database, graph processing.

Contents

Chapter 1

Introduction

For decades, computing capability has been scaling quite well. For example, the performance of world's top-1 supercomputer has improved 33× in the past decade [1], and the performance of a single GPGPU has even boosted 24× within three years[1]. The successes do not only owe to the technology development (Moore's Law), but also the innovations of computer architecture, such as super-scaler and multi-core. However, while the compute part is developing so fast, it is difficult for the memory part to catch up. For the memory bandwidth, it took JEDEC seven years to develop from DDR3 to DDR4, and after that (year 2014), the peak memory bandwidth of a single CPU [3] has only improved ∼15%. For the memory capacity, the DRAM DIMM capacity doubles every three years [4]. It is much slower than the computing scaling (about doubling every two years) [3] and heavily relies on the about-to-end Moore's Law. For the memory latency, it has only improved 26% during 11 years [5]. The unbalanced scaling capability between compute and memory turns into a grand gap.

To further illustrate the gap, Table 1.1 lists two-generation NVIDIA's GPUs' statistics as an example. From Pascal architecture to Volta [2], the computing performance has boosted 5.5× while the memory bandwidth improvement is only 1.2×; the computing resource improved

[1] Comparing NVIDIA K40 5Tops/s with V100 120Tops/s for FP16 performance [2].

40% but the memory capacity stays the same.

	Comp. *peak perf. (FP16)*	*Device* ***Mem.*** *Bandwidth*	***Comp.*** *transistors count*	*Device* ***Mem.*** *Capacity*
Pascal-2016	22Tops/s	732GB/s	15.3B	16GB
Volta-2017	120Tops/s	900GB/s	21.1B	16GB
	5.5× vs. 1.2×		1.4× vs. 1×	

Table 1.1: An example of two generations NVIDIA GPU [2] to show the gap between computing and memory.

1.1 Demands for Memory-Centric Architectures

While understanding the gap between the compute and memory, bridging the gap is severely demanded from both the hardware and application points of view, motivating us to focus more on the memory architecture design.

For the hardware, memory revolution is essential to achieve the exascale computing target, i.e., building a supercomputer that deliveries 10^{18}ops/s performance with 20GWatt power budget by 2023 [6]. It requires 11× improvement on performance and 8.3× improvement on energy efficiency in the next five years. There are two key challenges and both of them are caused by memory problems. The first one is referred to as the "memory wall", in which case the memory bandwidth turns to the system bottleneck, and the performance cannot be improved no matter how much the computing part improves. The second one is the "power wall", in which case energy efficiency must be improved before putting more compute resources due to a limited power budget. In this aspect, studies have showed accessing the data takes more than 2 orders of magnitude energy then the computing [7], leaving the memory as the bottleneck again.

For the applications, as we are entering the big data era, memory plays a more important role. Especially, some of the killer applications are memory bound. They require very simple

computing but feed on large memory bandwidth. For example, the graph processing applications could speedup ∼5× given ∼5× more memory bandwidth [8].

As a conclusion, all these demands inspire us to rethink the whole computer architecture design. After decades of mainly focusing on the computing part, now it is the time to shift our major focus to the memory part. On contrary to the conventional compute-centric architecture, all these demands are driving us towards the memory-centric architecture, which offers us many opportunities to bridge the gap between compute and memory.

1.2 Opportunities and Challenges

Towards the memory-centric design, there are two promising types of architectures, as shown in Figure 1.1.

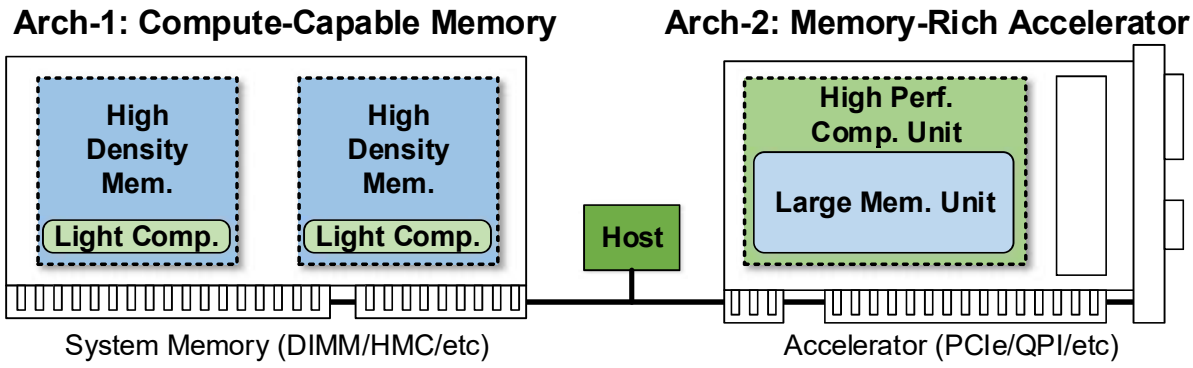

Figure 1.1: Two solutions: the compute-capable memory and the memory-rich accelerator.

Compute-Capable Memory. The first architecture (Figure 1.1(a)) is referred to as the compute-capable memory. This architecture designs lightweight computing units on the system main memory side. It is also known as Processing-in-memory (PIM) or In-memory computing architecture. The **opportunities** of this architecture is two folded. First, it can embrace the large memory internal bandwidth, which is otherwise wasted. In addition, the closer towards the memory cell, the larger bandwidth is offered. Figure 1.2(left side) shows the internal bandwidth potential of both DDR3 and HMC memory. From the DDR or HMC IO to the row buffer,

there is a 57× and 222× bandwidth improvement, respectively. Second, it eliminates the data movement between the host and the memory by performing the computing on the memory side. As shown in the right side of Figure 1.2, the energy spent on the data movement from the system memory (the last bar) is $\sim 100\times$ larger than that on computing, which highlights the importance of minimizing the data movement. It is reported that the compute-capable memory architecture can archive 80% energy saving on certain applications [8].

However, there are also **challenges**. First, memory industry is very cost-sensitive, so memory is highly optimized for low cost (density). Unfortunately, adding computing elements to inside the memory would incur larger area overhead than that could be accepted by the industry. Second, due to the cost-optimized design rule, it is difficult to design high performance computing units on the memory side. Even if 3D stacking technology is adopted, the empty area on the logic die where the memory-side processors locate are tightly limited [9]. For example, one of the HMC-based compute-capable memory reported 132Gops/s [10], whereas a state-of-the-art GPGPU offers 120Tops/s [2].

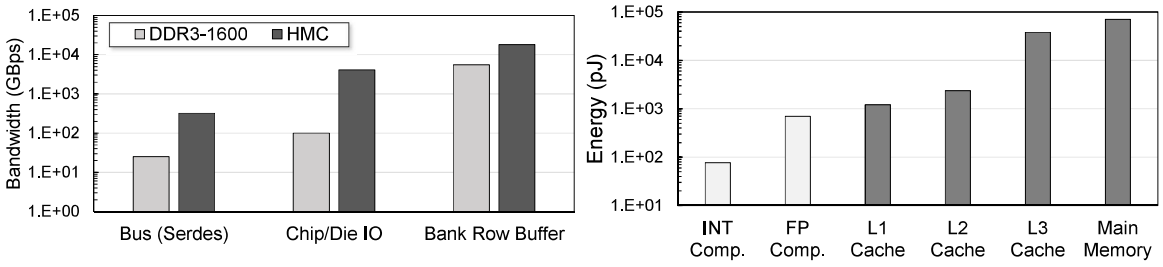

Figure 1.2: Left: Memory bus and internal bandwidth at chip/bank hierarchy for both DDR3 and HMC. Right: Data movement dominates the energy consumption (dynamic energy only with 32K-256K-4M-8G memory hierarchy at 45nm [11]).

Memory-Rich Accelerator. The second architecture (Figure 1.1(b)) is referred to as the memory-rich accelerator. It tries to merge the compute and memory by packing more on-die memories with the accelerator. The **opportunities** are obvious: accessing on-chip memory embraces larger bandwidth, lower latency, and smaller energy consumption. The benefits encourage many designs to adopt this architecture. For example, the neural network accelera-

tor Dadiannao [12] packs 36MB eDRAM scratchpad memory, Google's TPU packs a 24MB SRAM which consumes the same area as processing units [13], and POWER-9 [14] also packs 128MB eDRAM L3 cache.

However, the **challenges** are also noticeable. The capacity of the memory that can be packed on-die is limited. This is because to be compatible with CMOS technology, the on-die memory are either SRAM or eDRAM. Their memory cell area is large: SRAM cell is $\sim 146F^2$ [15] and eDRAM cell is $60F^2 \sim 80F^2$ [16], whereas the commodity DRAM is only $6F^2$. An exception is the interposer-based architecture, such as Volta GPU [2], which packs commodity DRAM with processors through links on the interposer. However, its bandwidth, latency, energy consumption are all not compatible with the on-die memory. For example, the on-die eDRAM in Dadiannao [12] offers 1.65TB/s bandwidth, 4.95ns latency, and 0.07pJ/bit [15] energy efficiency, whereas the interposer connected HBM in V100 [2] only has 0.9TB/s bandwidth, but requires ~200ns latency [17] and 3.7pJ/bit [18] energy.

1.3 Contributions

The **goal** of this dissertation is to explore both the compute-capable memory and memory-rich accelerator architecture, to bridge the gap between compute and memory. Meanwhile, we have addressed the challenges of both them, i.e., the tight area constraints for the compute-capable memories and the limited memory capacity for the memory-rich accelerator. To this end, the **key idea** of this dissertation is to leverage memory cell themselves for computing. Specifically, we have two compute-capable memory designs.

- PINATUBO, which accelerates the bulk bitwise operations inside the NVM-based main memory with minimal area overhead by leveraging the resistive feature of the memory cells.
- PRIME, a morphable processing-in-memory architecture for ReRAM-based main mem-

ory, which efficiently accelerates neural computation by leveraging ReRAMs unique yet largely overlooked property having both computation and data storage capability in one device.

We also have three memory-rich accelerator designs.

- NVCAM, which designs the accelerator with the content-addressable memory to serve searching tasking with high energy efficiency.
- DRISA, which is an in-situ accelerator built using DRAM technology with the majority of the area consisting of DRAM memory arrays, and computes with logic on every memory bitline, achieving the goal of large memory capacity for the accelerator.
- SCOPE, which combines the stochastic computing arithmetic with the DRISA's in-situ architecture, to further improve its performance.

The architectures that proposed in this dissertation have a board coverage in both terms of technologies and the applications: three of the designs adopts the emerging non-volatile memory technologies, and the other two designs leverage the offshore DRAM technology. In addition, these architectures have covered a large range of applications such as deep learning inference and training, graph processing, database, and bioinformatics.

To conclude, by exploiting the memory cells' special features, we leverage the memory for the computing tasks, and build better memory-centric architectures to further close the gap between compute and memory. The architectures proposed highlight their contributions to tackle the tight area constraints challenge for the compute-capable memory architecture and the limited memory capacity problem for the memory-rich accelerator architecture.

Chapter 2

Backgrounds and Related Work

In this chapter, we first introduce basic backgrounds about the system memory. Then, we survey the related work for both the compute-cable memory and the memory-rich accelerator architectures, with emphasize on demonstrate the distinguishing of the architecture proposed in this dissertation.

2.1 System Memory and DRAM Basics

Figure 2.1 shows the basic circuit of commercial DRAMs. A DRAM chip contains multiple banks, which are connected with global buses. Inside a bank, there are subarrays that share global BLs. Global decoders decode parts of the addresses to the global wordlines (WLs) that connected to different subarrays. Cell matrices (Mats) are basic memory arrays that line up and make a subarray. Every Mat has its private local WL decoders, drivers, and sense amplifiers (SAs). A DRAM cell is constructed with a access transistor and a capacitor (1T1C). Within one chip, only a single row in a subarray is activated at one time. The Mats in the subarray work in a lock-step manner.

A DRAM process and a logic process are totally different and incompatible [19]. It is

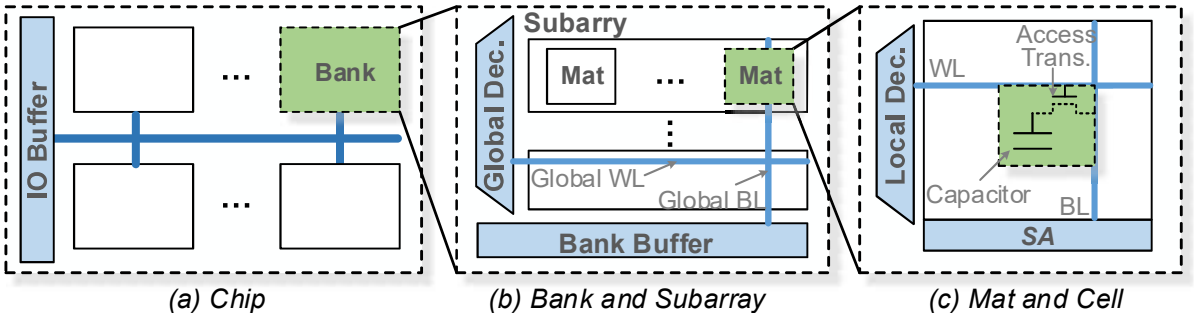

Figure 2.1: The DRAM basics. Glossary: WL - Wordline; BL - Bitline; SA - Sense Amplifier; Dec - Decoder.

difficult to build logics with a DRAM process, or the other way around. The transistors in a DRAM process are extremely optimized for low leakage, which draws their speed down. For example, substrate bias are added to increase threshold voltage for extra leakage saving. Moreover, a DRAM process usually has three metal layers but logic processes can get more than twelve metal layers, which means logic circuits in a DRAM process will suffer from higher interconnection overhead. As a summary, building complex logic circuits in a DRAM process is problematic with 22% performance degradation and 80% area overhead [19], which is a major reason why earlier PIM researches that put processors and DRAM on the same die haven't turned into a success. On the other hand, building DRAM cells with a logic process turns to be eDRAMs, which is not efficient, either. eDRAM results in 10x area overhead [16, 20], around 4× more power, and 100× shorter retention time [20], compared with DRAMs.

2.2 Emerging Nonvolatile Memory Basics

Device. Although the working mechanism and the features varies, the three typical types of the NVM have common basics: all of them are based on resistive cell. To represent logic "0" and "1", they rely on the difference of cell resistance (R_{high} or R_{low}), which is archived by different state of the device, i.e., high-resistance state (HRS) and low-resistance state (LRS). To

switch between logic "0" and "1", certain polarity, magnitude, and duration voltage is required to be applied on the cell.

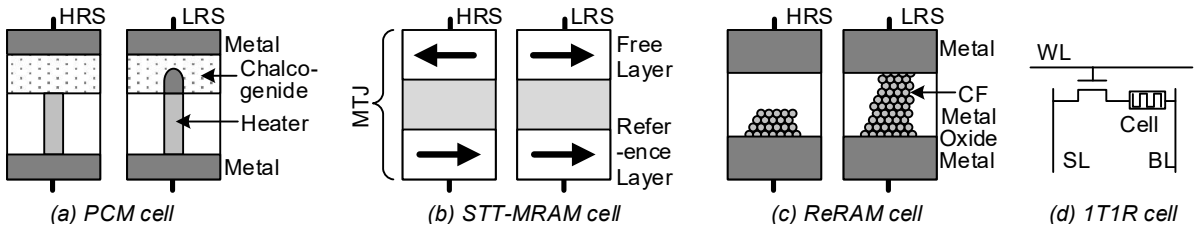

Figure 2.2: Device Basics for (a) PCM (b) STT-MRAM (c) ReRAM, and the (d) 1T1R Cell Structure.

PCM [21] uses phase change material (e.g., chalcogenide glass) to archive two device states. The amorphous state provides HRS and the crystalline state is the LRS. They represent logic "0" and "1" respectively. Switching between those two states requires different heating temperatures and durations, i.e., different write current. Specifically, changing from LRS to HRS is referred as a RESET operation, which requires a high and short current pulse. In contrast, switching from HRS to LRS is denoted as a SET operation, which requires a moderate and long current pulse.

STT-MRAM [22] relies on the magnetic tunnel junction (MTJ) with a fixed ferromagnetic layer (reference layer) and a programmable ferromagnetic layer (free layer). The magnetization direction of the free layer provides two different states. Parallel magnetization direction of the free and reference layer leads to LRS (logic "0"), and in contrast, the anti-parallel magnetization direction leads to HRS (logic "1"). To write logic "1", a negative voltage difference is required. To write logic "0", a positive voltage difference needs to be established.

ReRAM [23] sandwiches a metal-oxide layer with two metal layers. The nanoscale conductive filaments (CF) provides two states. A well established CF provides LRS (logic "1") and cut-off CF leads to HRS (logic "0"). When certain polarity voltage is required to switch between HRS and LRS.

Peripheral Circuitry. Because of the special feature of the memory cell device, e.g., resistive cell instead of charge-based cell, the peripheral circuitry demands special designs. *Cell*

structure of all the three technologies can implement the memory block with a 1T1R structure [24–26], where it has a wordline (WL) controlling the access transistor, a bitline (BL) for data sensing, and source line (SL) to sense to provide write current (for ReRAM and STT-MRAM in particular). In order to provide high density, 1S1R (1-selector-resistor) structure is proposed for both PCM [27] and ReRAM [28] with only $4F^2$ cell area. 3D-stacked memory-cell is also proposed [29, 30]. To further improve the density, multi-level cell (MLC) is proposed for all of those three NVM [31–33].

Sense amplifier (SA) is the major different between the NVM and conventional SRAM or DRAM. While conventional charge-based memory using a few transistor to amplify the charge difference, the resistance-based NVM has to convert the difference in resistance ($R_{\text{high}}/R_{\text{low}}$) as voltage or current, which are referred as voltage and current sensing (VSA and CSA), respectively. NVM prefers CSA [34] due to its advantages of fast access and robustness of variation. Besides, NVM requires extra reference cells for sensing [1]. Considering the large area of CSA and the extra reference cell, it is typical to share SA among adjacent columns, which leads to a smaller row buffer size [35].

Other difference besides SA in the peripheral circuitry also exits. The write driver (WD) [24, 30] is design to provide certain polarity, magnitude, and duration voltage or current across the BL and the SL. The support circuit for restore in DRAM is not necessary in NVM, thanks to its non-volatility and non-destructive read. Extra lift time enhancement circuit is probably required for PCM and ReRAM, e.g., flip-N-write [36] and read-before-write [37].

NVM-based system main memory. As the NVM technology turns mature, all the three technologies (PCM [37–39], ReRAM [40], and STT-MRAM [41–43]) are proposed as a candidate to be the next generation main memory, due to NVM's high density, better scalability, and ultra-low standby power. In order to tackle the limited lifetime problem of PCM and ReRAM, architecture-level solution such as wear-leveling [39] and comparison-based write [36, 37] are

[1] Although self-reference is possible in STT-MRAM, it incurs large latency/energy overhead.

proposed. To migrate the expensive write operation overhead, architecture supports such as write-cancelation [44] and data preset [45] are proposed.

2.3 Related Work on Compute-Capable Memories

Early Effort on Processing-in-Memory Architectures The idea of PIM can be traced back to the 1990s. A bulk of works, including IRAM [46], DIVA [47], Active Page [48], FlexRAM [49], proposed the PIM-based architecture by integrating computing logics and DRAM on one chip. We refer to these work as 2D-based PIM. Based on the observation that memory latency is 4x larger than raw DRAM access time, D. Patterson [46] proposed IRAM to merge the processing and DRAM on the same chip. By replacing the original cache hierarchy with DRAM on Alpha 21164 (the fastest processor at that time), they were able to achieve 1x-2x speedup on the applications of database and sparse computing. To further exploit the internal DRAM bandwidth, IRAM also explores the possibility of introducing PIM to vector processor, while the implementation of PIM on traditional 2D DRAM experiences great challenges in cost and manufacture. These work has been criticized for the difficulty of integrating complex logics with DRAM technologies. Even including simple adders in DRAM process technologies induces large overhead and low effective performance. However, recently, this approach has been revisited. Buffered Comparator [50, 51]) puts lightweight comparator with DRAMs. ProPRAM [52] leverages 2D NVM for PIM. It uses NVM's lifetime enhancement peripheral circuits (data-comparison write and flip-n-write) for in memory computing. Even thought the area overhead is partially solved by adding much less computing unit, the computing performance of that turns into a new bottleneck.

As a summary, PIM sticks with a main memory position that is optimized for memory density. Building logic elements inside the memory benefit a lot, but the area overhead makes a server limitation.

3D Stacking based Near Data Computing Architectures The emerging technology of 3D die stacking, either through-silicon-via (TSV) based 3D integration or interposer-based 2.5D integration, has alleviated the manufacture concerns of PIM to some extent, which is referred to as 3D-based PIM. The vertical 3D die-stacking with TSVs allows the stacking of multiple memory dies directly on top of the logic die to achieve high memory bandwidth; while the 2.5D silicon interposer allows a high bandwidth connection between the memory stack and the logic die on the same substrate. There are two main streams for the 3D-stacking memory: Hybrid Memory Cube (HMC) and High Bandwidth Memory (HBM) [8, 10, 53–66]. Since the power and area budget is quite limited in the logic die, the processor in HMC prefers to be light-weight, which might not support complex functions, such as branch prediction, dynamic scheduling, and cache hierarchy. IBM [67] proposed Active Memory Cube (AMC), which adds computational elements on the base logic layer to accelerate scientific computing applications. They designed the base logic layer for efficiency and versatility and came up with the following design choices: no caches, hardware scheduling of instructions, a vector instruction set, predicated execution, and gather-scatter accesses directly to memory in the Cube. Several attempts have also been made on HMC-based energy efficient domain specific accelerators. Guo *et al.* [68] proposes to integrate various accelerators into the base logic layer to obtain an energy efficient accelerated library for memory bounded operations, e.g. matrix operation, data reshaping, and FFT. Gao *et al.* [69] proposed Tetris, an HMC-based neural network accelerator, which moves the logic of the accumulation operation to the DRAM die, or further into the bank, to reduce the double access of the intermediate results in the neural network applications.

As a summary, the 3D-stacking based PIM architectures are more practical after that researches have taken the lessons from the early 2D-based PIM effort. It revives the PIM research with the help of the severer-than-ever demands of bridging the gap between compute and memory. However, the empty area on the logic die is limited [9], not to mention the fixed power

budget and thermal constraints for the system memory. As a result, the computing performance of the 3D-based PIM is limited. In addition, as shown in Figure 1.2, the closer the computing units are designed to the data, the more benefit we can have from the compute-capable memory architecture. Though on the memory side, the computing units and the memory are still in different dies in the 3D-based PIM. It might not embrace as much as the benefit comparing with the 2D-based PIM.

Revolutionary DRAM Designs Recent research has evolved DRAM memory by adding additional functional capability [70]. Rowclone [71] has supported in-DRAM row-to-row copy with minor modification on existing DRAMs. Seshadri *et al.* [72] has explored fast bulk bitwise operations in DRAMs. However, it only support AND/OR operations, which are not logic complete, and hence not for general purpose. DRAF [73] has proposed a DRAM-based FPGA, where DRAM cells are used as look-up tables.

As a summary, the revolutionary DRAM is more like a light version of the 2D-based PIM. To address the cost-sensitive challenge, they come up with smart ideas to minimize the area overhead. However, this come with the drawback of low computing performance, or limited support for the computing operation types.

2.4 Related Work on Memory-Rich Accelerators

Memory-Rich Processors Interposer-based 2.5D integration is already embraced by the industry for integrating 3D stacked memories with large-scale designs, due to its feasibility over true 3D integration in terms of cost and better thermal profile. For example, AMD's Fury X GPU integrates 4GB of 3D stacked High-Bandwidth Memory (HBM) [74]; Nvidia's Pascal/Volta GPU increases the capacity of HBM to 16GB [75]; Intel's Knights' Landing CPU also integrates 16GB high bandwidth stacked DRAM [76]. Xilinx recently also announced the Virtex Ultrascale+ FPGA that integrates 8GB HBM [77]. Lots of designs that accelerate NN appli-

cations with various platforms (ASIC, GPU, FPGA) with large on-chip memory [78–80] have been proposed. TrueNorth [81–84] computes with on-chip SRAM-based crossbars and counters. DaDianNao [78] has 36MB of on-chip memories, but *DRISA* has 512MB. NeuroCube [10] is a PIM architecture for NNs. Though its memory capacity is large, its performance is low.

As a summary, as mentioned in the introduction, the interposer connected memories, although having large memory, the bandwidth, latency, and energy efficiency are not as good as the on-die memories. On the other hand, the on-die memory's density is low and the memory capacity packed is not large enough.

In-situ Computing Accelerators Automata [85] has implemented a reconfigurable processor with a DRAM process. It computes with counters and finite-state machines, by storing programmed states in the DRAM while streaming in the data. Mikamonu [86–88] has proposed to compute with the `NOR` logic provided by the 3T1C DRAM cell or NVMs, but their proposal does not have data movement mechanisms. Therefore, complex functions (like full adders) are not supported. NVM is very suitable for associate memory implementation [89–93]. Researches have use the NVM-based TCAM in CPU [94], GPGPU [95], and accelerators [96,97]. NVM is also designed as LUT [98] and nonvolatile logic gates [99]. Recent work also take use of ReRAM crossbar's special feature, to implement IMPLY-based logics [100–102]. Also taking use of the crossbar structure, researcher are able to implement dot-product function with high energy efficiency, which is widely used in deep learning acceleration [103, 104].

Chapter 3

PINATUBO: A Processing in Emerging Non-volatile Memory Architecture for Bulk Bitwise Operations

*The **goal** of this chapter is to show Non-volatile memory (NVM)'s potential on enabling PIM architecture, while almost all existing efforts focus on DRAM systems and heavily depend on 3D integration.* NVM's unique features, such as resistance-based storage (in contrast to charge-based in DRAM) and current-sensing scheme (in contrast to the voltage-sense scheme used in DRAM), are able to provide inherent computing capabilities [101, 105]. Therefore, NVM can enable PIM without the requirement of 3D integration. In addition, it only requires insignificant modifications to the peripheral circuitry, resulting in a cost-efficient solution. Furthermore, NVM-enabled PIM computation is based on in-memory analog signals, which is much more energy efficient than other work that uses digital circuits.

In this chapter, we propose PINATUBO[1], a Processing In Non-volatile memory ArchiTecture

[1]Mount Pinatubo is an active volcano that erupted in 1991. We name our PIM design after Pinatubo, because PIM researches also had a great moment in 1990s, and we believe it is their time to revive. The novel architecture's impact will be like a volcano eruption.

for bUlk Bitwise Operations, including OR, AND, XOR, and INV operations. When PINATUBO works, two or more rows are activated simultaneously, the memory will output the bitwise operations result of the open rows. PINATUBO works by activating two (or more) rows simultaneously, and then output of the memory is the bitwise operation result upon the open rows. The results can be sent to the I/O bus or written back to another memory row directly. The major modifications on the NVM-based memory are in the sense amplifier (SA) design. Different from a normal memory read operation, where the SA just differentiates the resistance on the bitline between R_{high} and R_{low}, PINATUBO adds more reference circuit to the SA, so that it is capable of distinguishing the resistance of $\{R_{\text{high}}/2$ (logic "0,0"), $R_{\text{high}}||R_{\text{low}}$ (logic "0,1"), $R_{\text{low}}/2$ (logic "1,1")$\}$ for 2-row AND/OR operations. It also potentially supports multi-row OR operations when high ON/OFF ratio memory cells are provided. Although we use 1T1R PCM as an example in this paper, PINATUBO does not rely on a certain NVM technology or cell structure, as long as the technology is based on resistive-cell.

Our contributions in this chapter are listed as follows,

- We propose a low-cost processing-in-NVM architecture with insignificant circuit modification and no requirement on 3D integration.
- We design a software/hardware interface which is both visible to the programmer and the hardware.
- We evaluate our proposed architecture on data intensive graph processing and data-base applications, and compare our work with SIMD processor, accelerator-in-memory PIM, and the state-of-the-art in-DRAM computing approach.

3.1 Motivation and Overview

Bitwise operations are very important and widely used by database [106], graph processing [107], bio-informatics [108], and image processing [109]. They are applied to replace

expensive arithmetic operations. Actually, modern processors have already been aware of this strong demand, and developed accelerating solutions, such as Intel's SIMD solution SSE/AVX.

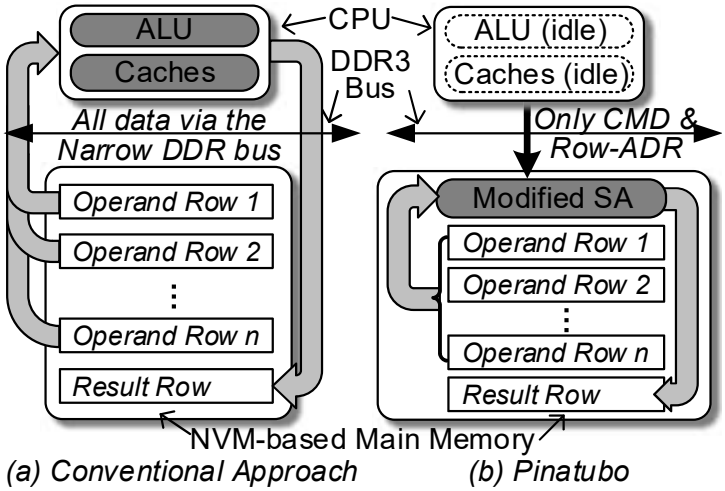

Figure 3.1: Overview: (a) Computing-centric approach, moving tons of data to CPU and write back. (b) The proposed PINATUBO architecture, performs *n*-row bitwise operations inside NVM in one step.

We propose PINATUBO to accelerate the bitwise operations inside the NVM-based main memory. Figure 3.1 shows the overview of our design. Conventional computing-centric architecture in Figure 3.1 (a) fetches every bit-vector from the memory *sequentially*. The data walks through the narrow DDR bus and all the memory hierarchies, and finally is executed by the limited ALUs in the cores. Even worse, it then needs to write the result back to the memory, suffering from the data movements overhead again. PINATUBO in Figure 3.1 (b) performs the bit-vector operations inside the memory. Only commands and addresses are required on the DDR bus, while all the data remains inside the memory. To perform bitwise operations, PINATUBO activates two (or more) memory rows that store bit-vector simultaneously. The modified SA outputs the desired result. Thanks to in-memory calculation, the result does not need the memory bus anymore. It is then written to the destination address thought the WD directly, bypassing all the I/O and bus.

PINATUBO embraces two major benefits from PIM architecture. First, the reduction of data movements. Second, the large internal bandwidth and massive parallelism. PINATUBO

performs a memory-row-length (typical 4Kb for NVM) bit-vector operations. Furthermore, it supports multi-row operations, which calculate multi-operand operations in one step, bringing the equivalent bandwidth $\sim$1000$\times$ larger than the DDR3 bus.

3.2 Architecture and Circuit Design

In this section, we first show the architecture design that enables the NVM main memory for PIM. Then we show the circuit modifications for the SA, LWL driver, WD, and global buffers.

3.2.1 From Main Memory to PINATUBO

Main memory has several physical/logic hierarchies. *Channels* runs in parallel, and each channel contains several *ranks* that share the address/data bus. Each rank has typical 8 physical *chips*, and each chip has typical 8 *banks* as shown in Figure 3.2 (a). Banks in the same chip share the I/O, and banks in different chips work in a lock-step manner. Each bank has several *subarrays*. As Figure 3.2 (b) shows, Subarrays share the GDLs and the global row buffer. One subarray contains several *MATs* as shown in Figure 3.2 (c), which also work in the lock-step manner. Each Mat has its private SAs and WDs. Since NVM's SA is much larger than DRAM, several (32 in our experiment) adjacent columns share one SA by a MUX.

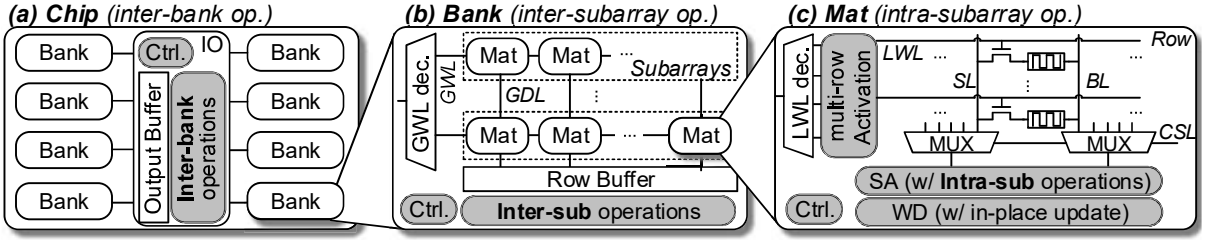

Figure 3.2: The PINATUBO Architecture. Glossary: Global WordLine (GWL), Global DataLine (GDL), Local WordLine (LWL), SelectLine (SL), BitLine (BL), Column Select-Line (CSL), Sense Amplifier (SA), Write Driver (WD).

According to the physical address of the operand rows, PINATUBO performs three types of bitwise operations: intra-subarray, inter-subarray, and inter-bank operations.

Intra-subarray operations. If the operand rows are all within one subarray, PINATUBO performs intra-subarray operations in each MAT of this subarray. As shown in Figure 3.2 (c), the computation is done by the modified SA. Multiple rows are activated simultaneously, and the output of the modified SA is the operation result. The operation commands (e.g., AND or OR) are sent by the controller, which change the reference circuit of the SA. We also modify the LWL driver is also implemented to support multi-row activation. If the operation result is required to write back to the same subarray, it is directly fed into the WDs locally as an in-place update.

Inter-subarray operations. If the operand rows are in different subarrays but in the same bank, PINATUBO performs inter-subarray operations as shown in Figure 3.2 (b). It is based on the digital circuits added on the global row buffer. The first operand row is read to the global row buffer, while the second operand row is read onto the GDL. Then the two operands are calculated by the add-on logic. The final result is latched in the global row buffer.

Inter-bank operations. If the operand rows are even in different banks but still in the same chip, PINATUBO performs inter-bank operations as shown in Figure 3.2 (a). They are done by the add-on logic in the I/O buffer, and have a similar mechanism as inter-subarray operations.

Note that PINATUBO does not deal with operations between bit-vectors that are either in the same row or in different chips. Those operations could be avoided by optimized memory mapping, as shown in Section 3.3.

3.2.2 Peripheral Circuitry Modification

SA Modification: The key idea of PINATUBO is to use SA for intra-subarray bitwise operations. The working mechanism of SA is shown in Figure 3.3. Different from the charge-based

DRAM/SRAM, the SA for NVM senses the resistance on the BL. Figure 3.3 shows the BL resistance distribution during read and OR operations, as well as the reference value assignment. Figure 3.3 (a) shows the sensing mechanism for normal reading (Though the SA actually senses currents, the figure presents distribution of resistance for simplicity). The resistance of a single cell (either R_{low} or R_{high}) is compared with the reference value ($R_{\text{ref-read}}$), determining the result between "0" and "1". For bitwise operations, an example for a 2-row OR operation is shown in Figure 3.3 (b). Since two rows are activated simultaneously, the resistance on the BL is the parallel connection of two cells. There could be three situations: $R_{\text{low}}||R_{\text{low}}$ (logic "1","1"), $R_{\text{low}}||R_{\text{high}}$ ("1","0"), and $R_{\text{high}}||R_{\text{high}}$ ("0","0")[2]. In order to perform OR operations, the SA should output "1" for the first two situations and output "0" for the last situation. To achieve this, we simply shift the reference value to the middle of $R_{\text{low}}||R_{\text{high}}$ and $R_{\text{high}}||R_{\text{high}}$, denoted as $R_{\text{ref-or}}$. Note that we assume the variation is well controlled so that no overlap exists between "1" and "0" region. In summary, to compute AND and OR, we only need to change the reference value of the SA.

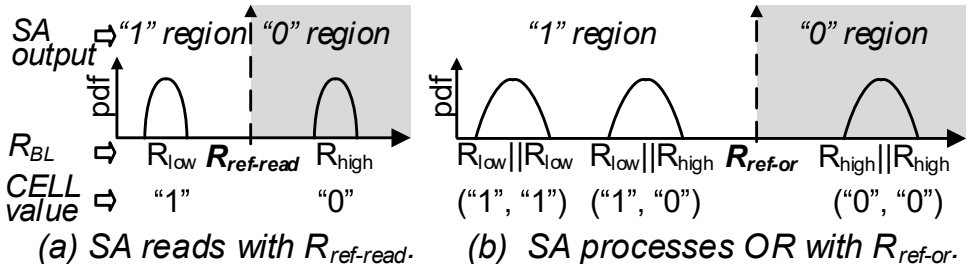

Figure 3.3: Modifying Reference Values in SA to Enable PINATUBO.

Figure 3.4 (a) shows the corresponding circuit modification based on the CSA [110]. As explained above, we add two more reference circuits to support AND/OR operations. For XOR, we need two micro-steps. First, one operand is read to the capacitor C_h. Second, the other operand is read to the latch. The output of the two add-on transistors is the XOR result. For INV, we simply output the differential value from the latch. The output is selected among

[2] "||" denotes production over sum operation.

READ, AND, OR, XOR, and INV results by a MUX. Figure 3.4 (b) shows the HSPICE validation of the proposed circuit. The circuit is tested with a large range of cell resistances from the recent PCM, STT-MRAM, and ReRAM prototypes [111].

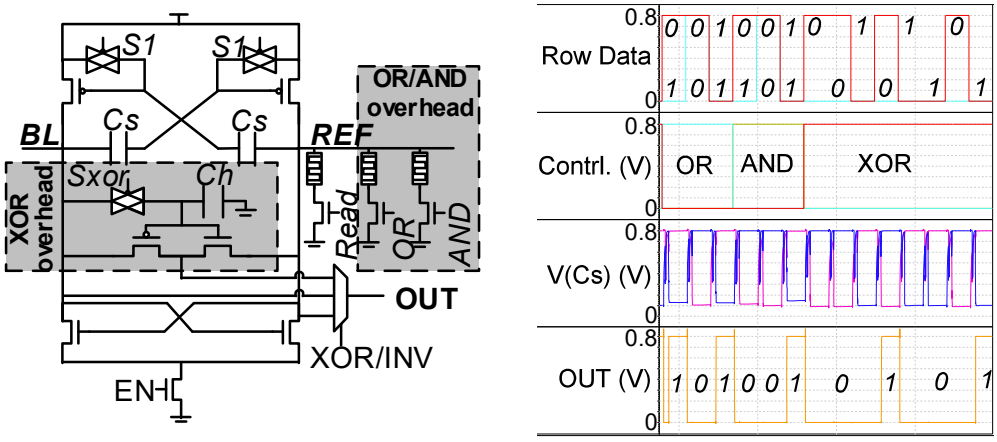

Figure 3.4: Current Sense Amplifier (CSA) Modification (left) and HSPICE Validation (right).

Multi-row Operations: PINATUBO supports multi-row operations that further accelerate the bitwise operations. A multi-row operation is defined as calculating the result of multiple operands at one operation. For PCM and ReRAM which encode R_{high} as logic "0", PINATUBO can calculate n-row OR operations[3]. After activating n rows simultaneously, PINATUBO needs to differentiate the bit combination of only one "1" that results in "1", and the bit combination with all "0" that results in "0". This leads to a reference value between $R_{\text{low}}||R_{\text{high}}/(n-1)$ and R_{high}/n. This sensing margin is similar with the TCAM design [112]. State-of-the-art PCM-based TCAM supports 64-bit WL with two cells per bit. Therefore we assume maximal 128-row operations for PCM. For STT-MRAM, since the ON/OFF ratio is already low, we conservatively assume maximal 2-row operation.

LWL Driver Modification: Conventional memory activates one row each time. However, PINATUBO requires multi-row activation, and each activation is a random-access. The modifications of the LWL driver circuit and the HPSICE validation are shown in Figure 3.5. Nor-

[3]Multi-row AND in PCM/ReRAM is not supported, since it is unlikely to differentiate $R_{\text{low}}/(n-1)||R_{\text{high}}$ and R_{low}/n, when $n > 2$.

mally, the LWL driver amplifies the decoded address signal with a group of inverters. We modify each LWL drive by adding two more transistors. The first transistor is used to feed the signal between inverters back and serves as a latch. The second transistor is used to force the driver's input as ground. During the multi-row activation, it requires to send out the RESET signal first, making sure that no WL has latched anything. Then every time an address is decoded, the selected WL signal is latched and stuck at VDD until the next RESET signal arrives. Therefore, after issuing all the addresses, all the corresponding selected WL are driven to the high voltage value.

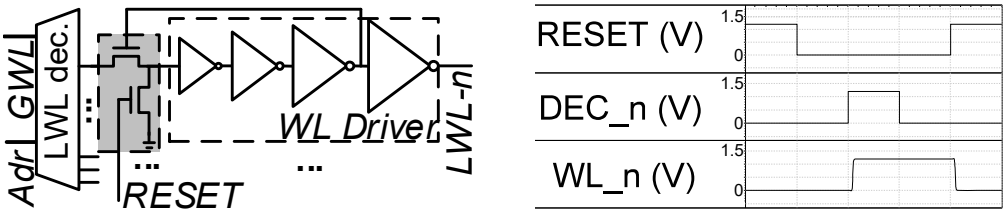

Figure 3.5: Local Wordline (LWL) Driver Modification (left) and HSPICE Validation (right).

WD Modification: Figure 3.6 (a) shows the modification to a WD of STT-MRAM/ReRAM. We do not show PCM's WD since it is simpler with unidirectional write current. The write current/voltage is set on BL or SL according to the write input data. Normally, the WD's input comes from the data bus. We modify the WD circuit so that the SA result is able to be fed directly to the WD. This circuit bypasses the bus overhead when writing results back to the memory.

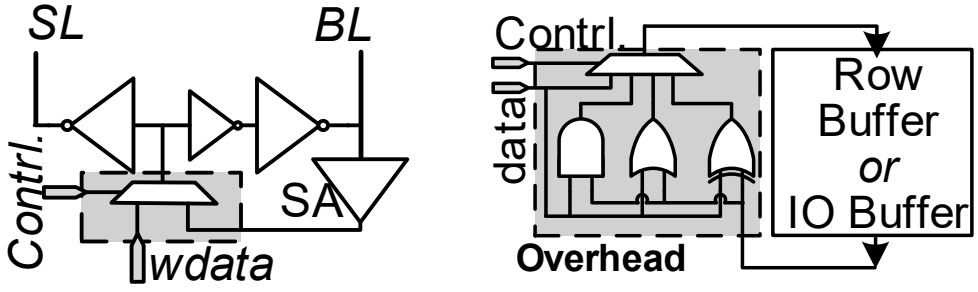

Figure 3.6: (a) Modifications to Write Driver (WD). (b) Modifications for Inter-Sub/Bank Operations

Global Buffers Modification: To support inter-subarray and inter-bank operations, we have to add the digital circuits to the row buffers or IO buffers. The logic circuit's input is the data

from the data bus and the buffer. The output is selected by the control signals and then latched in the buffer, as shown in Figure 3.6 (b).

3.3 System Support

Figure 3.7 shows an overview of PINATUBO's system design. The software support contains the programming model and run-time supports. The programming model provides two functions for programmers, including the bit-vector allocation and the bitwise operations. The run-time supports include modifications of the C/C++ run-time library and the OS, as well as the development of the dynamic linked driver library. The C/C++ run-time library is modified to provide a PIM-aware data allocation function. It ensures that different bit-vectors are allocated to different memory rows, since PINATUBO is only able to process inter-row operations. The OS provides the PIM-aware memory management that maximizes the opportunity for calling intra-subarray operations. The OS also provides the bit-vector mapping information and physical addresses (PAs) to the PIM's run-time driver library. Based on the PAs, the dynamic linked driver library first optimizes and reschedules the operation requests, and then issues extended instruction for PIM [113]. The hardware control part utilizes the DDR mode register (MR) and command. The extended instructions are translated to DDR commands and issued through the DDR bus to the main memory. The MR in the main memory is set to configure the PIM operations.

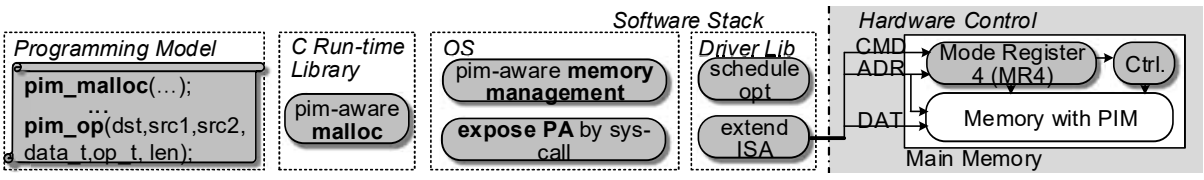

Figure 3.7: PINATUBO System Support.

3.4 Experiment

In this section, we compare PINATUBO with state-of-the-art solutions and present the performance and energy results.

3.4.1 Experiment Setup

The three counterparts we compare are described below:

SIMD is a 4-core 4-issue out-of-order x86 Haswell processor running at 3.3GHz. It also contains a 128-bit SIMD unit with SSE/AVX for bitwise operation acceleration. The cache hierarchy consists of 32KB L1, 256KB L2, and 6MB L3 caches.

S-DRAM is the in-DRAM computation solution to accelerate bitwise operations [114]. The operations are executed by charges sharing in DRAM. Due to the read-destructive feature of DRAM, this solution requires copying data before calculation. Only 2-row AND and OR are supported.

AC-PIM is an accelerator-in-memory solution, where even the intra-subarray operations are implemented with digital logic gates as shown in Figure 3.6 (b).

The *S-DRAM* works with a 65nm 4-channel DDR3-1600 DRAM. *AC-PIM* and *PINATUBO* work on 1T1R-PCM based main memory whose tRCD-tCL-tWR is 18.3-8.9-151.1ns [115]. *SIMD* works with DRAM when compared with *S-DRAM*, and with PCM when compared with *AC-PIM* and *PINATUBO*. Note that the experiment takes 1T1R PCM for a case study, but PINATUBO is also capable to work with other technologies and cell structures.

The parameters for *S-DRAM* are scaled from existing work [114]. The parameters for *AC-PIM* are collected from synthesis tool with 65nm technology. As to parameters for PINATUBO, the analog/mixsignal part, including SA, WD, and LWL, is extracted from HSPICE simulation; the digital part, including controllers and logics for inter-subarray/bank operations, is extracted from the synthesis tool. Based on those low-level parameters, we heavily modify NVsim [118]

Vector:	pure vector OR operations.
dataset:	*e.g. 19-16-1(s/r) means 2^{19}-lentgh vector, 2^{16} vectors, 2^{1}-row OR ops (sequntial/random access)*
Graph:	bitmap-based BFS for graph processing [107].
dataset:	*dblp-2010, eswiki-2013,amazon-2008 [116]*
Database:	bitmap-based database (Fastbit [106]) application.
dataset:	*240/480/720 number of quraying on STAR [117]*

Table 3.1: Benchmarks and Data Set

for the NVM circuit modeling, and CACTI-3DD [115] for the main memory modeling, in order to achieve high-level parameters. We also modify the PIN-based simulator Sniper [119] for SIMD processor and the NVM-based memory system. We develop an in-house simulator to evaluate the *AC-PIM*, *S-DRAM*, and PINATUBO. We show the evaluation benchmarks and data sets in Table 3.1, in which *Vector* only has OR operation while *Graph* and *Database* contain all AND, OR, XOR, and INV operations.

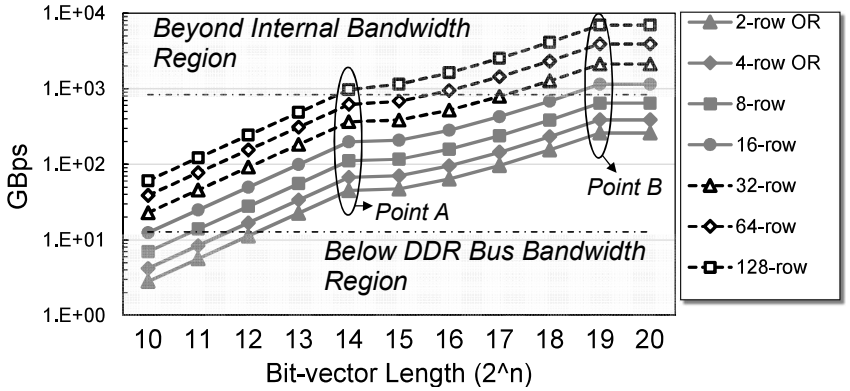

Figure 3.8: PINATUBO's Throughput (GBps) for OR operations.

3.4.2 Performance and Energy Evaluation

Figure 3.8 shows PINATUBO's OR operation throughput. We have four observations. First, the throughput increases with longer bit-vectors, because they make better use of the

memory internal bandwidth and parallelism. Second, we observe two turning points, *A* and *B*, after which the speedup improvement is showed down. *Turning point A* is caused by the sharing SA in NVM: bit-vectors longer than 2^{14} have to be mapped to columns the SA sharing, and each part has to be processed in serial. *Turning point B* is caused by the limitation of the row length: bit-vectors longer than 2^{19} have to be mapped to multiple ranks that work in serial. Third, PINATUBO has the capability of multi-row operations (as the legends show). For *n*-row OR operations, larger *n* provides larger bandwidth. Fourth, the y-axis is divided into three regions: the below DDR bus bandwidth region which only includes short bit-vectors' result; the memory internal bandwidth region which includes the majority of the results; and the beyond internal bandwidth region, thanks to the multi-row operations. DRAM systems can never achieve beyond memory internal bandwidth region.

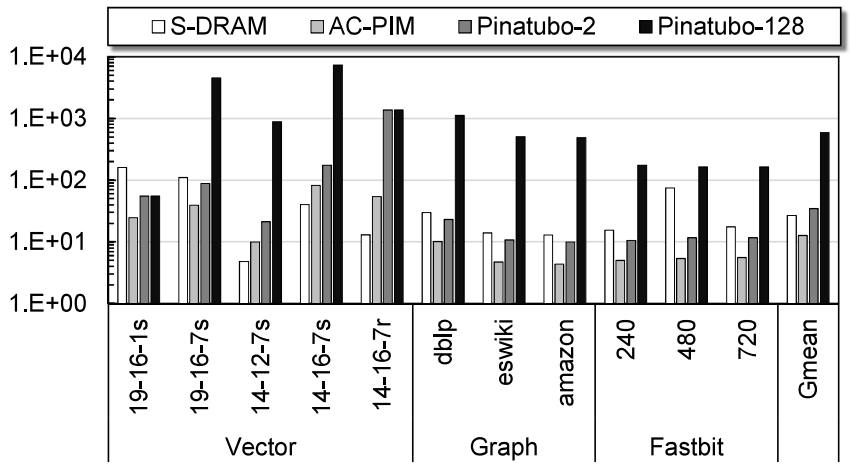

Figure 3.9: Speedup Normalized to *SIMD* Baseline.

We compare both PINATUBO of 2-row and 128-row operation with two aggressive baselines in Figure 3.9, which shows the speedup on bitwise operations. We have three observations: First, *S-DRAM* has better performance than *PINATUBO-2* in some cases with very long bit-vectors. This is because DRAM-based solutions benefit from larger row buffers, compared with the NVM-based solution. However, the advantage of NVM's multi-row operations still dominates. *PINATUBO-128* is 22× faster than *S-DRAM*. Second, the *AC-PIM* solution

is much slower than *PINATUBO* in every single case. Third, multi-row operations show their superiority, especially when intra-subarray operations are dominating. An opposite example is *14-16-7r*, where all operations are random accesses and it is dominated by inter-subarray/bank operations, so that *PINATUBO-128* is as slow as *PINATUBO-2*.

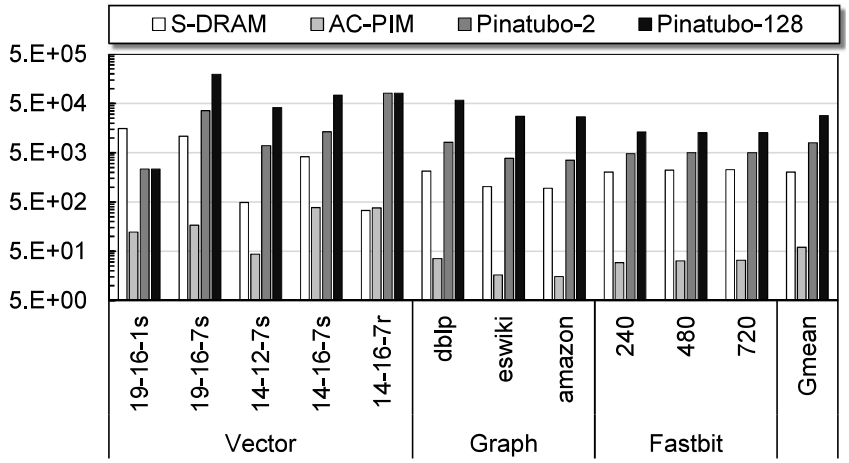

Figure 3.10: Energy Saving Normalized to *SIMD*.

Figure 3.10 shows the energy saving result. The observations are similar with those from speedup: *S-DRAM* is better than *PINATUBO-2* in some cases but worse than *PINATUBO-128* on average. *AC-PIM* never has a change to save more energy then any of the other three solutions, since both *S-DRAM* and *PINATUBO* rely on high energy efficient analogy computing. On average, PINATUBO saves 2800× energy for bitwise operations, compared with SIMD processor.

Figure 3.11 shows the overall speedup and energy saving of PINATUBO in the two real-world applications. The *ideal* legend represents the result with zero latency and energy spent on bitwise operations. We have three observations. First, PINATUBO almost achieves the ideal acceleration. Second, limited by the bitwise operations' proportion, PINATUBO can improve graph processing applications by 1.15× with 1.14× energy saving. However, it is data dependent. For the *eswiki* and *amazon* data set, since the connection is "loose", it has to spend most of the time searching for an unvisited bit-vector. For *dblp*, it has 1.37× speedup.

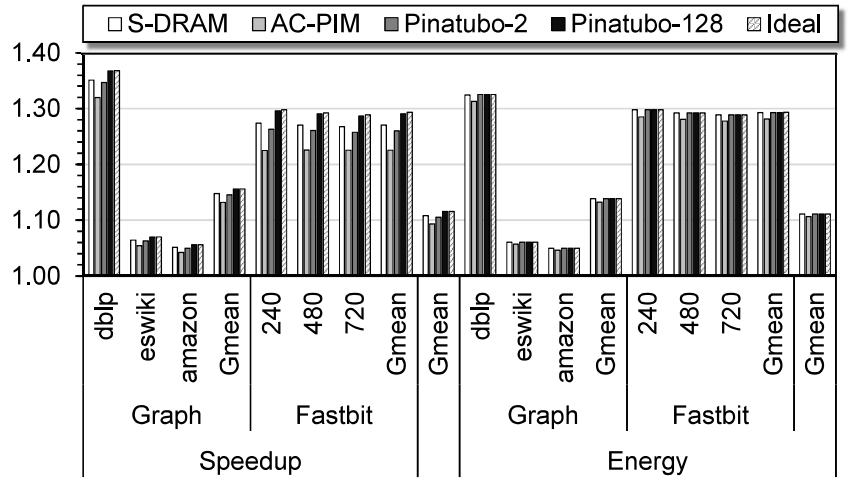

Figure 3.11: Overall Speedup and Energy Saving Normalized to *SIMD* Baseline.

Third, for the database applications, it achieves 1.29× overall speedup and energy saving.

3.4.3 Overhead Evaluation

Figure 3.12 shows the area overhead results. As shown in Figure 3.12 (a), PINATUBO incurs insignificant area overhead only 0.9%. However, *AC-PIM* has 6.4% area overhead, which is critical to the cost-sensitive memory industry. *S-DRAM* reports ∼0.5% capacity loss, but it is for DRAM-only result and orthogonal with PINATUBO's overhead evaluation. Figure 3.12 (b) shows the area overhead breakdown. We conclude that the majority area overhead are taken by inter-subarray/bank operations. For intra-subarray operations, XOR operations takes most of the area.

3.5 Summary

In this chapter, a processing-in-NVM architecture for bulk bitwise operations is proposed. The computation makes use of NVM's resistive-cell feature and achieves high performance

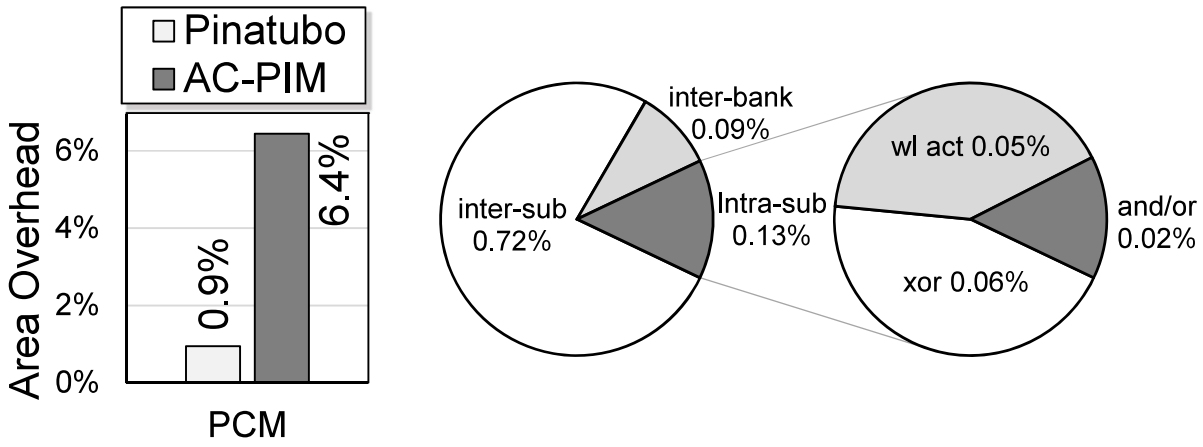

Figure 3.12: Area Overhead Comparison (left) and Breakdown (right).

and energy efficiency with insignificant area overheads. Experimental results show that the proposed architecture achieves $\sim$500$\times$ speedup and $\sim$28000$\times$ energy saving on bitwise operations, and 1.12$\times$ overall speedup, 1.11$\times$ overall energy saving on data intensive graph processing and database applications with real-world data.

Chapter 4

PRIME: Processing In ReRAM-based Main Memory

In this chapter, we propose a novel PIM architecture for efficient NN computation built upon ReRAM crossbar arrays, called PRIME, processing in ReRAM-based main memory. ReRAM has been proposed as an alternative to build the next-generation main memory [120], and is also a good candidate for PIM thanks to its large capacity, fast read speed, and computation capability. In our design, a portion of memory arrays are enabled to serve as NN accelerators besides normal memory. Our circuit, architecture, and software interface designs allow these ReRAM arrays to dynamically reconfigure between memory and accelerators, and also to represent various NNs. The current PRIME design supports large-scale MLPs and CNNs, which can produce the state-of-the-art performance on varieties of NN applications, e.g. top classification accuracy for image recognition tasks. Distinguished from all prior work on NN acceleration, PRIME can benefit from both the efficiency of using ReRAM for NN computation and the efficiency of the PIM architecture to reduce the data movement overhead, and therefore can achieve significant performance gain and energy saving. As no dedicated processor is required, PRIME incurs very small area overhead. It is also manufacture friendly

with low cost, since it remains as the memory design without requirement for complex logic integration or 3D stacking.

The contribution of this paper is summarized as follows:

- We propose a ReRAM main memory architecture, which contains a portion of memory arrays (full function subarrays) that can be configured as NN accelerators or as normal memory on demand. It is a novel PIM solution to accelerate NN applications, which enjoys the advantage of in-memory data movement, and also the efficiency of ReRAM based computation.
- We design a set of circuits and microarchitecture to enable the NN computation in memory, and achieve the goal of low area overhead by careful design, e.g. reusing the peripheral circuits for both memory and computation functions.
- With practical assumptions of the technologies of using ReRAM crossbar arrays for NN computation, we propose an input and synapse composing scheme to overcome the precision challenge.
- We develop a software/hardware interface that allows software developers to configure the full function subarrays to implement various NNs. We optimize NN mapping during compile time, and exploit the bank-level parallelism of ReRAM memory for further acceleration.

4.1 PRIME Architecture

We propose processing in ReRAM-based main memory, PRIME, which efficiently accelerates NN computation by leveraging ReRAM's computation capability and the PIM architecture. Figure 4.1(c) depicts an overview of our design. While most previous NN acceleration approaches require additional processing units (PU) (Figure 4.1(a) and (b)), PRIME directly leverages ReRAM cells to perform computation without the need for extra PUs. To achieve

this, as shown in Figure 4.1(c), PRIME partitions a ReRAM bank into three regions: memory (Mem) subarrays, full function (FF) subarrays, and Buffer subarrays.

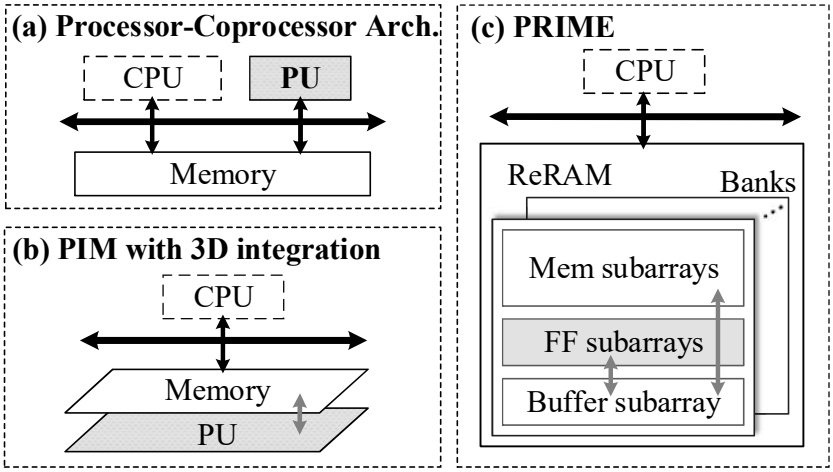

Figure 4.1: (a) Traditional shared memory based processor-coprocessor architecture, (b) PIM approach using 3D integration technologies, (c) PRIME design.

The Mem subarrays only have data storage capability (the same as conventional memory subarrays). Their microarchitecture and circuit designs are similar to a recent design of performance-optimized ReRAM main memory [120]. The FF subarrays have both computation and data storage capabilities, and they can operate in two modes. In memory mode, the FF subarrays serve as conventional memory; in computation mode, they can execute NN computation. There is a PRIME controller to control the operation and the reconfiguration of the FF subarrays. The Buffer subarrays serve as data buffers for the FF subarrays, and we use the memory subarrays that are closest to the FF subarrays as Buffer subarrays. They are connected to the FF subarrays through private data ports, so that buffer accesses do not consume the bandwidth of the Mem subarrays. While not being used as data buffers, the Buffer subarrays can also be used as normal memory. From Figure 4.1(c), we can find that for NN computation the FF subarrays enjoy the high bandwidth of in-memory data movement, and can work in parallel with CPU, with the help of the Buffer subarrays.

This section describes the details of our microarchitecture and circuit designs of the FF

subarrays, the Buffer subarrays, and the PRIME controller. These designs are independent of the technology assumptions for ReRAM based computation. For generality, we assume that the input data have P_{in} bits, the synaptic weights have P_w bits, and the output data have P_o bits. With practical assumptions, the precision of ReRAM based NN computation is a critical challenge. We discuss the precision issue and propose a scheme to overcome it in Section 4.1.4. Finally, more details are given about implementing NN algorithms with our hardware design.

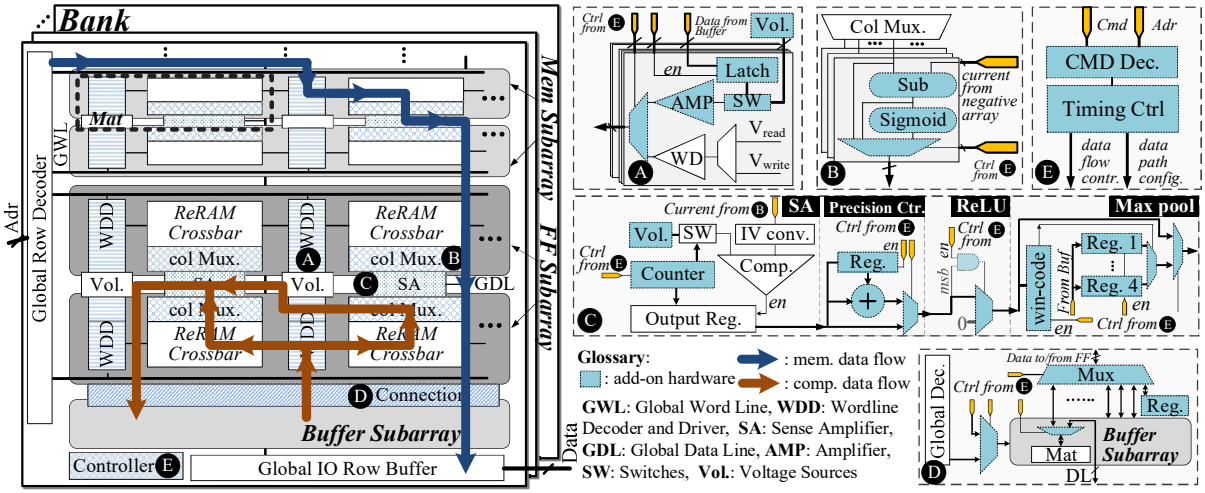

Figure 4.2: The PRIME architecture. Left: bank structure. The blue and red bold lines represent the directions of the data flow for normal memory and for computation, respectively. Right: functional blocks modified/added in PRIME. (A) Wordline driver with multi-level voltage sources; (B) column multiplexer with analog subtraction and sigmoid circuitry; (C) reconfigurable SA with counters for multi-level outputs, and added ReLU and 4-1 max pooling function units; (D) connection between the FF and Buffer subarrays; (E) PRIME controller.

4.1.1 FF Subarray Design

The design goal for FF subarray is to support both storage and computation with a minimum area overhead. To achieve this goal, we maximize the reuse of peripheral circuits for both storage and computation.

Microarchitecture and Circuit Design

To enable the NN computation function in FF subarrays, we modify decoders and drivers, column multiplexers (MUX), and sense amplifiers (SA) as shown in Figure 4.2.

Decoder and Driver. We add several components in decoders and drivers marked as light blue in Figure 4.2 Ⓐ. First, we attach multi-level voltage sources to the wordlines to provide accurate input voltages. NN computation requires that all input data are simultaneously fed into the corresponding wordline. Therefore, we add a latch to control the input voltage. The control signals determine the combination of voltage sources that provide the demanding input voltage. Second, to drive the analog signals transferring on the wordlines, we employ a separate current amplifier on each wordline. Third, rather than two voltage levels used in the memory mode (for read and write, respectively), NN computation requires $2^{P_{\text{in}}}$ levels of input voltages. We employ a multiplexer to switch the voltage driver between memory and computation modes. Finally, we employ two crossbar arrays store positive and negative weights, respectively, and allow them to share the same input port.

Column Multiplexer. In order to support NN computation, we modify the column multiplexers in ReRAM by adding the components marked in light blue in Figure 4.2 Ⓑ. The modified column multiplexer incorporates two analog processing units: an analog subtraction unit and a non-linear threshold (sigmoid) unit [121]. The sigmoid unit can be bypassed in certain scenarios, e.g. when a large NN is mapped to multiple crossbar arrays. In addition, in order to allow FF subarrays to switch bitlines between memory and computation modes, we attach a multiplexer to each bitline to control the switch. Since a pair of crossbar arrays with positive and negative weights require one set of such peripheral circuits, we only need to modify half of the column multiplexers. After analog processing, the output current is sensed by local SAs.

Sense Amplifier. Figure 4.2 Ⓒ shows the SA design with the following modifications as marked in light blue in the figure. First, NN computation requires SAs to offer much higher

precision than memory does. We adopt a P_o-bit ($P_o \leq 8$) precision reconfigurable SA design that has been tested through fabrication [122]. Second, we allow SA's precision to be configured as any value between 1-bit and P_o-bit, controlled by the counter as shown in Figure 4.2 Ⓒ. The result is stored in the output registers. Third, we allow low-precision ReRAM cells to perform NN computation with a high-precision weight, by developing a precision control circuit that consists of a register and an adder. Fourth, we add a hardware unit to support ReLU function, a function in the convolution layer of CNN. The circuit checks the sign bit of the result. It outputs zero when the sign bit is negative and the result itself otherwise. Finally, a circuit to support 4-1 max pooling is included. More details are discussed in Section 4.1.5.

Buffer Connection. Figure 4.2 Ⓓ shows the communication between the FF subarrays and the Buffer subarrays. One adjacent Mem subarray is configured to serve as a buffer for computation, which will be explained in Section 4.1.2. We enable an FF subarray to access any physical location in a Buffer subarray to accommodate the random memory access pattern in NN computation (e.g., in the connection of two convolutional layers). To this end, extra decoders and multiplexers are employed in the buffer connection unit. Additionally, we allow the data transfer to bypass the Buffer subarray in certain scenarios, e.g. when the output of one mat is exactly the input of another. After bypassing the Buffer subarrays, we employ a register as an intermediate data storage.

Benefits of Our Design are two-fold. First, our design efficiently utilizes the peripheral circuits by sharing them between memory and computation functions, which significantly reduces the area overhead. For example, in a typical ReRAM-based neuromorphic computing system [123], DACs and ADCs are used for input and output signal conversions; in a ReRAM-based memory system, SAs and write drivers are required for read and write operations. Yet, SAs and ADCs serve similar functions, while write drivers and DACs do similar functions. In PRIME, instead of using both, we reuse SAs and write drivers to serve ADC and DAC func-

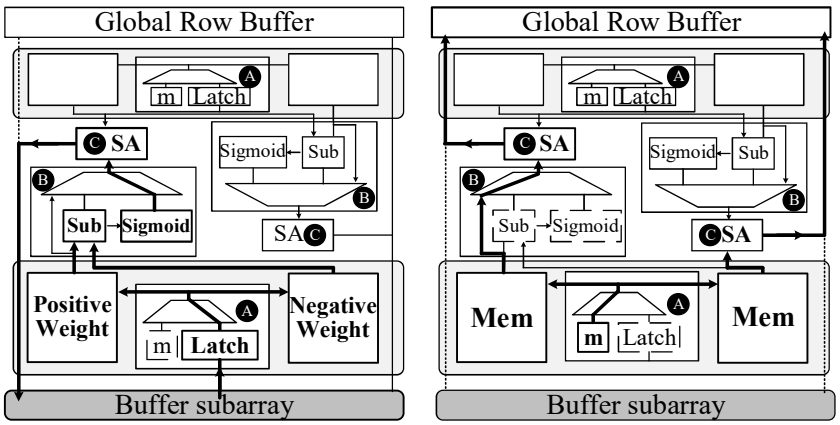

Figure 4.3: An example of the configurations of FF subarrays. (a) Computation mode; (b) memory mode.

tions by slightly modifying the circuit design. Second, we enable the FF subarrays to flexibly and efficiently morph between memory and computation modes. Furthermore, we can extend our design to aggressively configure the FF subarrays in an asymmetric manner: several mats in one FF subarray serve computation, while the rest mats as memory. Such design is feasible, because the multiplexers can connect all of the newly added circuits with the original peripheral circuits that only serve memory function.

Morphing Between Two Modes

Figure 4.3 shows two FF subarrays that are configured into computation and memory modes, respectively. The black bold lines in the figure demonstrate the data flow in each configuration. As shown in Figure 4.3(a), in computation mode, the FF subarray fetches the input data of the NN from the Buffer subarray into the latch of the wordline decoder and driver. After the computation in the crossbar arrays that store positive and negative weights, their output signals are fed into the subtraction unit, and then the difference signal goes into the sigmoid unit. The analog output is converted to digital signal by the SA is written back to the Buffer

subarray. As shown in Figure 4.3(b), in memory mode, the input comes from the read/write voltage selection (denoted by an **m** box), and the output bypasses the subtraction and sigmoid units.

The morphing between memory and computation modes involves several steps. Before the FF subarrays switch from memory mode to computation mode, PRIME migrates the data stored in the FF subarrays to certain allocated space in Mem subarrays, and then writes the synaptic weights to be used by computation into the FF subarrays. When data preparations are ready, the peripheral circuits are reconfigured by the PRIME controller, and the FF subarrays are switched to computation mode and can start to execute the mapped NNs. After completing the computation tasks, the FF subarrays are switched back to memory mode through a wrap-up step that reconfigures the peripheral circuits.

4.1.2 Buffer Subarrays

The goal of the Buffer subarrays is two-fold. First, they are used to cache the input and output data for the FF subarrays. Benefiting from the massive parallelism of matrix-vector multiplication provided by ReRAM crossbar structures, the computation itself takes a very short time. Moreover, the data input and output may be serial, and their latencies become potential bottlenecks. Therefore, it is necessary to cache the input and output data. Second, the FF subarrays can communicate with the Buffer subarrays directly without the involvement of the CPU, so that the CPU and the FF subarrays can work in parallel.

We choose to configure the adjacent memory subarray to the FF subarrays as the Buffer subarray, which is close to both the FF subarrays and the global row buffer so as to minimize the delay. We do not utilize the local row buffer because it is not large enough to serve typical NNs. We do not implement the buffer with low-latency SRAM due to its large area and cost overhead.

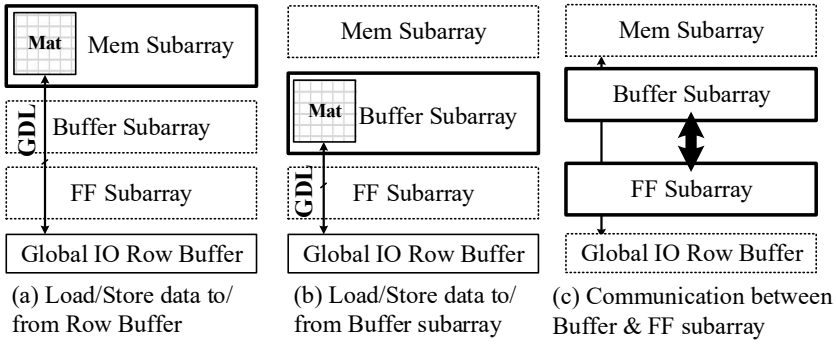

Figure 4.4: Data movements inside memory.

As described in Figure 4.4, the Buffer subarray and the FF subarrays are connected by the connection unit which enables the FF subarrays to access any data in the buffer. To fetch data for the FF subarrays, the data are first loaded from a Mem subarray to the global row buffer, and then they are written from the row buffer to the Buffer subarray. These two steps have to be done in serial due to the resource conflict, i.e. the global data lines (GDL). The communication between the Buffer subarray and the FF subarrays is independent with the communication between the Mem subarray and the globe row buffer. Therefore, when PRIME is accelerating NN computation, CPU can still access the memory and work in parallel. To write the data from the Buffer subarray to memory, the data go through the global row buffer to the corresponding Mem subarray. There are three advantages of this design. First, the short delay between the Buffer and the FF subarrays mitigates the data input/output bottleneck of NN computations. Second, normal memory accesses and NN computations can be done in parallel, as the data communication between the Buffer and FF subarrays is through the dedicated wires. Third, caching the output data in the Buffer subarray alleviates the impact of the slow write operations of ReRAM.

4.1.3 PRIME Controller

Figure 4.2 Ⓔ illustrates the PRIME controller that decodes instructions and provides control signals to all the peripheral circuits in the FF subarrays. A key role of the controller is to configure the FF subarrays in memory and computation modes. Table 4.1 lists the basic commands used by the controller. The left four commands generate control signals for the multiplexers in Figure 4.2, including the function selection of each mat among programming synaptic weights, computation, and memory, and also the input source selection for computation, either from the Buffer subarray or from the output of the previous layer. These commands are performed once during each configuration of the FF subarrays. The right four commands in Table 4.1 control the data movement. They are applied during the whole computation phase.

Datapath Configure	**Data Flow Control**
prog/comp/mem [mat adr][0/1/2]	fetch [mem adr] to [buf adr]
bypass sigmoid [mat adr] [0/1]	commit [buf adr] to [mem adr]
bypass SA [mat adr][0/1]	load [buf adr] to [FF adr]
input source [mat adr][0/1]	store [FF adr] to [buf adr]

Table 4.1: PRIME Controller Commands

4.1.4 Overcoming the Precision Challenge

The precision issue is one of the most critical challenges for ReRAM based NN computation. It contains several aspects: input precision, synaptic weight (or cell resistance) precision, output (or analog computation) precision, and their impacts on the results of NN applications (e.g. the classification accuracy of image recognition tasks).

Previous work has employed 1-bit to 12-bit synaptic weights for ReRAM based NN computation [124–126]. There have been active research going on with improving the resistance precision of MLC ReRAM cells. With a simple feedback algorithm, the resistance of a ReRAM device can be tuned with 1% precision (equivalent to 7-bit precision) for a single cell and about

3% for the cells in crossbar arrays [127, 128].

The latest results of the Dot-Product Engine project from HP Labs reported that, for a 256×256 crossbar array, given full-precision inputs (e.g. usually 8-bit for image data), 4-bit synaptic weights can achieve 6-bit output precision, and 6-bit synaptic weights can achieve 7-bit output precision, when the impacts of noise on the computation precision of ReRAM crossbar arrays are considered [129].

We evaluated the impacts of input and synaptic weight precisions on a handwritten digit recognition task using LeNet-5, a well-known CNN, over the MNIST database [130]. We adopt the dynamic fixed point data format [131], and apply it to represent the input data and synaptic weights of every layer. From the results as shown in Figure 4.5, for this NN application, 3-bit dynamic fixed point input precision and 3-bit dynamic fixed point synaptic weight precision are adequate to achieve 99% classification accuracy, causing negligible accuracy loss compared with the result of floating point data format. The results indicate that NN algorithms are very robust to the precisions of input data and synaptic weights.

Our PRIME design can be adapted to different assumptions of input precision, synaptic weight precision, and output precision. According to the state-of-the-art technologies used in ReRAM based NN computation, one practical assumption is that: the input voltage have only 3-bit precision (i.e. 8 voltage levels), and the ReRAM cells can only represent 4-bit synaptic weights (i.e. 16 resistance levels), and the target output precision is 6-bit. The data format we use is dynamic fixed point [131]. To achieve high computation accuracy with conservative assumptions, we propose an input and synapse composing scheme, which can use two 3-bit input signals to compose one 6-bit input signal and two 4-bit cells to represent one 8-bit synaptic weight.

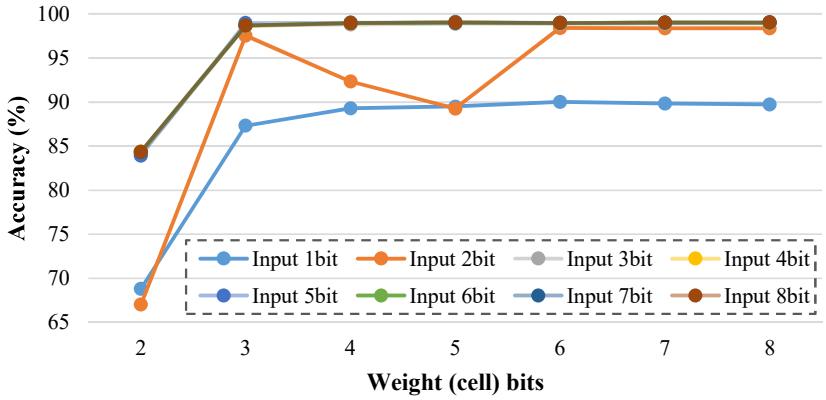

Figure 4.5: The precision result.

Input and Synapse Composing Scheme

We present the input and synapse composing algorithm first, and then present the hardware implementation. Table 4.2 lists the notations.

If the computation in a ReRAM crossbar array has full accuracy, the result should be

$$R_{\text{full}} = \sum_{i=1}^{2^{P_N}} \left(\sum_{k=1}^{P_{\text{in}}} I_k^i 2^{k-1} \cdot \sum_{k=1}^{P_w} W_k^i 2^{k-1} \right), \tag{4.1}$$

P_{in}, P_o, P_w	the number of bits for input/output/synaptic weights
P_N	the number of inputs to a crossbar array is 2^{P_N}
I_k^i, W_k^i	the k^{th} bit of the i^{th} input signal/synaptic weight
Ih_k^i, Il_k^i	the k^{th} bit of HIGH/LOW-bit part of the i^{th} input
Wh_k^i, Wl_k^i	the k^{th} bit of HIGH/LOW-bit part of the i^{th} weight

Table 4.2: Notation Description.

which has $(P_{\text{in}} + P_w + P_N)$-bit full precision. Since the target output is P_o-bit, we will take the highest P_o-bit of R_{full}. Then, the target result is denoted as shifting R_{full} to the right by

$(P_{\text{in}} + P_w + P_N - P_o)$ bits:

$$R_{\text{target}} = R_{\text{full}} \gg (P_{\text{in}} + P_w + P_N - P_o). \tag{4.2}$$

Now each input signal and synaptic weight are composed of two parts: high-bit part and low-bit part. We have,

$$\text{input: } \sum_{k=1}^{P_{\text{in}}} I_k^i 2^{k-1} = \sum_{k=1}^{P_{\text{in}}/2} (Ih_k^i 2^{k-1} \cdot 2^{P_{\text{in}}/2} + Il_k^i 2^{k-1}) \tag{4.3}$$

$$\text{weight: } \sum_{k=1}^{P_w} W_k^i 2^{k-1} = \sum_{k=1}^{P_w/2} (Wh_k^i 2^{k-1} \cdot 2^{P_w/2} + Wl_k^i 2^{k-1}). \tag{4.4}$$

Then, R_{full} will contain four parts (i.e., HH-part, HL-part, LH-part, and LL-part),

$$R_{\text{full}} = \sum_{i=1}^{2^{P_N}} \{ 2^{\frac{P_w+P_{\text{in}}}{2}} \cdot \underbrace{\sum_{k=1}^{P_{\text{in}}/2} Ih_k^i 2^{k-1} \sum_{k=1}^{P_w/2} Wh_k^i 2^{k-1}}_{HH-part}$$

$$+ 2^{\frac{P_w}{2}} \cdot \underbrace{\sum_{k=1}^{P_{\text{in}}/2} Il_k^i 2^{k-1} \sum_{k=1}^{P_w/2} Wh_k^i 2^{k-1}}_{HL-part} \tag{4.5}$$

$$+ 2^{\frac{P_{\text{in}}}{2}} \cdot \underbrace{\sum_{k=1}^{P_{\text{in}}/2} Ih_k^i 2^{k-1} \sum_{k=1}^{P_w/2} Wl_k^i 2^{k-1}}_{LH-part} + \underbrace{\sum_{k=1}^{P_{\text{in}}/2} Il_k^i 2^{k-1} \sum_{k=1}^{P_w/2} Wl_k^i 2^{k-1}}_{LL-part} \} . \tag{4.6}$$

Here, we rewrite R_{full} as

$$R_{\text{full}} = 2^{\frac{P_w+P_{\text{in}}}{2}} \cdot R_{\text{full}}^{\text{HH}} + 2^{\frac{P_w}{2}} \cdot R_{\text{full}}^{\text{HL}} + 2^{\frac{P_{\text{in}}}{2}} \cdot R_{\text{full}}^{\text{LH}} + R_{\text{full}}^{\text{LL}}. \tag{4.7}$$

We can also denote R_{target} with four parts:

$$R_{\text{target}} = R_{\text{tar}}^{\text{HH}} + R_{\text{tar}}^{\text{HL}} + R_{\text{tar}}^{\text{LH}} + R_{\text{tar}}^{\text{LL}}. \tag{4.8}$$

In equation (4.7), if the output of each R_{full} part is only P_o-bit, then,

- $R_{\text{tar}}^{\text{HH}}$: take all the P_o bits of $R_{\text{full}}^{\text{HH}}$ result
- $R_{\text{tar}}^{\text{HL}}$: take the highest $P_o - \frac{P_{\text{in}}}{2}$ bits of $R_{\text{full}}^{\text{HL}}$ result
- $R_{\text{tar}}^{\text{LH}}$: take the highest $P_o - \frac{P_w}{2}$ bits of $R_{\text{full}}^{\text{LH}}$ result
- $R_{\text{tar}}^{\text{LL}}$: take the highest $P_o - \frac{P_{\text{in}}+P_w}{2}$ bits of $R_{\text{full}}^{\text{LL}}$.

According to our assumptions, we have $P_{\text{in}} = 6$ (composed of two 3-bit signals), $P_w = 8$ (composed of two 4-bit cells), and $P_{\text{out}} = 6$ (enabled by 6-bit precision reconfigurable sense amplifiers). The target result should be the summation of three components: all the 6 bits of $R_{\text{full}}^{\text{HH}}$ output, the highest 3 bits of $R_{\text{full}}^{\text{HL}}$ output, and the highest 2 bits of $R_{\text{tar}}^{\text{LH}}$ output.

To implement synapse weight composing, P_{in} is loaded to the latch in the WL driver as shown in Figure 4.2 Ⓐ. According to the control signal, the high-bit and low-bit parts of the input are fed to the corresponding crossbar array sequentially. To implement synapse composing, the high-bit and low-bit parts of the synaptic weights are stored in adjacent bitlines of the corresponding crossbar array. As shown in Equation (4.8), R_{target} consists of four components. They are calculated one by one, and their results are accumulated with the adder in Figure 4.2 Ⓒ. The right shift operation, i.e. taking the highest several bits of a result, can be implemented by the reconfigurable SA. To take the highest n-bit of a result, we simply configure the SA as an n-bit SA.

4.1.5 Implementing NN Algorithms

MLP/Fully-connected Layer: Matrix-vector Multiplication. Matrix-vector multiplication is one of the most important primitives in NN algorithms. The ReRAM crossbar arrays are used to implement it: the weight matrix is pre-programmed in ReRAM cells; the input vector is the voltages on the wordlines driven by the drivers (as shown in Figure 4.2 Ⓐ); the output currents are accumulated at the bitlines. The synaptic weight matrix is separated into two matrices: one

storing the positive weights and the other storing the negative weights. They are programmed into two crossbar arrays. A subtraction unit (as shown in Figure 4.2 Ⓑ) is used to subtract the result of the negative part from that of the positive part. One challenge in using ReRAM crossbar arrays for neural computing is the limited precision of weights presented by ReRAM cells. To tackle this challenge, our design leverages low-precision (e.g., 4-bit) cells to achieve high-precision (e.g., 8-bit) results, taking advantage of the reconfigurable SA and the precision control hardware in Figure 4.2 (C).

MLP/Fully-connected Layer: Activation Function. Our circuit design supports two activation functions: sigmoid and ReLU. Sigmoid is implemented by the sigmoid unit in Figure 4.2 Ⓑ, and ReLU is implemented by the ReLU unit in Figure 4.2 Ⓒ. These two units can be configured to bypass in some scenarios. For example, when we compose a high-precision result with multiple low-precision results as shown in Equation (4.8), those low-precision results will bypass the sigmoid units. After the high-precision result is obtained, we feed it to the sigmoid unit.

Convolution Layer. The computation of the convolution layer is described as follows,

$$f_i^{\text{out}} = max(\sum_{j=1}^{n_{\text{in}}} f_j^{\text{in}} \otimes g_{i,j} + b_i, 0),\ 1 \le i \le n_{\text{out}}, \tag{4.9}$$

where f_j^{in} is the j-th input feature map, and f_i^{out} is the i-th output feature map, $g_{i,j}$ is the convolution kernel for f_j^{in} and f_i^{out}, b_i is the bias term, and n_{in} and n_{out} are the numbers of the input and output feature maps, respectively.

To implement the summation of n_{in} convolution operations ($f_j^{\text{in}} \otimes g_{i,j}$) plus b_i, all the elements of j convolution kernels $g_{i,j}$ are pre-programmed in the ReRAM cells of one BL or more BLs if they cannot fit in one, and the elements of f_j^{in} are performed as input voltages. We also write b_i in ReRAM cells, and regard the corresponding input as "1". Each BL will output the whole or part of the convolution result. If more BLs are used, it takes one more step to achieve

the final result. Next, the $\max(x,0)$ function is executed by the ReLU logic in Figure 4.2 ❻.

Pooling Layer. To implement max pooling function, we adopt 4:1 max pooling hardware in Figure 4.2 ❼, which is able to support n:1 max pooling with multiple steps for $n > 4$. For 4:1 max pooling, first, four inputs $\{a_i\}$ are stored in the registers, $i = 1,2,3,4$; second, we execute the dot products of $\{a_i\}$ and six sets of weights [1,-1,0,0], [1,0,-1,0], [1,0,0,-1], [0,1,-1,0], [0,1,0,-1], [0,0,1,-1] by using ReRAM to obtain the results of $(a_i - a_j)$, $i \neq j$; next, the signs of their results are stored in the Winner Code register; finally, according to the code, the hardware determines the maximum and outputs it. Mean pooling is easier to implement than max pooling, because it can be done with ReRAM and does not require extra hardware. To perform n:1 mean pooling, we simply pre-program the weights $[1/n,\cdots,1/n]$ in ReRAM cells, and execute the dot product of the inputs and the weights to obtain the mean value of n inputs.

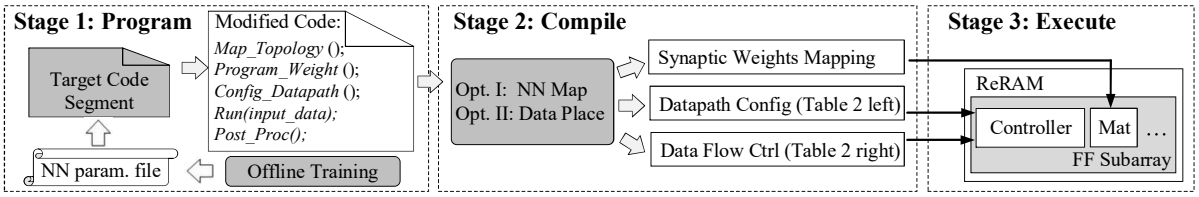

Figure 4.6: The software perspective of PRIME: from source code to execution.

Local Response Normalization (LRN) Layer. Currently, PRIME does not support LRN acceleration. We did not add the hardware for LRN, because state-of-the-art CNNs do not contain LRN layers [132]. When LRN layers are applied, PRIME requires the help of CPU for LRN computation.

4.2 System-Level Design

In this section, we present the system-level design of PRIME. The software-hardware interface framework is described. Then, we focus on the optimization of NN mapping and data allocation during compile time. Next, we introduce the operating system (OS) support for

switching FF subarrays between memory and computation modes at run time.

4.2.1 Software-Hardware Interface

Figure 4.6 shows the stack of PRIME to support NN programming, which allows developers to easily configure the FF subarrays for NN applications[1]. From software programming to hardware execution, there are three stages: programming (coding), compiling (code optimization), and code execution. In the programming stage, PRIME provides application programming interfaces (APIs) so that they allow developers to: 1) map the topology of the NN to the FF subarrays, *Map_Topology*, 2) program the synaptic weights into mats, *Program_Weight*, 3) configure the data paths of the FF subarrays, *Config_Datapath*, 4) run computation, *Run*, and 5) post-process the result, *Post_Proc*. In our work, the training of NN is done off-line so that the inputs of each API are already known (*NN param.file*). Prior work explored to implement training with ReRAM crossbar arrays [125, 133–137], and we plan to further enhance PRIME with the training capability in future work.

In the compiling stage, the NN mapping to the FF subarrays and the input data allocation are optimized (as described in Section 4.2.2). The output of compiling is the metadata for synaptic weights mapping, data path configuration, and execution commands with data dependency and flow control. The metadata is also the input for the execution stage. In the execution stage, PRIME controller writes the synaptic weights to the mapped addresses in the FF subarrays; then it (re-)configures the peripheral circuits according to the *Datapath Configure* commands (Table 4.1 left) to set up the data paths for computation; and finally, it executes *Data Flow Control* commands (Table 4.1 right) to manage data movement into or out of the FF subarrays at runtime.

[1]Due to the space limit, we only depict the key steps at high level while the design details of the OS kernel, compiler, and tool chains are left as engineering work.

4.2.2 Compile Time Optimization

NN Mapping Optimization

The mapping of the NN topology to the physical ReRAM cells is optimized during compile time. For different scales of NNs, we have different optimizations.

Small-Scale NN: Replication. When an NN can be mapped to a single FF mat, it is small-scale. Although we can simply map a small-scale NN to some cells in one mat, the other cells in this mat may be wasted. Moreover, the speedup for very small NNs is not obvious, because the latency of the peripheral circuits may overwhelm the latency of matrix-vector multiplication on ReRAM cells. Our optimization is to replicate the small NN to different independent portions of the mat. For example, to implement a $128-1$ NN, we duplicate it and map a $256-2$ NN to the target mat. This optimization can also be applied to convolution layers. Furthermore, if there is another FF mat available, we can also duplicate the mapping to the second mat, and then the two mats can work simultaneously, as long as the Buffer subarray has enough bandwidth.

Medium-Scale NN: Split-Merge. When an NN cannot be mapped to a single FF mat, but can fit to the FF subarrays of one bank, it is medium-scale. During the mapping at compile time, a medium-scale NN has to be split into small-scale NNs, and then their results are merged. For example, to implement a $512-512$ NN on PRIME with $256-256$ mats, it is split into four $256-256$ parts ($[M_{1,1}, M_{1,2}; M_{2,1}, M_{2,2}]$) and mapped to four different mats. After they finish computation, the results of $M_{1,1}$ and $M_{2,1}$ are added to get the first 256 elements of the final result, and the sum of the results of $M_{1,2}$ and $M_{2,2}$ forms the second 256 elements of the final result.

Large-Scale NN: Inter-Bank Communication. A large-scale NN is one NN that cannot be mapped to the FF subarrays in a single bank. Intuitively, we can divide it into several medium-scale trunks and map each trunk to the same bank serially in several stages. This

naive solution requires reprogramming the FF subarrays at every stage, and the latency overhead of reprogramming may offset the speedup. Alternatively, PRIME allows to use multiple banks to implement a large-scale NN. These banks can transfer data to each other and run in a pipelined fashion to improve the throughput. Like prior work [138], the inter-bank data movement is implemented by exploiting the internal data bus shared by all the banks in a chip. PRIME controller manages the inter-bank communication, and can handle arbitrary network connections. If all the banks are used to implement a single NN, PRIME can handle a maximal NN with $\sim 2.7 \times 10^8$ synapses, which is larger than the largest NN that have been mapped to the existing NPUs (TrueNorth [139], 1.4×10^7 synapses). In Section 4.3, we implement an extremely large CNN on PRIME, *VGG-D* [132] which has 1.4×10^8 synapses.

Bank-level Parallelism and Data Placement

Since FF subarrays reside in every bank, PRIME intrinsically inherits bank-level parallelism to speed up computation. For example, for a small-scale or medium-scale NN, since it can be fitted into one bank, the FF subarrays in all the banks can be configured the same and run in parallel. Considering FF subarrays in each bank as an NPU, PRIME contains 64 NPUs in total (8 banks$\times$8 chips) so that 64 images can be processed in parallel. To take advantage of the bank-level parallelism, the OS is required to place one image in each bank and to evenly distribute images to all the banks. As current page placement strategies expose memory latency or bandwidth information to the OS [140, 141], PRIME exposes the bank ID information to the OS, so that each image can be mapped to a single bank. For large-scale NNs, they can still benefit from bank-level parallelism as long as we can map one replica or more to the spare banks.

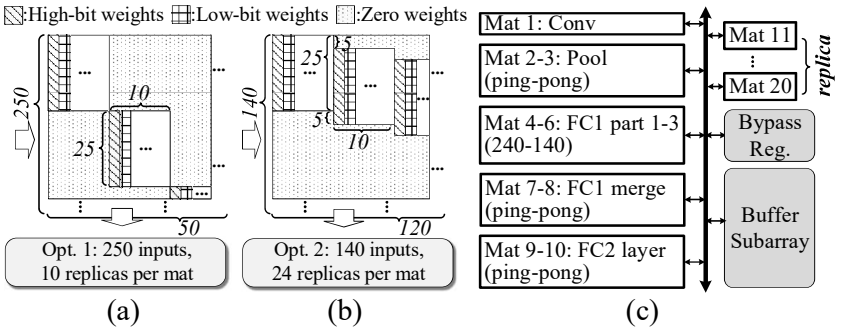

Figure 4.7: A mapping example. (a) Optimization choice 1 and (b) optimization choice 2 for the convolutional layer; (c) mat-level mapping and optimization.

A Mapping Case Study: CNN-1

We show how CNN-1 in Table 4.3 is mapped to PRIME in this section. We use 256×256 ReRAM mats, and ReRAM cells in FF subarrays can be programmed in 4-bit precision and the synaptic weights of the NN are 8-bit in the application. Therefore, to present one weight, two cells are used. In CNN-1, the **convolutional layer** has five 5×5 kernels. Then, the weight matrix is of size $25(=5 \cdot 5) \times 10(=5 \cdot 2)$ [2]. In order to improve latency and ReRAM's utility, the compile-time optimization may replicate the matrix 10 times, and map a 250×100 weight matrix to a 256×256 mat, as shown in Figure 4.7(a). Actually, the optimization can be better for a convolutional layer. Because two adjacent convolution operations share most of the inputs, e.g., 20 out of 25 inputs for a 5×5 kernel in this case, we are able to map more replicas of the 25×10 weight matrix in one mat in a smart way, as shown in Figure 4.7(b). In this way, we make 24 replicas in one mat, increasing the number of output data from 50 to 120 as well as reducing the number of input data from 250 to 140. In CNN-1, the **pooling layer** adopts $4:1$ max pooling. Since each pooling requires a 4×6 weight matrix, we can map 42 replicas in one mat. In CNN-1, the **fully connected layers** are $720-70-10$. Since the input dimension is larger than 256, we apply the *split-merge* mapping. First, the $720 \times 140(=70 \cdot 2)$

[2]For clarity, we only talk about the positive part of the weight matrix. The mapping optimization also applies to the negative part.

weight matrix is mapped into 3 mats, each of size 240×140. Then, the merging of the results from those three mats is performed in another mat to achieve the final results. In the merging mat, since the weight matrix is 3×1, we can execute 85 merging operations in one mat at the same time.

In each mat, computation and data input/output can work in a pipelined way, thanks to input latches and output registers in Figure 4.2 Ⓐ and Ⓒ. All mats can work in parallel, and the data communication among them either goes through the Buffer subarray or bypasses it with the help of the bypassing registers (as shown in Figure 4.2 Ⓒ). When a mat has bypassed inputs (not from the Buffer subarray but from the bypassing registers) and is dedicated for a whole layer (not one of a set of mats for split-merge mapping), in order to improve the throughput, we can duplicate the mat and make the two mats work in a ping-pong mode, in which one mat receives its inputs from the previous layer while the other does the computation. As shown in Figure 4.7(c), the pooling layer, the merging layer of the fully connected layer 1, and the the fully connected layer 2, are configured to work in ping-pong mode. The whole system takes ten mats, Mats 1 to 10, and we duplicate it using Mats 11 to 20. We do not replicate more because with two copies the data communication latency totally hides the computation latency. More copies cannot further improve the throughput because the latency-dominated data communication is serial among mats through the bus-based Buffer subarray.

4.2.3 Run Time Optimization

When FF subarrays are configured for NN applications, the memory space is reserved and supervised by the OS so that it is invisible to other user applications. However, during the run-time, if none or few of their crossbar arrays are used for computation, and the page miss rate is higher than the predefined threshold (which indicates the memory capacity is insufficient), the OS is able to release the reserved memory addresses as normal memory. It was observed

that the memory requirement varies among workloads, and prior work has proposed to dynamically adjust the memory capacity by switching between SLC and MLC modes in PCM-based memory [142]. The page miss rate curve can be tracked dynamically by using either hardware or software approaches [143]. In our design, the granularity to flexibly configure a range of memory addresses for either computation or memory is crossbar array (mat): when an array is configured for computation, it stores multi-bit synaptic weights; when an array is used as normal memory, it stores data as single-bit cells. The OS works with the memory management unit (MMU) to keep all the mapping information of the FF subarrays, and decides when and how much reserved memory space should be released, based on the combination of the page miss rate and the utilization of the FF subarrays for computation.

4.2.4 Discussion

Overhead of Writing Synaptic Weights. In the execution stage, before the computation starts, we need to program the target ReRAM cells according to the synaptic weights. We adopt a similar mapping engine described in prior work [144]. Programming cells to certain weights is equivalent to writing MLC ReRAM cells, which is both time- and energy-consuming. To reduce the overhead of programming, data comparison write schemes are used [145].

ReRAM Endurance. PRIME does not deteriorate ReRAM's lifetime. Prior techniques that extend ReRAM's lifetime can be integrated into PRIME seamlessly. When serving as memory, ReRAM works as SLCs with 10^{12} [146, 147] lifetime. When serving as accelerators, ReRAM works as MLCs and they are only written at the programming stage with synaptic weights. Given a 10-year lifetime and 10^9 MLC endurance[3], PRIME can be re-programmed every 300ms, which is far ahead of the expected programming rate that occurs daily, weekly, or even monthly.

[3]MLC endurance is reported between 10^7 [148] to 10^{12} [147]. We adopt 10^9 to have a conservative lifetime estimation.

Sneak Current and IR-Drop. When serving as memory, a ReRAM crossbar array suffers from multiple possible current paths. Many previous studies proposed different solutions to solve the sneak current problem at device, circuit, and architecture levels [120,149–154], which are adoptable in PRIME. However, when serving for computing, the sneak current does not matter, since it contributes to the computation [155]. The IR-drop on a wordline is a challenge for both memory and computation functions. Xu *et al.* [120] proposed a double-sided ground biasing technique to solve this problem for ReRAM based main memory, which is adopted in PRIME. Moreover, proper training [137] and layout engineering [156] can further address the IR-drop for ReRAM-based NN computation. We have considered the impact of the IR-drop on computation precision in our design. That is one important reason for our conservative assumption in PRIME that ReRAM based analog computation has only 6-bit output precision.

4.3 Evaluation

In this section, we evaluate our PRIME design. We first describe the experiment setup, and then present the performance and energy results and estimate the area overhead.

4.3.1 Experiment Setup

Benchmark. The benchmarks we use (*MlBench*) comprise six NN designs for machine learning applications, as listed in Table 4.3. *CNN-1* and *CNN-2* are two CNNs, and *MLP-S/M/L* are three multilayer perceptrons (MLPs) with different network scales: small, medium, and large. Those five NNs are evaluated on the widely used *MNIST* database of handwritten digits [130]. The sixth NN, *VGG-D*, is well known for ImageNet ILSVRC [132]. It is an extremely large CNN, containing 16 weight layers and 1.4×10^8 synapses, and requiring $\sim 1.6\times10^{10}$ operations.

PRIME Configurations. There are 2 FF subarrays and 1 Buffer subarray per bank (totally

MlBench		*MLP-S*	784-500-250-10
CNN-1	conv5x5-pool-720-70-10	*MLP-M*	784-1000-500-250-10
CNN-2	conv7x10-pool-1210-120-10	*MLP-L*	784-1500-1000-500-10
VGG-D	conv3x64-conv3x64-pool-conv3x128-conv3x128-pool conv3x256-conv3x256-conv3x256-pool-conv3x512 conv3x512-conv3x512-pool-conv3x512-conv3x512 conv3x512-pool-25088-4096-4096-1000		

Table 4.3: The Benchmarks and Topologies.

64 subarrays). In FF subarrays, for each mat, there are 256×256 ReRAM cells and eight 6-bit reconfigurable SAs; for each ReRAM cell, we assume 4-bit MLC for computation while SLC for memory; the input voltage has 8 levels (3-bit) for computation while 2 levels (1-bit) for memory. With our input and synapse composing scheme, for computation, the input and output are 6-bit dynamic fixed point, and the weights are 8-bit.

Methodology. We compare PRIME with several counterparts. The baseline is a CPU-only solution. The configurations of CPU and ReRAM main memory are shown in Table 4.4, including key memory timing parameters for simulation. We also evaluate two different NPU solutions: using a complex parallel NPU [157] as a co-processor (pNPU-co), and using the NPU as a PIM-processor through 3D stacking (pNPU-pim). The configurations of these comparatives are described in Table 4.5.

Processor	4 cores; 3GHz; Out-of-order
L1 I&D cache	Private; 32KB; 4-way; 2 cycles access;
L2 cache	Private; 2MB; 8-way; 10 cycles access;
ReRAM-based Main Memory	16GB ReRAM; 533MHz IO bus; 8 chips/rank; 8 banks/chip; tRCD-tCL-tRP-tWR 22.5-9.8-0.5-41.4 (ns)

Table 4.4: Configurations of CPU and Memory.

We model the above NPU designs using Synopsys Design Compiler and PrimeTime with 65*nm* TSMC CMOS library. We also model ReRAM main memory and our PRIME system with modified NVSim [158], CACTI-3DD [159] and CACTI-IO [160]. We adopt Pt/TiO2-x/Pt devices [128] with $R_{\text{on}}/R_{\text{off}} = 1k\Omega/20k\Omega$ and 2V SET/RESET voltage. The FF subarray is modeled by heavily modified NVSim, according to the peripheral circuit modifications, i.e., write driver [161], sigmoid [121], and sense amplifier [122] circuits. We built a trace-based in-house simulator to evaluate different systems, including CPU-only, PRIME, NPU co-processor, and NPU PIM-processor.

Description		Data path	Buffer
pNPU-co	Parallel NPU [157] as co-processor	16×16 multiplier 256-1 adder tree	2KB in/out 32KB weight
pNPU-pim	PIM version of parallel NPU, 3D stacked to each bank		

Table 4.5: The Configurations of Comparatives.

4.3.2 Performance Results

The performance results for *MlBench* are presented in Figure 4.8. *MlBench* benchmarks use large NNs and require high memory bandwidth, and therefore they can benefit from PIM. To demonstrate the PIM advantages, we evaluate two pNPU-pim solutions: pNPU-pim-x1 is a PIM-processor with a single parallel NPU stacked on top of memory; and pNPU-pim-x64 with 64 NPUs, for comparison with PRIME which takes advantages of bank-level parallelism (64 banks). By comparing the speedups of pNPU-co and pNPU-pim-x1, we find that the PIM solution has a 9.1× speedup on average over a co-processor solution. Among all the solutions, PRIME achieves the highest speedup over the CPU-only solution, about 4.1× of pNPU-pim-x64's. PRIME achieves a smaller speedup in *VGG-D* than other benchmarks, because it has to map the extremely large *VGG-D* across 8 chips where the data communication between banks/chips is costly. The performance advantage of PRIME over the 3D-stacking PIM solu-

tion (pNPU-pim-x64) for NN applications comes from the efficiency of using ReRAM for NN computation, because the synaptic weights have already been pre-programmed in ReRAM cells and do not require data fetches from the main memory during computation. In our performance and energy evaluations of PRIME, we do not include the latency and energy consumption of configuring ReRAM for computation, because we assume that once the configuration is done, the NNs will be executed for tens of thousands times to process different input data.

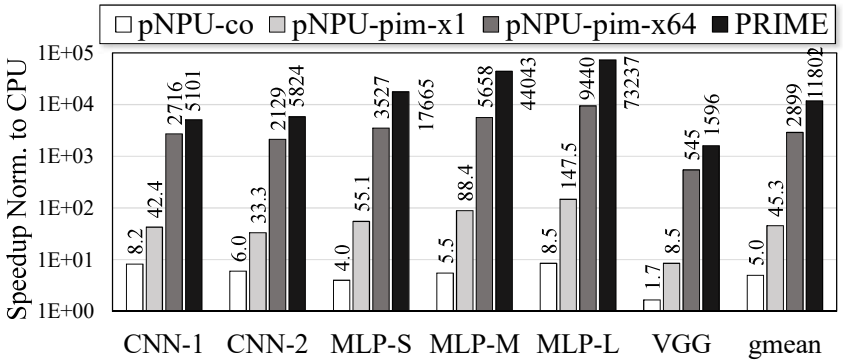

Figure 4.8: The performance speedups (*vs.* CPU).

Figure 4.9 presents the breakdown of the execution time normalized to pNPU-co. To clearly show the breakdown, we evaluate the results of pNPU-pim with one NPU, and PRIME without leveraging bank parallelism for computation. The execution time is divided into two parts, computation and memory access. The computation part also includes the time spent on the buffers of NPUs or the Buffer subarrays of PRIME in managing data movement. We find that pNPU-pim reduces the memory access time a lot, and PRIME further reduces it to zero. Zero memory access time does not imply that there is no memory access, but it means that the memory access time can be hidden by the Buffer subarrays.

4.3.3 Energy Results

The energy saving results for *MlBench* are presented in Figure 4.10. Figure 4.10 does not show the results of pNPU-pim-x1, because they are the same with those of pNPU-pim-x64.

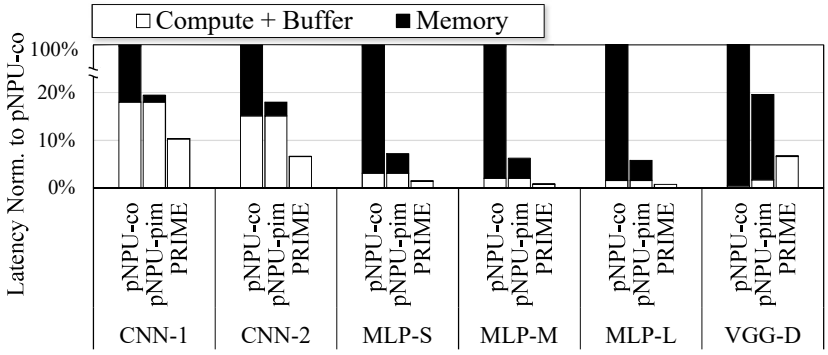

Figure 4.9: The execution time breakdown (*vs.* pNPU-co).

From Figure 4.10, PRIME shows its superior energy-efficiency to other solutions. pNPU-pim-x64 is several times more energy efficient than pNPU-co, because the PIM architecture reduces memory accesses and saves energy. The energy advantage of PRIME over the 3D-stacking PIM solution (pNPU-pim-x64) for NN applications comes from the energy efficiency of using ReRAM for NN computation.

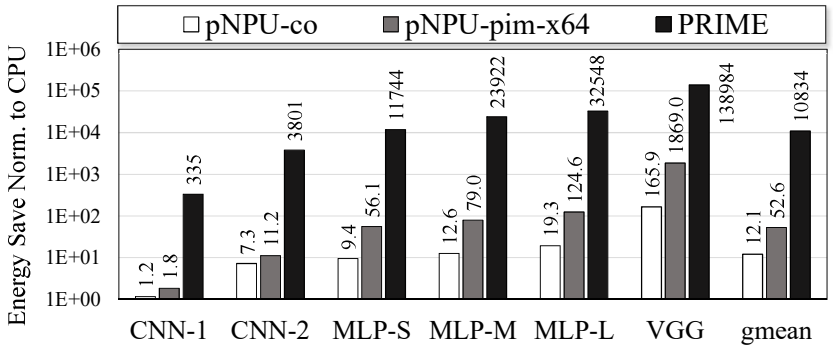

Figure 4.10: The energy saving results (*vs.* CPU).

Figure 4.11 provides the breakdown of the energy consumption normalized to pNPU-co. The total energy consumptions are divided into three parts, computation energy, buffer energy, and memory energy. From Figure 4.11, pNPU-pim-x64 consumes almost the same energy in computation and buffer with pNUP-co, but saves the memory energy by 93.9% on average by decreasing the memory accesses and reducing memory bus and I/O energy. PRIME reduces all

the three parts of energy consumption significantly. For computation, ReRAM based analog computing is very energy-efficient. Moreover, since each ReRAM mat can store 256×256 synaptic weights, the cache and memory accesses to fetch the synaptic weights are eliminated. Furthermore, since each ReRAM mat can execute as large as a $256 - 256$ NN at one time, PRIME also saves a lot of buffer and memory accesses to the temporary data. From Figure 4.11, CNN benchmarks consume more energy in buffer and less energy in memory than MLP benchmarks. The reason is that the convolution layers and pooling layers of CNN usually have a small number of input data, synaptic weights, and output data, and buffers are effective to reduce memory accesses.

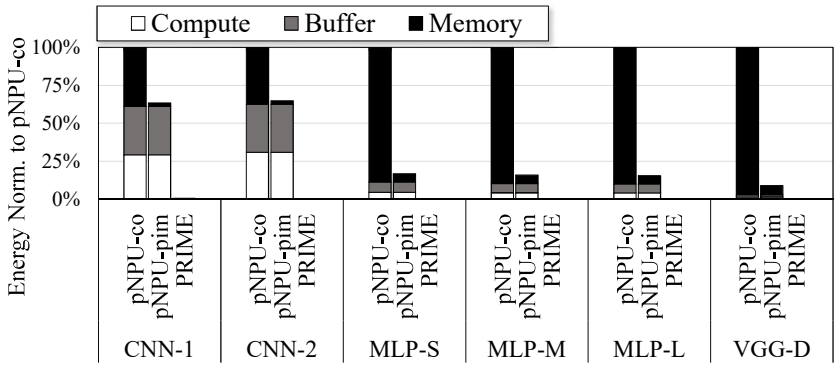

Figure 4.11: The energy breakdown (*vs.* pNPU-co).

4.3.4 Area Overhead

Given two FF subarrays and one Buffer subarray per bank (64 subarrays in total), PRIME only incurs 5.76% area overhead. The choice of the number of FF subarrays is a tradeoff between peak GOPS and area overhead. Our experimental results on *Mlbench* (except VGG-D) show that the utilities of FF subarrays are 39.8% and 75.9% on average before and after replication, respectively. For *VGG-D*, the utilities of FF subarrays are 53.9% and 73.6% before and after replication, respectively. Figure 4.12 shows the breakdown of the area overhead in a mat of an FF subarray. There is 60% area increase to support computation: the added driver

takes 23%, the subtraction and sigmoid circuits take 29%, and the control, the multiplexer, and etc. cost 8%.

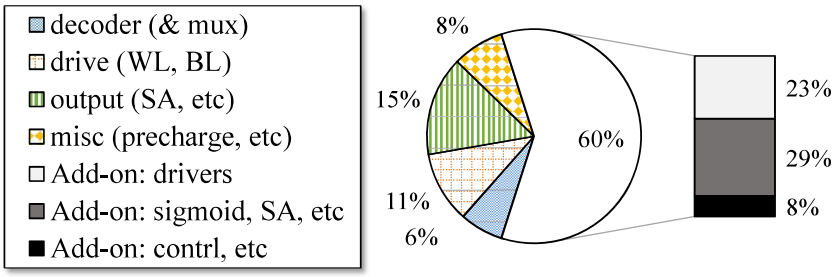

Figure 4.12: Area Overhead of PRIME.

4.4 Conclusion

This paper proposed a novel processing in ReRAM-based main memory design, PRIME, which substantially improves the performance and energy efficiency for neural network (NN) applications, benefiting from both the PIM architecture and the efficiency of ReRAM-based NN computation. In PRIME, part of the ReRAM memory arrays are enabled with NN computation capability. They can either perform computation to accelerate NN applications or serve as memory to provide a larger working memory space. We present our designs from circuit-level to system-level. With circuit reuse, PRIME incurs an insignificant area overhead to the original ReRAM chips. The experimental results show that PRIME can achieves a high speedup and significant energy saving for various NN applications using MLP and CNN.

Chapter 5

NVSIM-CAM: A Circuit-Level Simulator for Emerging Nonvolatile Memory based Content-Addressable Memory

Ternary Content-Addressable Memories (TCAMs) are used for a wide variety of applications, such as associative caches, networking routers, and search engines. TCAMs provide fast match/mismatch responses for in-memory content searching. Conventionally, TCAMs are implemented by SRAMs with 16 transistors per cell [162]. The large cell area and the corresponding large power consumption result in poor scalability. However, the emergence of the nonvolatile memory (NVM) based TCAM (nvTCAM) offers an alternative to overcome the challenge. The emerging NVMs, i.e., Magnetoresistive RAM (MRAM) [163], Phace-Changing RAM (PCM) [164], and Resistive RAM (ReRAM) [165], provide small cell area, nonvolatility, and zero standby power consumption. Consequently, it not only improves the memory design to be denser and more power efficient, but also makes evolution for the TCAM design: a new generation of the nvTCAM with significant area reduction, low power consumption, better scalability, and instant-on/off features.

The nvTCAM has an even boarder influence. It can also pave ways for corresponding architecture innovations, which otherwise, are impossible with conventional SRAM-TCAMs. For example, thanks to nvTCAM's low power consumption and the instant-on/off feature, Chang*et al.* [166] has proposed to use nvTCAM for deep packet inspection in the energy-hungry IoT scenario. SRAM-TCAM is not competent due to large leakage power. Similarly, taking advantages of the nvTCAM's high density feature, Ipek*et al.* [167] has proposed a nvTCAM-based accelerator for data intensive applications. SRAM-TCAM is not adoptable due to the poor scalability. These work has shown the trend that as the nvTCAM keeps developing, there will be more edge-cutting techniques that call for the architecture/system design rethinking.

In order to keep pase with the ever-changing nvTCAM technology, a circuit-level model and a simulation tool are essential. However, building such a tool is challenging. First, targeting at emerging technology, the tool should not only focus on circuit modeling, but also be able to capture performance related device characters. Second, different from the almost fixed SRAM-TCAM cell structure, the nvTCAM faces a diversity of the cell structures. The tool should be able to support the flexibility of the cell design. Third, most of the nvTCAMs are sensitive with the search word size, which results in extra design knobs and constraints. Considering all these challenges, the nvTCAM design ends up with a much larger design space than that of either NVMs or SRAM-TCAMs. To find a sweet point from the large design space, the automatic tool is preferred.

Previous work set foot in modeling either the SRAM-TCAM or the emerging NVM, but they are not adoptable for the nvTCAM modeling. CACTI [168] and its variants [169] have modeled SRAM-based full associate cache, but no emerging technologies are supported. Sherwood*et al.* [170] has proposed a power model for TCAMs, but again, they only focused on SRAM-TCAMs. On the other hand, NVSim [158] is widely used for emerging NVM performance, area, and power evaluation, but the support for nvTCAM is not yet developed. Most recently, Chen*et al.* [171] has made comprehensive comparisons and design space explorations

among three types of MRAM-TCAM cells. However, they did not provide a universal model and simulation platform, either.

In this chapter, for the first time, we develop a universal simulation tool for nvTCAMs, named NVSIM-CAM. A following case study presents this tool's competency of early stage projection and design space exploration, with a novel 3D vertical ReRAM based nvTCAM design. Our specific contributions in this paper are listed as follows,

- We develop a circuit-level model of nvTCAM which provides full-support for the indispensable diversity and flexibility of the nvTCAM design given their many possible choices of cell structures and circuit optimizations. We implement our model on a simulator framework NVsim with heavy modifications of its code base.
- We validate our model with fabricated nvTCAM prototypes, and the results show that we can achieve 3.5% error on average for several chips with different designs. We also demonstrate the competency of the tool in exploring a huge design space of nvTCAM at an early design phase.
- We propose a novel and extremely high-density TCAM design based on low-cost vertical 3D vertical ReRAM. We use NVSIM-CAM to evaluate the design and demonstrate 234× higher density than the state-of-the-art design. We then project the superiority and identify the limitation of 3DvTCAM based on our evaluations with the tool. We also discuss the potential applications and architecture innovations facilitated by the 3DvTCAM.

5.1 Background and Overview

This section shows an overview of the micro-architecture of a general nvTCAM, and the framework of the developed NVSIM-CAM tool.

5.1.1 Principle working mechanism of TCAMs

Though there are plenty types of nvTCAM cell structures, they stick to the principle NOR-type[1] working mechanism. The TCAM cells are connected together with the matchline (ML). In order to describe three states ("0", "1", and "x"), the cell usually contains two single-level cells or one MLC. The querying data is transferred to each corresponding cells through the searchline (SrL) during search operations. By properly encoding the storing data and the querying data, the cell can output a logical "0" for mismatch and logic "1" for match. The ML performs an overall logic "AND" operation for all the cell matching result on it: If any mismatch shows up ("0"), the final result shows a mismatch ("0").

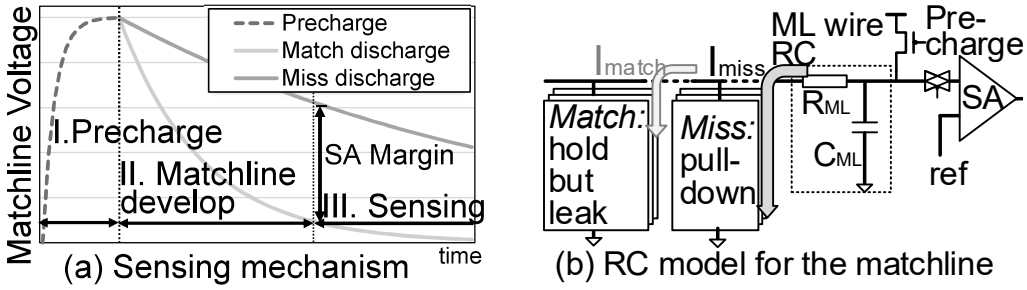

Figure 5.1: The ML sensing machanism.

Figure 5.1 (a) shows the sensing mechanism for (nv)TCAM. There are three phases. In the first phase, the ML is charged to a high voltage. In the second phase, the SrLs are activated to evaluate matching. The ML starts to discharge. In the last phase, the ML voltage difference between match and mismatch is large enough for a sense amplifier (SA) to sense. We denote the minimal ML voltage difference between a match and a mismatch as the sense margin (shown in Figure 5.1 (a)). Figure 5.1 (b) shows the circuit model. A mismatch cell generates I_{miss} to discharge the ML. This current is much larger than I_{match}, which is the leaking discharge current from a match cell. A reasonable design should have a small I_{miss} to prevent leakage during match, so that the SA sense margin is large enough. At the same time, a larger I_{match} is preferred, since the ML can discharge quickly and it results in a smaller search latency.

[1]NAND-type suffers from voltage drop and pool stability, and hence is rarely adopted [172].

5.1.2 The bank organization and components

Figure 5.2 (left) shows the bank-level architecture for a nvTCAM. As the lower level micro-block, Mats within a bank are connected with H-tree [158] (the bus-like connection is also supported). In order to support a large query word size, the word is able to be partitioned among the Mats. In this situation, the search operations inside each Mat work simultaneously, and their results are merged (by AND logic) at the joint point of the H-tree routing.

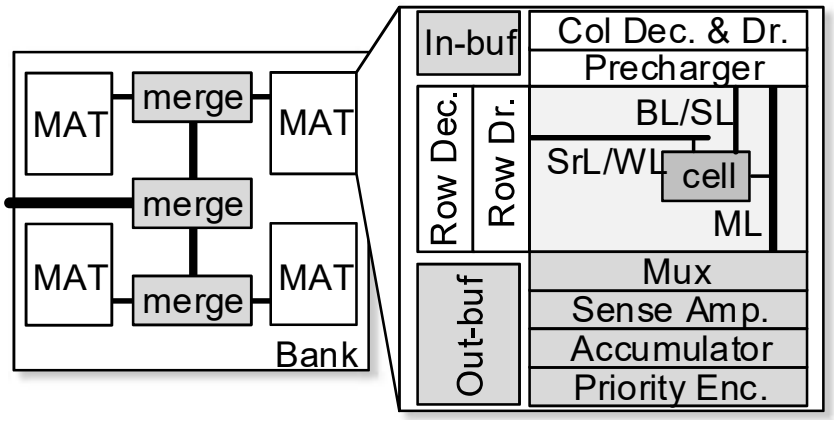

Figure 5.2: The bank organization (left) and components within a Mat (right). Glossary: Decoder (Dec.), Driver (Dr.), Encoder (Enc.)

Figure 5.2 (right) shows the components that build a nvTCAM Mat. NVSIM-CAM is based on NVSim [158] but there are plenty of differences between the TCAM and normal memories, as marked with dark colors. Besides the WL/BL/SL, there are also SrLs and MLs in nvTCAMs. There are also unique components in nvTCAM, including the accumulator and the priority encoder.

5.1.3 The framework of NVSIM-CAM

Figure 5.3 shows the framework of the NVSIM-CAM tool. The design knobs include data organizations, technologies, component settings, and cell designs. The detailed configurable items are listed in Table 5.1. These design knobs are either fixed for a certain design projection, or input as a range for DSEs. The optimization objective and design constraints are supported.

The output of NVSIM-CAM is the performance/power/area parameters of the best design that meets the design specification and constraints.

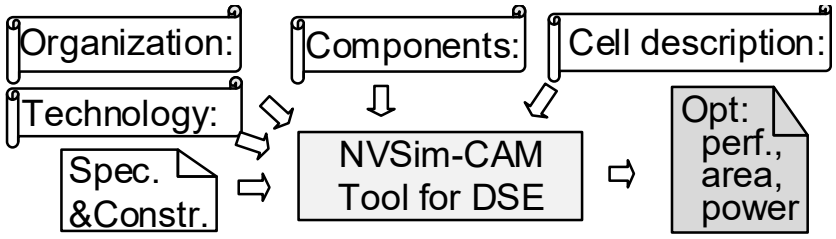

Figure 5.3: The framework of NVSIM-CAM.

Organization:	**Component:**
Bank/Mat size: H-tree, partition	SA types (vol./cur.)
SA sharing: Mux, local/global	Buf., Acc., Priority enc.
Bit serial width (if appliable)	Drivers opt. target
Cell description:	
Cell type (Diode/NMOS/Direct); Device parameters	
Description for each port: transistor size, V/I in every op.	

Table 5.1: NVSIM-CAM's input and design knobs.

5.2 NVSIM-CAM Development

In this section, we show the development of NVSIM-CAM. We first show how NVSIM-CAM models different cell structures. Then, in order to improve the simulation precision, we propose the customized-SA based modeling. In the end, we validate NVSIM-CAM with fabricated nvTCAM prototypes.

5.2.1 Description of various cell structures

The nvTCAM cell structure design is flexible. For example, from simple cells of 2T2R [164] or 3T1R [165], to the complex cell of 6T2R [173], there are reasonable designs with fabricated prototypes. They are adopted according to different design targets. To model the cell diversity,

we focus on three aspects: the cell's impact on peripheral circuits, the intra-cell currents, and the cell's impact on ML development.

Cell's impacts on peripheral circuits

A cell can have multiple ports connected to the row/column wires (WL/SrL/etc). For example, the 3T1R cell [174] is connected to three row wires and three column wires. These wires determine the corresponding row/column drivers and the multiplex's design. To capture these impacts, we need a description of each port, including the connected transistor's size and the connected wire's width. By these descriptions, the RC model of the row/column wire is established. Then, with the description of the voltage and/or current that applied to this port during search/write operations, the maximal current on the wire is calculated, and hence we have the parameters for driver/mux design (RC load and maximal current).

Intra-cell currents

Two kinds of intra-cell currents are considered in NVSIM-CAM. First, there could be direct current (DC) in the cells during search, for example, in the 4T2R cell [175]. The DC needs to be counted for power consumption. Second, although based on NVM, the nvTCAM cell still suffers from leakage, which needs to be included in the leakage power.

Cell's impacts on ML development

The ML development is essential for calculating sensing latency and checking sense margin constraints. To model various cell structures' impact on ML, we classify all those cell designs into three categories: Diode-access, NMOS-access, and Direct-access, as shown in Figure 5.4. To support various cell structure designs, the circuit in the dashed box is flexible with any design.

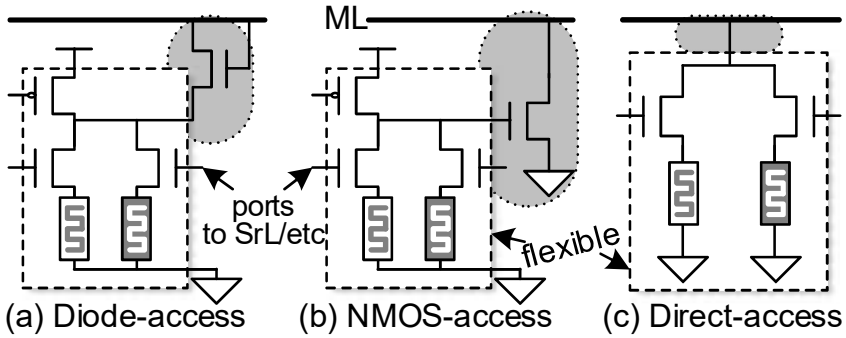

Figure 5.4: Three types of nvTCAM cell structures.

Diode-access nvTCAM. A diode (implemented by NMOS) is used to connect the cells to the ML. While mismatching, a low voltage is generated to the diode and turns it on, setting a path to discharge the ML. For matches, a higher-than-ML voltage is output to the diode and turns it off. The 4T2R cell structure [163] in Figure 5.4 (a) is an example. A match operation connects one or two storage cells with R_H to the circuit, generating a high enough voltage to turn the diode off. If mismatch happens, it connects one storage cell with R_L to the diode with a low voltage that turns it on, and hence discharges the ML.

NMOS-access nvTCAM. A pull-down NMOS is used to connect the cell circuit to the ML. A mismatch/match generates an high/low voltage to the NMOS's gate. It further discharges the ML or keeps its voltage high. The 4T2R cell structure [175] in Figure 5.4 (a) is an example,which is similar with the Diode-access example.

Direct-access nvTCAM. The cells are directly connected to the ML with the access transistors. A mismatch connects cells with R_L to the ML and generate large discharge current. A match connects at least one cell with large resistance R_H to the ML, results in a leaking current but small enough to keep the ML voltage high for a long time. The 2T2R cell structure [164] in Figure 5.4 (d) is an example.

Different categories result in different discharging paths. The ML delay is able to be calculated accordingly. We show the detailed calculation in Section 5.2.2.

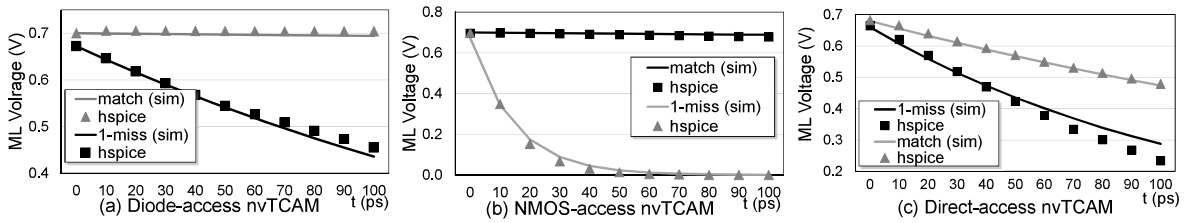

Figure 5.5: Validating ML model in NVSIM-CAM with HSPICE simulations for all three cell categories.

5.2.2 ML delay modeling

We focus on ML developing phase in the three phases for sensing shown in Figure 5.1 (a). For the other two phases, previous methodologies are adaptable for modeling. For example in the precharge phase, since no cell is turned on in that phase, the precharge latency and power is independent with store/search data pattern, and is able to be calculated with ML's RC parameters. We extend the BL delay model in NVSim [158] for ML modeling. Recall that the voltage dividing bitline delay model is described as follows,

$$\tau = R_M C_{\mathrm{ML}} + \frac{1}{2} R_{\mathrm{ML}} C_{\mathrm{ML}}, \quad V_{\mathrm{sense}} = V_s \cdot e^{-\frac{t_{\mathrm{ML}}}{\tau}}, \tag{5.1}$$

where R_M is the equivalent cell resistance, R_{ML} and C_{ML} are the ML's RC parameters. V_s is the precharged voltage, and V_{sense} sets the timing that the SA is enabled to sense. t_{ML} is the ML development delay.

The nvTCAM ML delay modeling is different from normal memory's BL model in two aspects. First, besides calculating the delay, we also need to check the sensing margin constraint for nvTCAM. To this end, we calculate the mismatch that provides the worst case for sensing (with largest R_M^{miss}, usually the case that only one cell misses) in Equation (5.2). In this worse case, the ML discharge is slower than any other mismatch cases, and the ML developed voltage is closest to the match case. Therefore, we calculate the sense margin with it. t_{ML} is defined as

the latency that the ML discharges to V_{sense} in the worst mismatch, as follows,

$$V_{\text{miss}}^{\text{max}} = V_{\text{sense}} = V_s \cdot e^{-\frac{t_{\text{ML}}}{\tau_{\text{miss}}}}, \quad V_{\text{match}} = V_s \cdot e^{-\frac{t_{\text{ML}}}{\tau_{\text{match}}}}, \tag{5.2}$$

where the ML voltage at t_{ML} in the match situation is denoted as V_{match}. The voltage difference between the match (V_{match}) and the worst mismatch ($V_{\text{miss}}^{\text{max}}$) should be larger than the sensing margin.

Second, different from normal memories, the nvTCAM R_M calculation depends on both the cell categories and the match results. For Diode/NMOS-access cells, R_M in match and worst-case mismatch situations are calculated as follows,

$$R_M^{\text{match}} = \frac{R_{\text{off}}}{N}, \quad R_M^{\text{miss}} = R_{\text{on}} || \frac{R_{\text{off}}}{N-1}, \tag{5.3}$$

where R_{on} and R_{off} are the on/off equivalent resistance of a diode/NMOS transistor. N denotes the number of cells that are activated on the ML at one time. Note that different from NMOS-access cells, Diode-access cells have a fast voltage drop before the ML discharges. The fast voltage drop turns the diode off for the match cells. For direct-access cells, the R_M calculation is calculated as follows,

$$R_M^{\text{match}} = \frac{R_H + R_{\text{on}}}{N}, \quad R_M^{\text{miss}} = R_L + R_{\text{on}} || \frac{R_H + R_{\text{on}}}{N-1}, \tag{5.4}$$

where R_L/R_H are the low/high resistance of the NVM cell.

The ML model for the three types of nvTCAMs are then validated with HSPICE simulations, as shown in Figure 5.5. It shows that the model (lines) is able to represent the real delay from the HSPICE (dots) precisely for each category of the nvTCAM cells.

5.2.3 Customize SA

The support for customized SAs is essential. Unlike the well developed SRAM's SA, the NVM (especially nvTCAM) SA designs are very complex and flexible. They are usually designed with special consideration of a particular device or cell structure. Moreover, both of the SA performance and area take an important portion in the overall chip evaluation. For example, Chang*et al.* [176] has shown that by designing a better SA, the overall read latency embraces a 6.3∼8.1× improvement. The SA design is even more important in nvTCAM. There are more SAs in nvTCAM, because SAs are usually not shared among MLs in order to provide a better searching parallelism. Fabricated prototypes show that the SA area takes as large as 13.7%∼30.5% of the overall area [173, 177]. In addition, we also need to count the reference circuit that used for SA reference value generation. However, the reference circuit usually takes large area, and varies from a few dummy cells to a entire data array, in order to tackle process variation for emerging technologies. For example, the PCM based 2T2R nvTCAM [164] has one out of nine array for the reference circuit.

Figure 5.6: NVSIM-CAM's supporting for customized SA.

Even though the SA design is so flexible and important, existing tools only support fixed SA designs with constant parameters. NVSim [158] supports three types of SA designs but still not sufficient. To cope with this problem, NVSIM-CAM supports customized SAs, which provides a interface to import the any SA parameters (either from HSPICE simulation or literature) into the tool, when a more precise result is expected.

Figure 5.6 shows the framework. First, we pre-run the NVSIM-CAM to extract the model of the ML for the following customized SA design. The user could design their SA by HSPICE

or simply gather data from literature, as long as the custom SA's latency and power parameters are given back to the NVSIM-CAM. For the area, if it is not available, a netlist with transistor size is also acceptable, in which case, the NVSIM-CAM estimates the layout footprint [158].

By applying the custom SA design, the validation error (in the case of Table 5.2) reduces from 8% to 4.3%.

5.2.4 Other components

Some of the nvTCAM design require extra peripheral circuits. NVSIM-CAM provides the bit serial accumulator and the priority encoder. We describe the modeling as follows.

Accumulator. The accumulator is used to support bit serial search, which activates only parts of the ML each time and searches the whole word serially. The accumulator gathers the partial matching results and generates the final result for the serial searching. The accumulator circuit includes a register and an AND logic [177]. It also contains a power gating transistor to gate the ML whenever a mismatch happens during the serial searching to save power. By applying the bit serial searching, the number of match leakage path is reduced and hence the sense margin is improved.

Priority Encoder. The priority encoder only generates the lowest address of all the matching entries. The encoder is implemented to facilitate particular applications [166] or to reduce global wiring for the results. The priority encoder is made of a multiple match resolver (MMR) block and a normal encoder block. NVSIM-CAM models the MMRs according to a look ahead 3-level folding design [178]. The basic MMR block (8 entries) is based on dynamic logic [178], and there are two look ahead signals. The basic blocks are serially connected and the look ahead signals are connected in a hierarchical folding style so that the overall latency is reduced from $O(N)$ to $O(\log N)$. We validate NVSIM-CAM's priority encoder model with a 256-bit encoder with the fabricated result [178]. It shows that NVSIM-CAM achieves 12.96%

latency error and 16.18% power error[2].

5.2.5 Validation with fabricated prototypes

In order to validate NVSIM-CAM, we compare the projected result from NVSIM-CAM against the fabricated prototypes. Table 5.2 validates the Diode-access nvTCAM cell model. Table 5.3 works for the NMOS-access model, and Table 5.4 validates the Direct-access model. Nonvolatile technologies of MRAM, PCM, and ReRAM are all examined by those validations. Despite of the limited data we achieve from the literature, NVSIM-CAM manages to achieve a estimation with error around 5%. Even though the error rate is acceptable, we would like to point out that each of those chips is fabricated by a technology with in-house parameters, but NVSIM-CAM is based on PTM. The errors from the technology library could be a major source for the error. Therefore, the significant of this tool lies in relative comparisons, such as DSE shown in the next section.

90nm, Diode, 32-bit, 64-entry			
Metric	*Actual*	*Projected*	*Error*
Area (μm^2):	17118.95[3]	16378.50	-4.3%
L_{Search} (ns):	2.50	2.571	2.6%
E_{Search} (pJ):	–	4.606	–

Table 5.2: Validation: 4T2R MRAM [163].

180nm, NMOS, 32-bit, 128-entry			
Metric	*Actual*	*Projected*	*Error*
Area (μm^2):	–	83157.52	–
L_{Search} (ns):	1.20	1.14	-5.34%
E_{Search} (pJ):	–	51.661	–

Table 5.3: Validation: 4T2R ReRAM [175].

[2] We scale the 600nm data to 32nm for comparison and hence result in unexpected error.

[3] Blank area is excluded.

[4] Test and reference circuit is embedded.

90nm, Direct, 64-bit, 2048-entry, 8-mat			
Metric	*Actual*	*Projected*	*Error*
Area (μm^2):	–[4]	34636.95	–
L_{Search} (ns):	1.90	1.85	-2.5%
E_{Search} (pJ):	–	144.42	–

Table 5.4: Validation: 2T2R PCM [164].

5.3 Design Space Exploration with NVSIM-CAM

In this section, we perform DSE with NVSIM-CAM to show its competency. We first show the search word size's impact. Then, we explore how the scaling of technology affects the nvTCAM design. In the end, we present an overall DES example to show that the optimization is never trivial.

5.3.1 Exploring search word size's impacts

Figure 5.7 shows the search word size's impact on the ML delay and the sense margin. For the configurations, we use 14nm FinFET technology [179] and ReRAM [175]. We set the the ML length the same as the search word size. Three cell structures with 4T2R [163], 4T2R [175], and 2T2R [164] are selected as representatives for the three nvTCAM cell structure categories. The observations are presented as follows.

Diode-access nvTCAM cells provide **support for long search words, but suffer from large search latency**. It is able to minimize the leaking current (I_{match}) if matching, results in the large sense margin and hence good scalability. However, the current when it mismatches (I_{miss}) is also small, and it causes a longer ML discharge delay. Figure 5.7 shows the trend. Its ML delay is larger than other types in most of the cases, and the sense margin almost stays the same, as the word size scales up.

NMOS-access nvTCAM provides **support for long words, and fast search for short**

words. This is because it have both small I_{match} and large I_{miss}. However, the downside is the large cell area. As shown in Figure 5.7, the ML delay gets larger when word gets longer. It is because the large cell area causes a long ML and hence a large RC delay. For sense margin, it is even better than the Diode-access nvTCAMs, because it does not contains a fast voltage drop before the ML development.

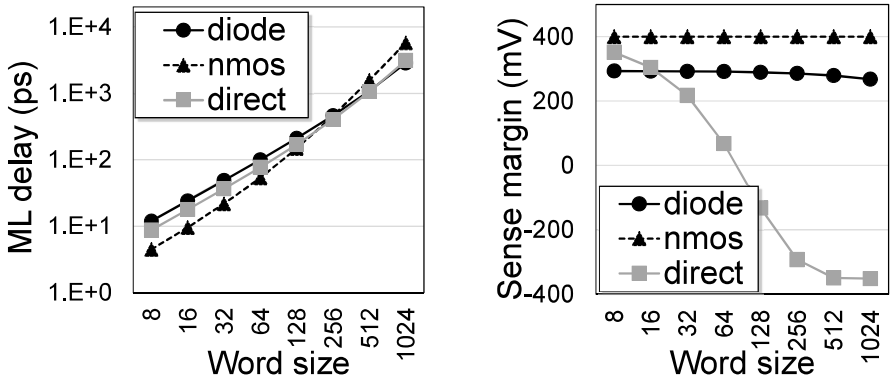

Figure 5.7: Exploring word size's impact on ML delay and sense margin for three types of cell structures.

Direct-access nvTCAM offers best scalability but **cannot afford a long search word**. It benefits from a considerable small cell area. However, the leaking current I_{match} is large, and hence it turns to be a limitation for a long search word. Figure 5.7 shows that when the word size is larger than 32-bit, the sense margin is below 80mV, which is difficult for the SA design. Therefore, Mat partition or bit serial searching is required to support longer search words.

5.3.2 Exploring technology's impact

We study the impact of technology scaling on the performance of nvTCAM design. For the configurations, we use the same cells in Section 5.3.1. The bank contains a single Mat with 64-bit word and 256 entries. We implement the FinFET technology models in NVSIM-CAM and the device parameters are extracted from PTM [179,180]. We observe from NVSIM-CAM that, overall the latency and energy result scales well with the technology development. However, we notice that **for Direct-access nvTCAMs, the technology scaling hurts the sense margin**.

Beyond the 22nm technology, 64-bit word size hits the 80mV sense margin constraint, and hence is difficult for sensing. It brings up the scaling challenges for the area-efficient Direct-access cells. Data encoding schemes [164] or ECC could be the feasible solutions.

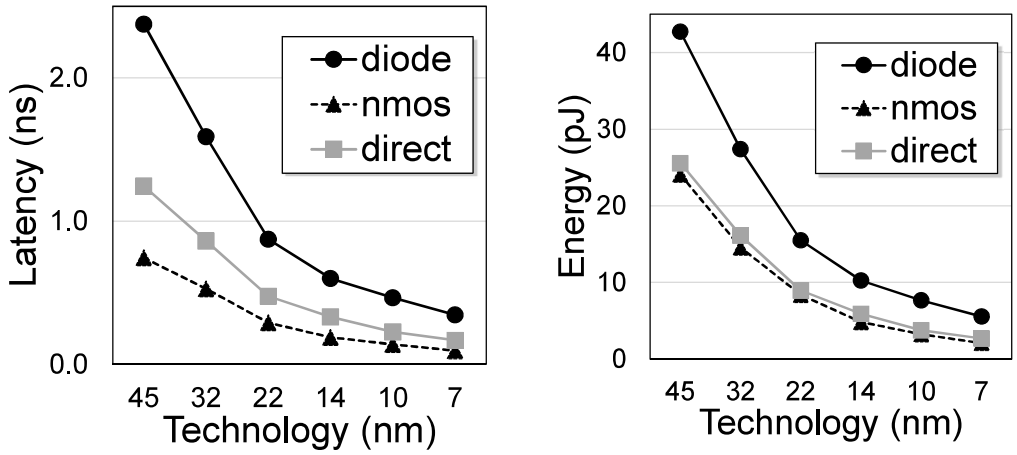

Figure 5.8: Technology's impact on search latency and energy for three types of cell structures.

5.3.3 An overall DSE example

In order to show the nvTCAM DSE is nontrivial, we show an overall DSE example with NVSIM-CAM in Figure 5.5. For the configurations, we set the data organization as a single Mat with 128-bit word and 64 entries. The cell library contains three cells from the three categories, i.e., the 4T2R [163] Diode-access cell, the 3T1R [174] NMOS-access cell, and the 2T2R [164] Direct-access cell. We optimize the design for different targets such as area and search latency.

We observes from the DSE result that **there is no such a design choice that wins for every design target**. For the area optimization, the Direct-access cell providing small cell area is adopted. However, the Direct-access cell suffers from word size scalability challenge. It has to apply bit serial search scheme to achieve the 128-bit search word requirement. As a result, the area optimized design scarifies search latency for a smaller area. For the search latency and energy optimizations, the NMOS-access cell wins, thanks to the large I_{miss} it provides.

	Area Opt.	**L_{Search} Opt.**	**E_{Search} Opt.**	**E_{Write} Opt.**	**Leakage Opt.**
Area (μm^2):	**5746.551**	33327.997	21879.867	27913.421	5746.551
Search Latency (ns)	16.736	**1.332**	5.201	72.992	1121.232
Search Energy (fJ/bit)	27.878	25.533	**23.772**	599.912	1976.749
Write Energy (nJ)	102.992	106.26	168.25	**95.142**	102.992
Leakage (μW)	47.507	2089.383	418.904	63.778	**47.507**
Cell Sturcture	2T2R-direct	3T1R-nmos	3T1R-nmos	4T2R-diode	2T2R-direct
MLC	No	Yes	Yes	No	No
Bit Serial	64-bit	–	–	–	1-bit
Driver Opt.	area-opt	latency-opt	area-opt	area-opt	area-opt

Table 5.5: Design Space Exploration for 14nm ReRAM based 128-bit 64-entry nvTCAM.

However, it is not optimistic considering write energy, since the 3T1R cell has to use the MLC feature, which is more difficult to write. In the end for the leakage optimization, the Direct-access cell is better for two reasons: first, its intra-cell leakage is much smaller; second, the smaller area leads to shorter wires, and hence smaller drivers with smaller leakage.

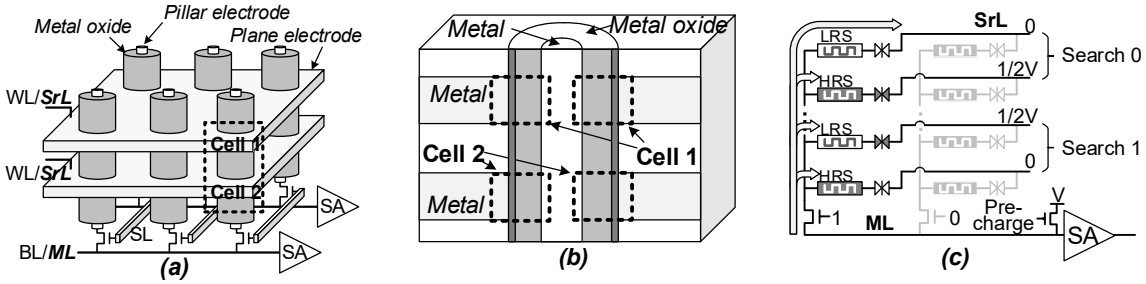

Figure 5.9: (a) 3D Vertical ReRAM based TCAM structure. (b) Cell section. (c) Circuit model when searching.

5.4 3D Vertical ReRAM based TCAM: A case study

In this section, we propose 3DvTCAM, a novel nvTCAM based on the 3D vertical ReRAM (3DVReRAM). As a case study, we explore the benefit and also challenges faced by the proposed high density TCAM, showing that how the NVSIM-CAM tool helps to project the

emerging device developments. In the end, we also discuss that how the projection further facilitates architecture-level innovations.

5.4.1 The 3D Vertical ReRAM TCAM cell

We briefly introduce the background, since the proposed 3DvTCAM is based on the 3DVReRAM [181, 182]. As shown in Figure 5.9 (a), the 3DVReRAM structure is similar with 3D-NAND. Each horizontal plane makes the WL. The vertical pillars with metal oxide around the central metal pillar provide the metal-oxide-metal sandwich structure when contacting with the horizontal planes, and hence build ReRAM cells (a clearer sectional view is shown in Figure 5.9 (b)). The pillar is connected with a access transistor controlled by SL, and then a row of pillars are connected to the BL The signal from a certain pillar is sent to the SA for read. Even though the 3DVReRAM is a ultra dense multi-layer transistor-less design that provides extreme cost efficiency [183], it faces the sneak path problem. Selector based device [184] is proposed to solve the problem by increasing the non-linearity.

We propose the 3DvTCAM based on the 3DVReRAM, as shown in Figure 5.9 (a) and (c). The horizontal plane is used as the SrL, and the BL is used as the ML. A word is stored vertically along a vertical pillar, and a bit is built up with a couple of cells encoding the "0/1/x" states. For searching, a SrL inputs (0, 1/2V)/(1/2V,0). The SL signal only activates one of the pillars connected to the ML at one time. The first column and first row in Table 5.6 shows how 3DvTCAM searches and stores "0/1/x". For normal read and write, it remains the same as 3DVReRAM.

Cell Content	**1 (L, H)**	**0 (H, L)**	**X (H, H)**
Search 0 (0, 0.5V)	$\mathbf{I_{0L}+I_{1H}}$	$I_{0H}+I_{1L}$	$I_{0H}+I_{1H}$
Search 1 (0.5V, 0)	$I_{1L}+I_{0H}$	$\mathbf{I_{1H}+I_{0L}}$	$I_{1H}+I_{0H}$
Search X (0.5V, 0.5V)	$I_{1L}+I_{1H}$	$I_{1H}+I_{1L}$	$I_{1H}+I_{1H}$

Table 5.6: Cell current during ML developing (darker means larger).

In order to make sure the 3DvTCAM work, we prefer the I_{miss} to be large enough to discharge the ML, and the I_{match} small enough to hold the ML voltage and provide enough sense margins. We show the discharge current with all combination of SrL voltage and storage cell resistance as follows,

$$I_{0L} = \frac{V}{R_L},\ I_{0H} = \frac{V}{R_H},\ I_{1L} = \frac{V}{2K_rR_L},\ I_{1H} = \frac{V}{2K_rR_H},$$
$$K_r = \frac{R(\frac{1}{2}V_{\text{read}})}{R(V_{\text{read}})}, \qquad I_{0L} \gg \max\{I_{0H}, I_{1L}, I_{1H}\}, \tag{5.5}$$

where I_{0L} and I_{1L} denotes the current with 0V or 1/2V input at SrL to a cell with R_L resistance, and K_r represents the cell nonlinearity. From the equation we observe that, if provided a large K_r and a large on/off resistance ratio, current I_{0L} will be much smaller than other possible currents and hence ensure the correctness of the TCAM. Table 5.6 shows the cell current of all possible combinations. A darker table cell represents a larger current. Based on this, we show the calculation of the equivalent cell resistance as follows,

$$R_M^{\text{match}} = \frac{2K_rR_L}{N} || \frac{R_H}{N}, \quad R_M^{\text{miss}} = R_L || \frac{2K_rR_H}{2N-1}. \qquad \max I_{\text{match}} = (I_{1L} + I_{0H}) \cdot N, \tag{5.6}$$

$$\min I_{\text{miss}} = I_{1H} \cdot (2N-2) + I_{0L} + I_{1H}. \tag{5.7}$$

Based on the resistance calculation and the ML model in Section 5.2.2, we show the ML discharge calculation for 3DvTCAM as follows,

$$V_o(t) = \frac{V_s(R_M + R_T)}{2R_{\frac{1}{2}V}} + (V_s - \frac{V_s(R_M + R_T)}{2R_{\frac{1}{2}V}})e^{-\frac{t}{\tau}} \tag{5.8}$$

where $R_{\frac{1}{2}V}$ are the overall equivalent resistance connected to the SrLs with 1/2V as input. Different from other nvTCAM, the SrL the 1/2V input makes some of the discharge paths to the 1/2V instead of the ground. We then validate the model with HSPICE simulations in

Figure 5.10, where the lines represent results from NVSIM-CAM and the points represent results from the HSPICE. The results show both mismatch and match scenarios with K_r as 20 and 500. It shows that our model fits well with the real data.

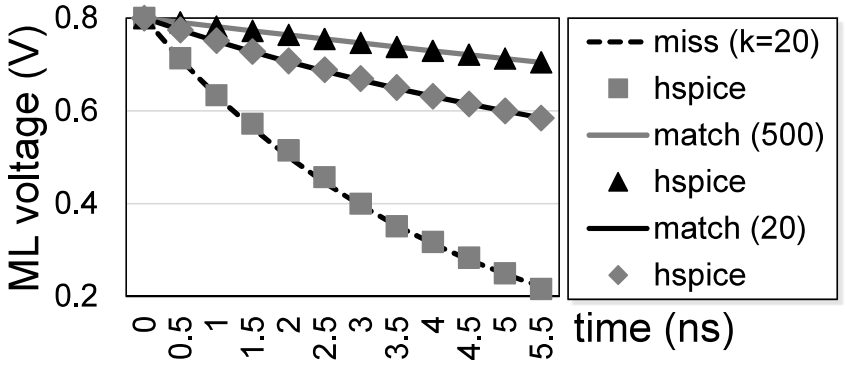

Figure 5.10: ML delay validation with HSPICE.

5.4.2 Exploring feature of the new cell

We project the advantages and disadvantages of the new cell with the help of 3DvTCAM in this subsection. For all the experiment, we apply 14nm FinFET technology and other configurations following the work from Cong*et al.* [181].

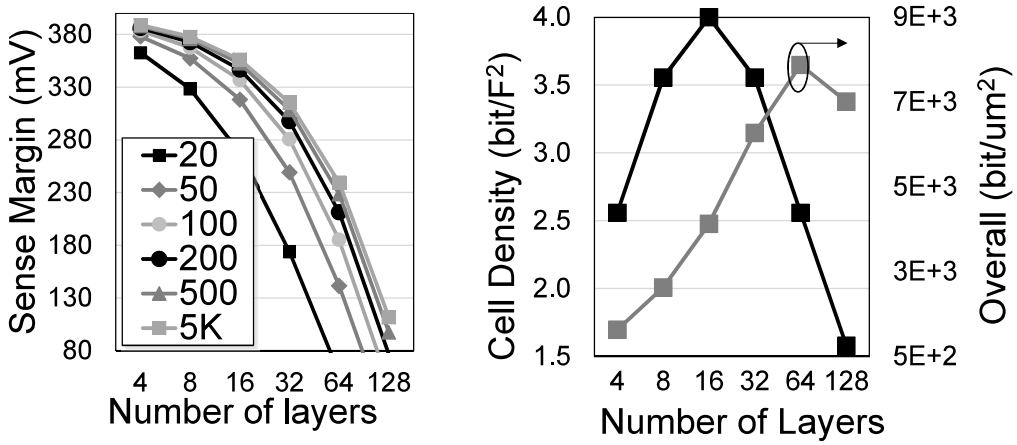

Figure 5.11: Layers v.s. sense margin and density (256×256 array).

We have three observations after studying the impact of the number of layers and the non-linearity factor K_r in Figure 5.11. First, **more layers hurts the sense margin badly**. This is

because a larger number of layers results in more discharge paths during match, and it makes the sense margin drop exponentially. Second, **a larger K_r helps** to provide smaller I_{match} and hence better sense margin. However, beyond $K_r = 50$, increasing K_r barely enlarges the sense margin anymore. This observation shows that **aggressive nonlinearity device technology cannot completely overcome the layer scaling challenge**. Instead, we have to apply bit serial searching or Mat partition design to achieve a longer search word. Third, except for increasing the searching parallelism, **we do not have a motivation for a large number of layers**, from density point of view. Since the aspect ratio is fixed, more layers makes the array area larger, and hence longer wires and larger driver. Figure5.11 shows that the density sweet point is 16 layers while only considering cell density and 64 layers for the overall density.

We also explore the number of layer's impact on the overall search latency and power in Figure 5.12. Although the number of layer (capacity) increases, the latency almost stays the same, which shows 3DvTCAM's **good scalability**. Also, the K_r's impact on latency is negligible. For the power, it doesn't change while layer (capacity) increases, either. The power per bit reduces exponentially.

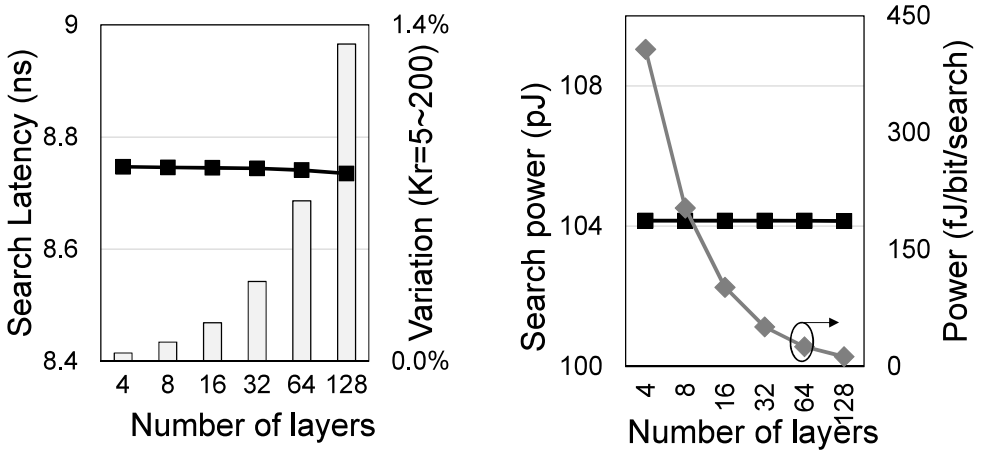

Figure 5.12: Layers v.s. overall latency and power.

We also compare the 3DvTCAM with conventional 2D nvTCAM design in Figure 5.13. For configurations, we use 128-bit search word, 32 layer design with K_r=20. We take the 4T2R

NMOS-access cell [175] for the 2D baseline. For speed, **the 3DvTCAM is slower than a 2D nvTCAM** by ∼58% even when the capacity goes as large as 1GB. However, **3DvTCAM is more energy efficient**. since the power consumption results show ∼2.7× and ∼506× improvement than a 2D nvTCAM of 1MB and 1GB, respectively. That is owed to the good scalability of 3DvTCAM that the length of global wire increases such slower than that of the 2D case. For area comparison, **3DvTCAM provides a ultra dense solution** that saves up to ∼234× area, which is much larger than the 3D layer factor (32 layers). This is owed to the area reduction of the global wires and drivers.

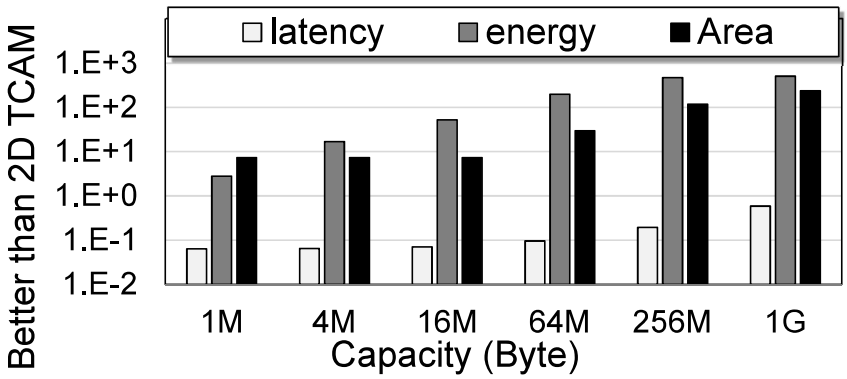

Figure 5.13: Compare 3DvTCAM with 2D design.

5.4.3 Discussion: Turning the projection into architecture innovations

In this subsection, we discuss a potential boarder impact of the NVSIM-CAM tool. We discuss the architecture-level innovation facilitated by the projection result from NVSIM-CAM. In the 3DvTCAM case, we propose a processing-in-storage architecture targeting at energy-efficient acceleration for DNA alignment algorithms.

DNA alignment algorithms like BLAST [185] face challenges, since they need to process huge data sets (one trillion bases and increasing exponentially [186]). The data sets are too large to store in the DRAM main memory. They have to be stored in the storage, such as the 3DReRAM based fast storage system. The alignment algorithm requires searching among the

whole database to pick up the hit sequences for further alignment operations. As shown in Figure 5.14 (a), the processor needs to fetch every raw data from the storage for searching.

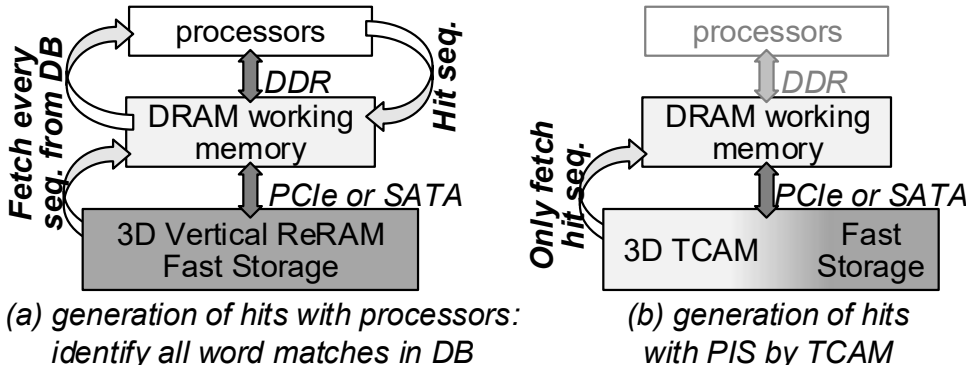

Figure 5.14: Processing-in-storage with 3DvTCAM.

In order to eliminate the unnecessary data movement, we propose a processing-in-storage (PIS) architecture, where part of the fast storage 3DVReRAM is designed as 3DvTCAM. The search operation for hit sequence is then able to be performed inside the storage with the help of the TCAM, as shown in Figure 5.14 (b). By applying the PIS architecture, the data movement is minimized that only a few hit sequences are fetched to the processer. Therefore, a large portion of energy is saved. Moreover, the search operation embraces a larger bandwidth, since it gets rid of the limitation of the narrow data bus, and takes advantage of the massive bank/Mat-level parallelism inside the storage. Conventional SRAM-TCAM or nvTCAM are not competent to support the PIS architecture. Because their large area cannot fit in with the storage class design. This architecture innovation is facilitated by the NVSIM-CAM projection of the 3DvTCAM ($\sim$234$\times$ denser than 2D nvTCAM).

5.5 Conclusion

In order to model and project the ever changing emerging NVM based TCAM design, we propose a circuit-level model and develop a simulation tool, NVSIM-CAM. The tool is able to capture the flexibility of the nvTCAM design and is validated with both HSPICE simulations

and fabricated prototypes. Based on NVSIM-CAM, we perform the DSE for different types of nvTCAM cells. In order to show how NVSIM-CAM helps for early stage projection of potential novel TCAM designs, we explore 3DvTCAM, a proposed 3D vertical ReRAM based TCAM, as a case study. We also discuss NVSIM-CAM's potential for facilitating further architecture innovations.

Chapter 6

DRISA: A DRAM-based Reconfigurable In-Situ Accelerator

To bridge this gap between the computing and the memory, extensive work has been done to explore possible solutions, which can be classified into two categories: The first approach, referred to as the **memory-rich processor**, sticks with the computing-centric architecture while bringing more memory on-chip. For example, modern CPU processors integrate up to 128MB embedded DRAM (eDRAM) caches [187], latest GPU processors integrate up to 16GB 3D High-bandwidth memory(HBM) with 2.5D interposer [188]. This on-chip/in-package memory not only reduces energy-consuming off-chip memory accesses, but also provides higher memory bandwidth, improving system performance. The second approach, referred to as the **compute-capable memory**, switches to the memory-centric processing-in-memory (PIM) architecture. Lightweight processing units are designed in the logic die of 3D stacking memories [57] or in the same DRAM die in 2D cases [50, 189] for near-data computing or in-memory computing. This approach significantly reduces the traffic between the host and memories, and embraces the large internal memory bandwidth.

However, both approaches have limitations. As shown in Figure 6.1, bringing large on-chip

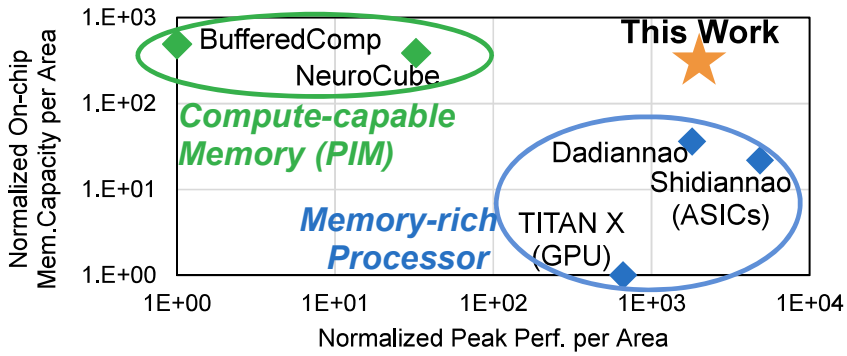

Figure 6.1: The on-chip memory capacity and computing capability of various approaches [10, 50, 78, 190].

memory to the powerful memory-rich processor architectures (the lower right corner) boosts the performance, but the memory capacity is still not enough for data intensive applications. On the other hand, the PIM approaches (the upper left corner) effectively bond more memory to the computing resources, but the performance is not as competitive as GPU/ASICs. For example, Neurocube achieves 132GOPs [10], while the latest GPU can reach 44TOPs [190]. Emerging applications, such as deep learning and bioinformatics (like meta-genome data analysis [191]), are both compute and memory intensive, with a challenging demand for *both* powerful computing and large memory capacity/bandwidth (the upper right corner in Figure 6.1), which may not be satisfied by either of these approaches.

Designing a novel architecture to achieve the goal in the target region in Figure 6.1 is challenging. It is difficult to keep adding more memories to processors, since even the high-density eDRAM suffers from a much larger cell size ($60F^2 - 80F^2$ [16, 192]) than DRAMs ($6F^2$). On the other hand, it is also difficult to improve PIM's performance. For the 3D-based PIM, the area of the processing unit is limited by the logic die's area budget [9]. For the 2D-based PIM, building complex logics with DRAM process technologies results in large area and cost, making the approach unviable for the DRAM industry [62].

The ***goal*** of this chapter is to build a processing unit that provides both high computing per-

formance and large memory capacity/bandwidth (the upper right region in Figure 6.1). To that end, we present a DRAM-based Reconfigurable In-Situ Accelerator architecture, *DRISA*. The accelerator is built using DRAM technology with the majority of the area consisting of DRAM memory arrays, and computes with logic on every memory bitline (BL). By applying the DRAM technology, we achieve the goal of *large memory capacity* for the accelerator. Furthermore, *DRISA*'s *in-situ computing* architecture eliminates unnecessary off-chip accesses and provides ultra-high internal bandwidth. To *avoid the large overhead* caused by building logic with DRAM process, *DRISA* uses simple and serially-computing BL logic. The BL logic has bitwise Boolean logic operations (like `NOR`), which are either performed by the memory cell itself, or by a few add-on gates. *DRISA* can be reconfigured to compute various functions (like additions) by serially running the functionally complete Boolean logical operations with the help of hierarchical internal data movement circuits. Finally, to achieve *high performance* with these simple and serially-computing logic elements, multiple rows, subarrays, and banks are activated simultaneously to provide massive parallelism. We compare four different design options, and present a case study of accelerating the state-of-the-art convolutional neural networks (CNNs). The contributions of this paper are summarized as follows:

- We propose an accelerator architecture, *DRISA*, built with DRAM technology. It provides large on-chip memory and in-situ computing benefits. To reduce the overhead of building logic with DRAM process, we use simple Boolean logic operations for computing but achieve high performance after optimizations.
- A set of circuits and microarchitectures are implemented in *DRISA*, including the BL logic design, the reconfigurable scheme, hierarchical internal data movement circuits, and controllers. Optimizations for unblocking the internal data movement bottlenecks and reducing activation latency and energy are presented to achieve higher performance.
- We use CNN acceleration as a case study to demonstrate the effectiveness of our approach, with resource allocation optimizations. We compare four different *DRISA* designs and

present conclusions that guide efficient *DRISA* design. We also compare *DRISA* with the state-of-the-art ASIC and GPU solutions for the CNN case study.

6.1 Overview

The Key Idea. To implement in-situ computing with large on-chip memory, we build *DRISA* with DRAM process technology. The main challenge is to efficiently build complex logic functions within the DRAM process. We solve this problem by only building simple Boolean logic operations. Figure 6.2 shows a logical overview of *DRISA*. To avoid building complex circuitry in DRAM process technology, we leverage vast, parallel DRAM internal resources to increase computational ability by serially cascading on simple Boolean logic. The bitwise Boolean logic operations are implemented in an efficient manner for each BL. To compute, *DRISA* first opens two rows, performs logical operations using SAs modified with logic and shifters, and then writes back to a result row. It achieves reconfigurability by implementing different sets of functions serially for a desired overall function. However, in order to increase the throughput with multiple DRAM resources, multiple rows, subarrays, and banks need to be activated simultaneously, leading to challenges that we describe next.

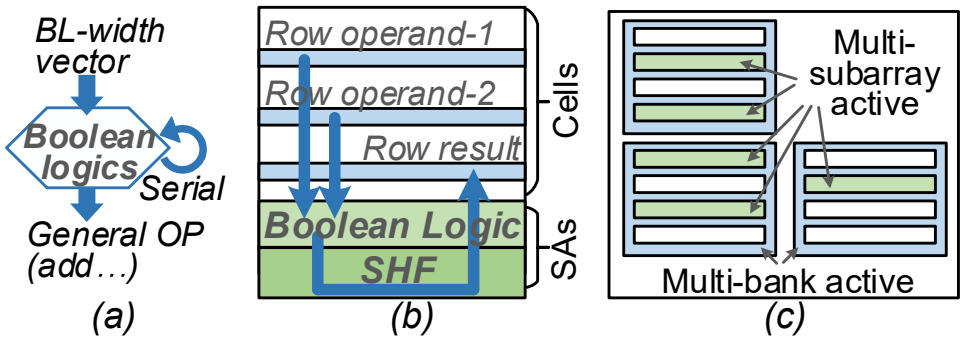

Figure 6.2: A logical overview of *DRISA*. (a) Performing general operations through serially running Boolean logic operations. (b) Implementing Boolean logic operations with SA's help for each BL. (c) Multi-bank/subarray activation for more parallelism.

Challenges and Solutions. Our *DRISA* architecture exploits massive DRAM parallelism and

achieves large computational throughput for in-situ reconfigurable computing. However, new design methodologies are required to achieve high performance. We outline the challenges and our contributions to address them:

Challenge-1:*Achieving high performance with the simple and serial logic elements.* We target *DRISA* as an accelerator instead of as host memory to avoid tight area constraints, and hence we can optimize it for high performance. *DRISA* requires simultaneous activation of multiple subarrays and banks to provide large parallelism and thereby large computational throughput. To solve this, we propose bank reorganization to enable activating multiple rows (Section 6.2.4).

Challenge-2:*Unblocking the internal data movement bottleneck.* We propose group/bank buffers to isolate local movements in DRAM and enable moving multiple data buffers in parallel (Section 6.2.2). We also reorganize the bank to reduce data collisions on the shared data bus by designing a hierarchical bus (Section 6.2.4).

Challenge-3:*Optimizing ACT to reduce its latency and energy.* Activation (ACT) is a basic step for *DRISA* computing. Directly adopting a DRAM ACT mechanism results in large latency and energy overheads. Our bank reorganization makes WLs/BLs shorter (Section 6.2.4). We also present split computing and storage array regions, μ-operations, and local instruction decoding to save latency and energy on ACT (Section 6.2.3).

6.2 DRISA Architecture

We show *DRISA*'s architecture design in Figure 6.3. It inherits most aspects of standard DRAM design. However, one more hierarchy, called a group, is added between the hierarchy of the chip and bank. Groups are connected with a bus (*gBus*), and controlled by the chip-level controllers (*cCtrl*). Within a bank, bank-level decoders are modified as controllers (*bCtrl*). Bank buffers (*bBuf*) are added to help data movements. In a subarray, a subarray-

level controller (*sCtrl*) is added. In a mat, the cell array is split into two regions, for storage and computing, respectively. The SA is modified to support the computing. Extra hardware to support data movements is also added. A mat is logically partitioned vertically into lanes, and each lane is equivalent to an n-bit processing unit. We elaborate on the design details and justify the design choices in the rest of this section.

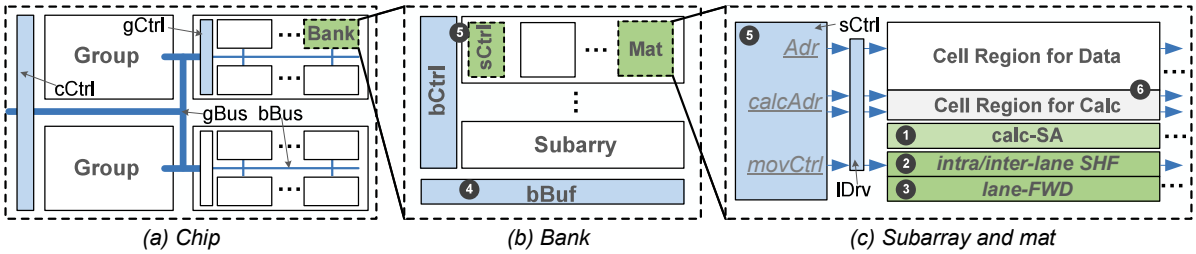

Figure 6.3: *DRISA* architecture design. (Glossary - cCtrl/gCtrl/bCtrl/sCtrl: chip/group/bank/subarray controller. gBus/bBus: group/bank bus. bBuf: bank buffer. lDrv: local driver. SHF: shifter. FWD: forwarding.)

6.2.1 Microarchitecture for Computing

The basic operating units for the reconfigurable computing are the Boolean logic operations and the shifters (❶ and ❷ in Figure 6.3, respectively). For the logic part, there are two approaches that can make DRAMs computing-capable: the *3T1C* solution and the *1T1C* solution. Both of these approaches share the same shifter design.

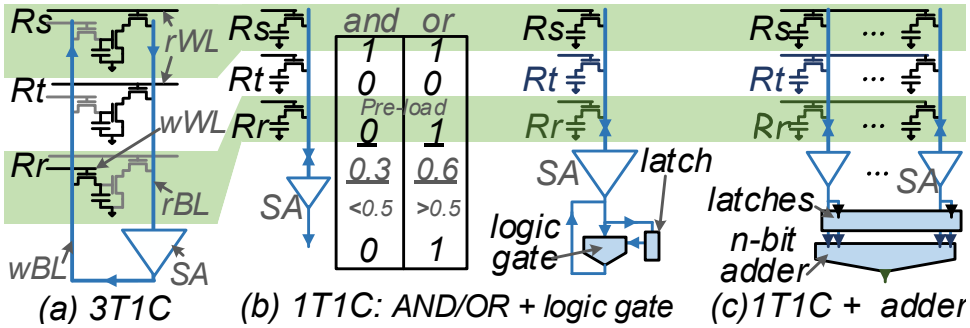

Figure 6.4: Cell structures. (Glossary - Rs/Rt/Rr: Operand source row s and source row t, result row r.)

3T1C-based computing: This design changes standard DRAM cells to *3T1C* and uses cells themselves for computing. Therefore no other circuits are required. *3T1C* cell was used in early DRAM designs [193]. As shown in Figure 6.4(a), both WLs and BLs are separated as two lines for read and write, respectively. The cell includes two separated read/write access transistors, and an extra transistor that decouples the capacitor from the read BL. The third transistor also connects the cell in a `NOR` style on the read BL. Therefore, the *3T1C* cell naturally performs `NOR` logic (`NOR` itself is functionally complete) on BL without any extra design changes.

1T1C-based computing: This design keeps the standard DRAM cells unchanged, but uses extra circuits attached to the SAs for computing. There are two types of computing. First, it calculates `AND` and `OR` using the method proposed by Seshadri *et al.* [72] (on the left part of Figure 6.4(b)). The result row is pre-stored with 0 (for `AND`) or 1 (for `OR`), and three rows are activated simultaneously. Then, after charge sharing (shown as 0.3 and 0.6 in the figure), the SA readout is the logic result, and the result is restored to the result row during the row closing. Note that this operation will also destroy the operand rows, so a row copy before the operations is required. However, the problem is that `AND`/`OR` alone are not logically complete. Therefore, the second types of computing is demanded. *DRISA* calculates other logic function like `NOT` to achieve logical completeness, as shown in the right part of Figure 6.4(b). Extra circuits for a latch and logic gates (to perform one or some of Boolean logic operations) are added. The operand *Row s* (R_s) is activated first, and the data is stored in the latch. Then, the operand *Row t* (R_t) is activated, and its data, along with the data in the latch, is fed into the logic gate. The result is then read out or restored to the result row. Taking the *1T1C*-based solution to an extreme scenario, we can also design a n-bit adder circuit for n-bit BLs, as shown in Figure 6.4(c).

Both of the *3T1C*- and *1T1C*-based solutions make each BL computing-capable. This architecture makes in-situ computing possible, since the memory cell and the BL logic are tightly coupled.

Circuits for intra-lane SHF: Shifters are required in our architecture because the bitwise Boolean logic operations only performs logical operations but not arbitrary data movement. Shifters are designed for data shuffling, thereby enabling general-purpose computing. Figure 6.5(a) shows the shifter circuits that take a 4-bit lane as an example. The circuits are located at ❷ in Figure 6.3. Figure 6.5(b) and (c) show the examples for left shift-2 and right shift-3. The circuit design is similar to a barrel shifter. For an n-bit lane, we implement arithmetic 1/(n-1)-bit right shifts. We also implement 1/$exp2$-bit left shifts. The left shift is either a logical or arithmetic shift. We design filling lines that can fill in 0/1 accordingly. This design can then perform arbitrary shifts by running serially.

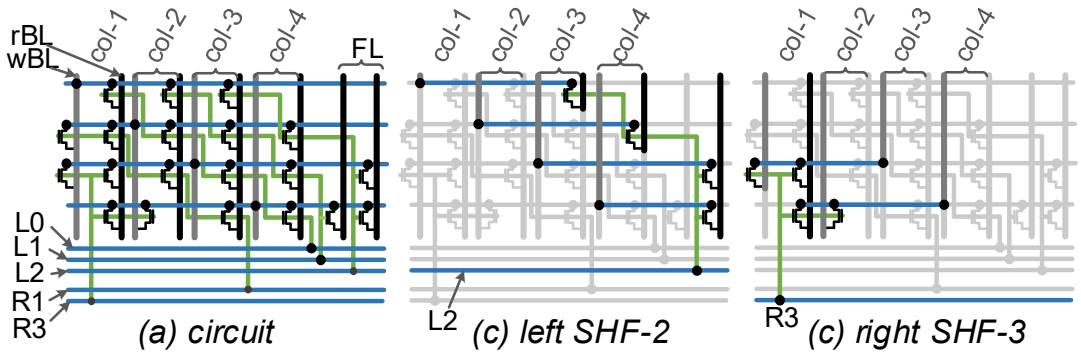

Figure 6.5: The shift (SHF) circuits and examples. (Wires - Green: Poly. Black/Gray: Metal-1. Blue: Metal-2. Glossary - Lx/Rx: Left/Right shift x. rBL/wBL: read/write bitline. FL: Filling line.)

Our shifter design is optimized for common cases. Even though 1-bit left/right shift alone is functionally complete, we design extra (n-1) bit right shift circuits to cover the common case that makes the whole lane all-zero/one according to its sign bit. We also design special $exp2$-bit left shift circuits to cover the most common shifts in full adders, making it 11% faster. We do not design special circuits for every possible shift cases, thus saving 60% shifter circuit area, 84% shifter latency, and 52% shifter energy, without any performance degradation for all μ-operations defined in Table 6.1.

Reconfigurable computing: With the functionally complete logic and shifters, *DRISA* can theoretically accomplish any operations by running serially. In Figure 6.6, the *3T1C* one-cycle

NOR logic is used as an example to show how *DRISA* supports some frequently used operations, i.e., selection (*SEL*), addition (*ADD*), and multiplication (*MUL*). We use the notation shown in the upper-left corner of Figure 6.6, where the blocks denote rows in DRAMs, white rows are inputs, blue rows are intermediate terms, and green rows are outputs. Arrows connect the sources and result of a NOR logic.

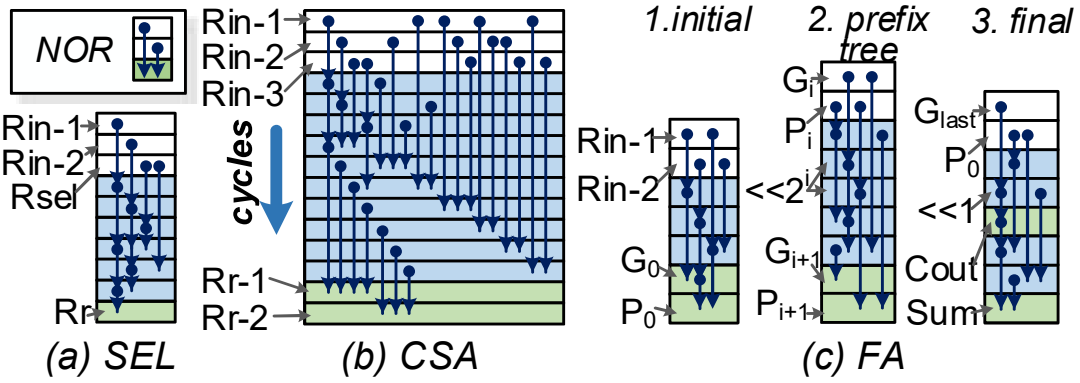

Figure 6.6: Building operations with basic Boolean logic operations.

To calculate *SEL*, we duplicate the selector 0/1 to a whole row. Then the *SEL* is broken down into NOR logic step by step, as shown in Figure 6.6(a), which takes 7 cycles and 6 intermediate terms in total. For adders, we show both a carry-save adder (*CSA*), which has three inputs and two outputs without a carry-out, and a full adder (*FA*), which has two inputs and one output and a carry-out. Breaking down the *CSA* into NOR logics, we get Figure 6.6(b). The *FA* calculation is shown in Figure 6.6(c). There are three steps: Step one initializes and generates partial terms P_0 and G_0. Step two follows the prefix tree [194] logics, generates P_i and G_i, and iterates $log(n)$ times for an n-bit adder. During this step, left shifting is required. Step three finally generates the sum and the carry-out. For the *MUL*, the calculation depends on the multiplier's bit width. If the multiplier is binary, the *MUL* collapses as a XNOR logic. Otherwise, *MUL* is calculated by generating partial terms with shifter and *SEL* and then sums all these partial terms with both *CSA* and *FA*.

Figure 6.7 (left) shows the cycles to compute *CSA* and *FA*. *DRISA* favors CSA because as

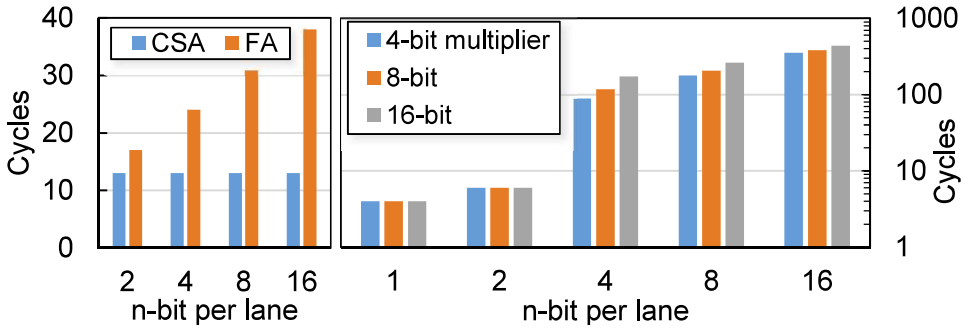

Figure 6.7: *3T1C* computing cycles for *ADD* and *MUL*.

each lane's bit width increases (x-axis), *CSA* retains a constant latency. Figure 6.7 (right) shows *MUL* with different lane counts and multiplier widths, both in bits. For 1-bit and 2-bit *MUL*, it takes only 5-6 cycles. However, if the multiplier has more than 3 bits, it takes hundreds of cycles for computing.

6.2.2 Microarchitecture for Data Movement

A general purpose processor requires flexible data movement. The shift circuits mentioned above only cover inter-lane movement. We design circuits for data movement between lanes in this subsection, in order to have a functionally complete hierarchical shift solution. Specifically, within the subarrays, we design *inter-lane SHF* circuits. Between subarrays, we have improved *RowClone* [71]. We also have the *lane-FWD* circuits, to move data from and to arbitrary lanes.

Circuits for inter-lane SHF: *Inter-lane SHF* (❷ in Figure 6.3) shifts a row from one lane to its adjacent lane. The circuits are shown in Figure 6.8(a). The shifter contains ping/pong phases. We choose this two-step shift because it saves area, since all lanes share the same shift data wire (the two upper blue wires in Figure 6.3).

Figure 6.8(b) and (c) show the example of left shift. If shifting right, *ping* and *odd* are first selected for the ping-phase, and then *pong* and *even* for the pong-phase.

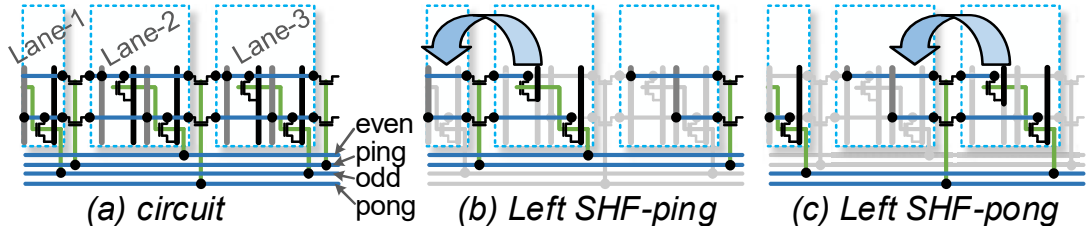

Figure 6.8: The *inter-lane SHF* circuits and examples.

Circuits for lane-FWD: *Lane-FWD* (❸ in Figure 6.3) supports random read/write from/to an arbitrary lane. Figure 6.9(a) shows the circuit design. Figure 6.9(b) and (c) show examples of reading Lane-1 and writing Lane-2, respectively.

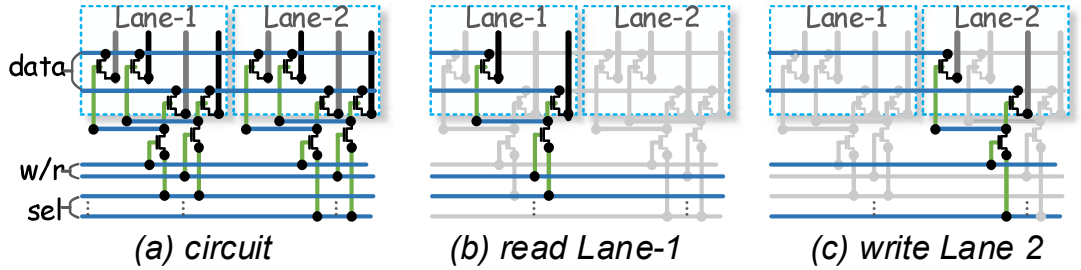

Figure 6.9: The lane-FWD circuits and examples

Enhanced RowClone with bank buffer: We improve upon the existing *RowClone* [71] technique in *DRISA*. Besides original row-to-row copy, *DRISA* can choose either to copy the whole row, or to repeatedly copy the data from a certain lane. Furthermore, we add bank buffers (❹ in Figure 6.3) in order to tackle Challenge-2 (preventing data movement from becoming the bottleneck). The limitation of *RowClone* lies in the shared memory data bus. Although multiple subarrays/banks work simultaneously in *DRISA*, the initial *RowClone* only works with two subarrays at one time since it utilizes the shared data bus between banks. Bank buffers, which are implemented by registers, isolate intra-bank *RowClone*s from other banks, so that multiple intra-bank *RowClone*s work in parallel in different banks.

6.2.3 Microarchitecture for Controllers

Instruction design: We abstract an instruction set shown in Table 6.1. The instruction follows the R-format in MIPS [195], which contains the opcode, the address for two (or possibly one or three) input rows and the output row, and the *funct* code that describes detailed controls. *DRISA* has the basic instructions for Boolean logic operations and data movement mentioned earlier, as shown in the left side of Table 6.1. In addition, there are also μ-operations, including frequently used functions (*SEL, ADD, MUL, MAX,* etc). Next, bulk data copy and also compute reductions with addition/maximal/minimal operators are supported. Finally, a vector-wise inner-product operation is also supported. Note that instructions like control transfer are carried out by the host and therefore not included in Table 6.1.

Basic Instr.	**opcode**	**funct**	**μOps.**	**opcode**	**funct**
	Logic (NOR etc)	type of logic		*Calc. (FA etc)*	N/A
	SHF	L/R, offset, filling		*Bulk-copy*	length, stripe
	Lane-SHF	L/R, offset		*R-SUM*	length
	Dup-copy	N/A		*R-MAX/MIN*	length
	Copy	bank/group/chip		*Inner-product*	length

Table 6.1: The basic instructions and μ-operations.

Multi-level controllers: *DRISA* has four levels of controllers (❺ in Figure 6.3): chip, group, bank, and subarray-level controllers. They support simultaneous multi-subarray/bank activation for better parallelism. The first two levels (chip/group) of controllers are essentially decoders, but they can also help with data movement. The bank-level controllers decode the instructions. They convert the instructions and μ-operations into addresses, vector lengths, and control codes, and then send them to the controllers in the active subarray. The subarray controller consists of address latches, local decoders, and counters. The address latches are essential for multi-subarray activation [196]. The counters are used for continuously updating addresses to local decoders for the bulk-style μ-operations.

Split array regions: The cell array is split into the data region and the compute region (❻ in

Figure 6.3). They share BLs and SAs, but have separate decoders in the subarray controllers. This separation reduces the area and performance overhead while supporting multi-row activations (required by computing in Section 6.2.1). A strong decoder that activates multiple rows in one cycle is costly. On the other hand, designing a latch for each local WL and serially decoding for the active rows [197] wastes too much latency and energy. Instead, in the split array case, the data region that has most of the cells does not need multi-row activation. The compute region that stores the intermediate data (blue rows in Figure 6.6) only contains a few (typically 16) rows. Designing a strong one-cycle decoder is much easier.

Without the split regions, a strong one-cycle decoder takes 204.3% area overhead (compared with a normal 256 fan-out decoder). On the other hand, the serial solution only takes 4.3% area overhead, but results in 10.8% peak performance degradation. After adapting the split cell region idea, we have one-cycle decoding with only 19.02% area overhead.

6.2.4 Optimizing Bank Reorganization

We reorganize bank/array in *DRISA* to optimize for performance and energy efficiency. Conventionally in DRAM memories, bank/array organizations are optimized for memory density. We switch the optimization objective in *DRISA* since we are now designing accelerators instead of memories.

Improving the parallelism: We have to improve parallelism to achieve high performance. It takes *DRISA* around 30 cycles for *FA*. If considering each cycle as `tRC`, *DRISA*'s adder runs as slow as ∼1MHz. To overcome Challenge-1, we present two techniques: (1) *DRISA* simultaneously activates multiple subarrays in multiple banks.

To achieve this, each subarray and bank has their independent controllers with latches. Previous work [196] shows such modification incurs ignorable area overhead. The detailed controller design is shown in Section 6.2.3. (2) Furthermore, *DRISA* activates more rows at

one time. We reorganize the banks and subarray by making the subarray smaller (few number of rows) but with larger quantity. Therefore, the number of rows/subarrays that can be active simultaneously is increased. Later, Figure 6.11 shows that it costs 58% more area but gains 4× more parallelism. This is worthwhile when optimizing for performance per area.

However, there are limitations for out parallelism improvements. First, the power budget is a hard constraint. Second, modern DRAMs use open-BL architecture, where the SA works with a differential sensing mechanism. It needs an idle adjacent subarray as a reference. Therefore, adjacent subarrays cannot be activated simultaneously, i.e., we can at most activate 50% of all the subarrays. Third, more compute parallelism is not necessarily better, since internal data movement may become the bottleneck. Section 6.4.2 shows that the subarray's effective utilization can be as low as 10%, due to data blocking.

Unblocking the data movement bottleneck: For Challenge-2, we propose four techniques. (1) We design bank/group buffers in Section 6.2.2, in order to isolate local movements and parallelizing them. (2) We reorganize the banks by reducing the number of subarrays per bank while increasing the bank quantity. Fewer subarrays per bank reduces the data conjunctions for intra-bank data movements. (3) We add groups and group buffers, so that inter-bank data movements inside different groups work in parallel. (4) We propose the bulk-style μ-operations (Section 6.2.3) to reduce instruction data movement (and decoding). One μ-operation only requires one-time data transfer and decoding, and then the local controller auto-generates bulk instructions.

Figure 6.10 (left) shows that reducing the number of subarrays per bank from 1024 to 64 achieves 576× better performance with only 1.5% area overhead. This increase of resource utilization is the evidence that this performance gain comes from faster data movements. Figure 6.10 (right) shows that as the number of groups increases from 1 to 16, it achieves 3.65× better performance. In addition, by adapting μ-operations, we achieve another 22.94% and 3.43% reduction on latency and energy, respectively.

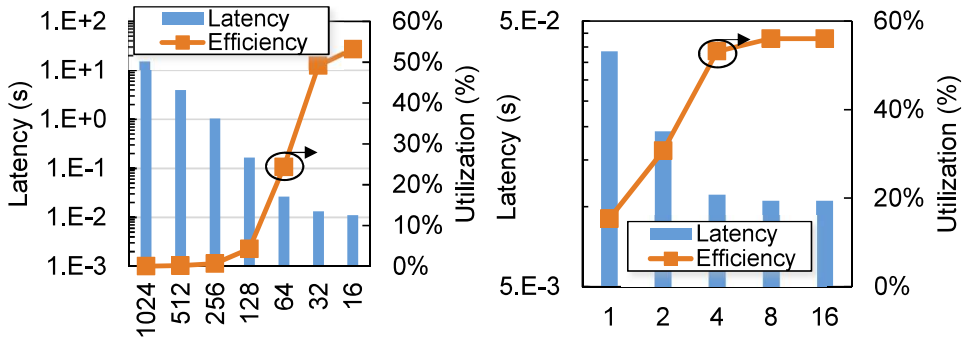

Figure 6.10: The latency and resource utilization for real application (VGG-16 on *1T1C-mixed*). Left: Impact of number of subarrays per bank. Right: Impact of number of groups.

Optimizing ACT latency and energy: Reducing the ACT overhead is essential in *DRISA*. In a typical DRAM, a activation cycle (tRC) takes 46ns [198] and 24.9% of the memory power consumption [199]. Such a long clock period and large energy consumption are challenging for *DRISA*, which computes serially with multiple cycles per operation. To tackle this Challenge-3, we propose three techniques. (1) We reorganize arrays with shorter BLs (fewer columns) and WLs (fewer rows) [200]. Shorter BLs and WLs result in smaller RC and hence smaller latency and energy, as well as easier and faster decoding. (2) We include the extra group hierarchy between the chip and banks. This makes the bus become hierarchical and hence more scalable, so that the bus overhead is reduced. (3) The μ-operations also help; one μ-operation may contain hundreds of ACT instructions, but it is only transferred on the global bus for one instance. Average ACT cost is then reduced.

Figure 6.11 shows the ACT energy and latency for four cases (A to D) with area results. *Case-A* is the bank organization in the original DRAM memory. We observe that the BL dominates both the latency (41%) and energy (99%) in ACTs. This is the motivation for our first technique that switches to shorter WL/BLs, as shown by *Case-B*. 4$\times$ shorter BLs and 8$\times$ shorter WLs yields 62.5% latency and 66.9% energy/bit reduction. The downside is 58% larger area. However, this is acceptable since *DRISA* is an accelerator optimized for performance, not

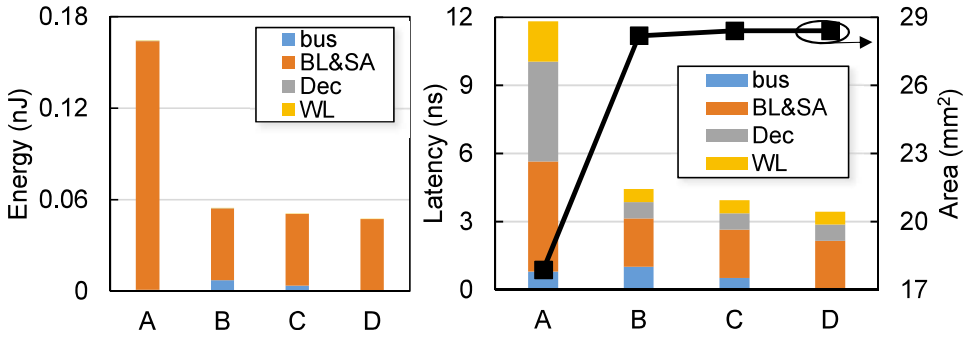

Figure 6.11: Reducing the activation energy and latency (A: 8 banks, 1K-16K subarray; B: 512 banks, 256-2K subarray; C: B with group (128 banks per group); D: C with local decoding.)

a memory optimized for density. *Case-C* shows the second technique's benefit. By having the group hierarchy, it reduces the latency and energy spent on the bus by 49.9% and 50%, respectively. *Case-D* shows that the third technique is effective. By adapting μ-operations, it further saves 12.7% latency and 6.9% energy.

6.2.5 System Integration

We briefly discuss how to integrate *DRISA* into the system (detailed software/system support is out of the scope of this paper and planned as future work). Considering that *DRISA* is a co-processor or accelerator instead of a memory, it is integrated in the same manner as a GPU or FPGA, not a PIM system.

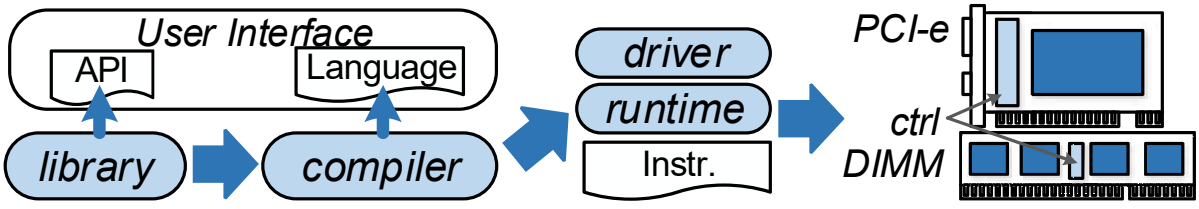

Figure 6.12: Integrating *DRISA* into the System.

For the software component , *DRISA* follows the same programming model as Automata [85], GPU, or FPGA. *DRISA* requires a special programming language or framework, like CUDA

for GPU and AP SDK [201] for Automata. A corresponding compiler is also necessary. In order to map general purpose program onto *DRISA*, programmers can treat *DRISA* as a multiple issued vector machine, similar to programming with AXE/SSE. To make programming easier, application-specific APIs should also be provided to the users. The *DRISA* compiler compiles the high-level descriptions into *DRISA* instructions, and it works along with the driver and the runtime engine to offload tasks onto *DRISA*, transfer data, and control *DRISA* to finish the task.

For the hardware, both PCIe and DIMM solutions are applicable. PCIe integration (like GPU, FPGA, Automata [85]) provides sufficient power delivery and well-developed control system. The DIMM solution (like AC-DIMM [202]) requires *DRISA* to support a DDR-like interface but function like an accelerator. This solution is still within active research. The advantages of the DIMM solution is simplicity for scaling-out, considering that the number of DIMM slots is much more than PCIe's. The downside is that the power budget for each slot is low, which limits the performance of every individual *DRISA*. For both of those solutions, there is an SoC controller on board, which supports inter-chip communications.

To scale-out for applications with larger data sets than *DRISA*'s memory capacity, *DRISA* follows the solution of multi-GPUs, which leaves the partitioning job to programmers or frameworks (like Torch [203]).

6.2.6 Discussion

Limitations. *DRISA* is not suitable for floating point calculation, even though it is capable of this functionality. The limitation lies in the lock-step within a subarray, since all lanes in one subarray share the same controller. This dramatically hurts the performance of floating point operations, because the internal control of floating point calculation is data dependent. For example, the shift for significands alignment is based on the subtraction result of exponential biases. Therefore, every lane potentially requires a different bit shift, which is not supported

by the lock-step architecture. Instead, a single floating point operation is required to run on a whole subarray instead of a single lane, massively reducing lane-level parallelism.

Process variation. Multiple experimental studies and patents have already established the viability of 3T1C and 3 row activation designs in DRAM, even in the presence of manufacturing variations. In addition, Micron's Automata has demonstrated the feasibility of heterogeneous circuits design with DRAM process technology. Specifically for DRISA, we examine the impact of these variations and general DRAM challenges, since the computed results depends on the nominal operation of the proposed bitwise logic computational the DRAM cell level.

The first challenge is with variable cell voltage level due to charge-leak, thereby impacting the charge-sharing operation with bitline. *DRISA* is not impacted by this challenge since every bitwise logical operation is preceded by a data-copy to the source and destination rows (Rs, Rt). This naturally constitutes DRAM restore operation, charging the voltage levels to the cell value and offsets any charge leaking that could affect cell-sharing operation with three-rows. In addition, *DRISA* can tolerate 8ms retention time, compared with 64ms in commodity DRAM, which makes it even more robust.

The second challenge relates to variation in the cell-level capacitances that could affect the bitwise logical operation due to strong or weak capacitances. Fortunately, *DRISA* has a 4-point approach that ensures strong immunity to these challenges:

• First, process variations' impact in the context of multi-row activation in 1T1C-based designs were already examined in detailed by Buddy-RAM [204]. The impact was shown to be minimal, affecting special patterns of cell values with specific cell capacitance strengths. However, even for these cases, measurements show that the logic function sustains even with $\pm$20% process (40% cell-cell) variation. For the case of *DRISA* design, the theoretical limits allow $\pm$33% capacitance variation for a 3 wordline design, but our evaluation places a tighter bound of $\pm$28% to ensure safe margin for sense amplifier. DRAM systems today have a process variation much less than this tolerable $\pm$28% (56% cell to cell). For example, the capaci-

tance difference between two generation is only 10∼15% [205]. Also, industry inventions on new DRAM capacitance structures significantly increase the capacitance of DRAM cell and therefore reduce the impact of variation [205].

• Next, as discussed in Section 6.2.1, *DRISA* array structure is fundamentally different and is tailored to be an accelerator with 16× smaller array size than commodity DRAM (256-by-2048 v.s. 1024-by-8096). This results in a proportionately smaller number of cells sharing the local bit lines which is therefore significantly shorter. This in turn improves the ratio between local bitline and the cell capacitances and therefore the sensing ability beyond commodity DRAM and other existing approaches.

• The impact of process variation can also be handled at the circuit level. Unlike cost-optimized commodity DRAMs, *DRISA* can tune the SA design and spend more area for extra reliability [206]. Also, in 1T1C, we can also use the logic gates (Figure 6.4(b) right side) for pure digital computing, where no multi-row ACT or SA is involved, immune to process variation.

• Finally, we can apply architecture-level method to avoid defective modules. For example, *DRISA* can examine the capacitance variation during the manufacturing and testing phase. Cells that are detected to contain more than the acceptable variation will be masked and instead replaced with spare row and column by using already prevalent fusing techniques, consistent with existing DRAM. Since the spare DRAM cells are already implemented in the state-of-the-art DRAM chips, there is no extra hardware overhead. With a high threshold, we expect this to produce similar yield as a normal process. Another way is to apply defect-aware mapping method similar to ArchSheild [207].

The final challenge relates to DRAM yield as a result of co-existence of logical elements and the DRAM cells. By virtue of design, these logical components i.e. shifter, are integrated after the sense amplification stage and do not interfere with the highly-optimized DRAM cell level, IO lines' layout. As such, it does not affect DRAM yield either.

***DRISA* is an accelerator, not host memory.** We position *DRISA* as an accelerator or co-

processor instead of as part of the host memory, because memory designs are extremely optimized for low-cost, but our target is to build a high performance accelerator. Conventionally, DRAM is cost sensitive and is unlikely to be changed. However, by avoiding being a part of the host memory, *DRISA*'s area is not the primary optimization priority, and we can trade-off area overhead for better performance. As we change the design goal to high performance, *DRISA* has greater room to re-design the DRAM arrays and peripheral circuits for high performance parallel computation. In addition, we treat *DRISA*'s memory space similar to the device memory or scratchpad memory on GPU/FPGA/Automata, which avoids issues in data coherence, data reorganization, and address translation if using *DRISA* as host memory.

6.3 Accelerating CNN: A Case Study

In this section, we map CNNs (inference) on *DRISA* as a case study. Note that the purpose of the case study is to show the methodology of application mapping. *DRISA* is not limited to only CNN applications.

6.3.1 Quantizing CNN for *DRISA*

NN algorithms are originally based on floating point calculations. NN data quantization work [208–212] helps to tackle the challenge by quantizing the floating point activation data and weight into fewer bits of fixed-point data and then retraining the NN to reduce the accuracy loss. Furthermore, research studies have found it is even possible to quantize weights into binary data [213–215]. After proper training, BWN [215] (binary weight, floating point activation data) shows 0%, 8.5%, and 5.8% top-1 accuracy degradations, compared with all floating point golden models on AlexNet [216], ResNet-18 [217], and GoogleNet [218], respectively. Even more aggressively, BNN [214] and XNOR-Net [215] also binarize the activation data. It

shows Top-1 accuracy degradation of 12.4% and 18.1% on AlexNet and ResNet-18, respectively.

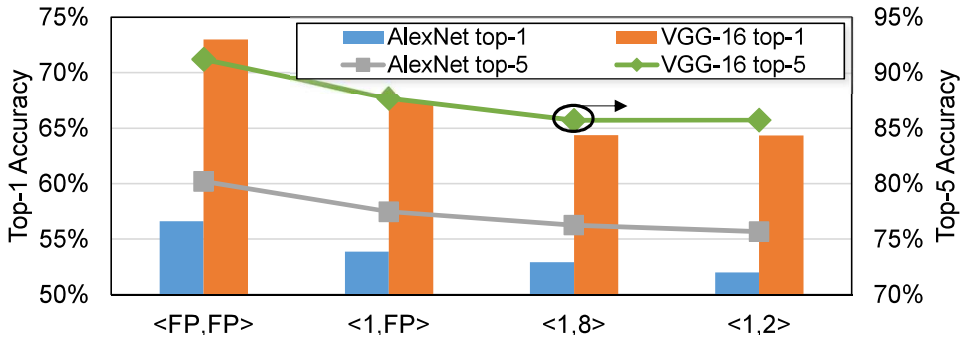

Figure 6.13: Training *DRISA*-friendly CNN on ImageNet (¡x,n¿ denotes x-bit weights and n-bit fixed point activation data).

We apply 1-bit weights with 8-bit activations (¡1,8¿) for *DRISA*. This is because *DRISA* runs faster if one of the multipliers have 1 or 2 bits, while it is insensitive to the other operand (Figure 6.6). Therefore, it is not necessary to adopt the extreme BNN/XNOR-Net quantization cases (both binary weight and activation data). Instead, we need an eclectic way with binary weights and fixed-point activation data (between BWN and BNN/XOR-NeT). Specifically, we use binary weights for all layers (including the first and last layers), and use shifter for approximate weight scaling. Figure 6.13 shows our training result for AlexNet and VGG-16 [219] on ImageNet [220]. Note that Figure 6.13 shows a "worst" case scenario, since we have not applied fine tuning for our training. More training epochs, larger batch sizes, image augmentation [79], better initialization [209], and better learning rate tuning will effectively increase the accuracy. It has much larger potential since BWN (¡1,FP¿) [215] has been reported as 56.8% top-1 accuracy, which sets a upper bound of our accuracy.

6.3.2 Resource Allocation

DRISA provides row-to-row operations, which are treated as SIMD vector instructions (like AVX [221]). In CNN applications, more than 99.9% of the operations can be aggregated as

vector operations [222]. For these scalar operations or vector operations that are shorter than *DRISA*'s row size, we fill zeros in any unused slots.

In order to efficiently utilize *DRISA* hardware, we need to optimize resource allocations for CNNs. CNNs have lots of inherent parallelism (e.g., batch, feature map, and pixel-level parallelism), and so does the *DRISA* architecture (i.e., group, bank, subarray and lane-level parallelism). How to effectively allocate the hardware resource to the application is challenging. We follow two design philosophies: (1) The data movement and computation tasks should be balanced, in order to achieve the highest efficiency. (2) Since *DRISA* favors *CSA* compared to *FA*, we should avoid using *FA* (Figure 6.7).

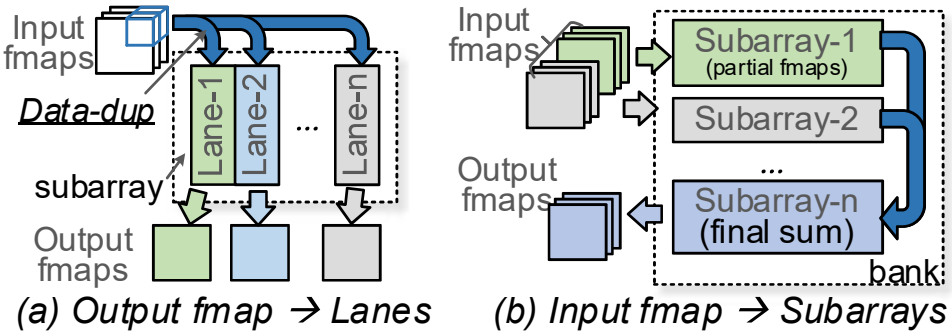

Figure 6.14: The basic resource allocation scheme for lanes and subarrays.

We design the resource allocation scheme as shown in Figure 6.14, following these design philosophies. First, all the input data from feature maps are duplicated to all the lanes in a subarray (Figure 6.14(a)). These lanes work in parallel after weights are preloaded. The output of each lane is a dependent output feature. The purpose for this mapping scheme is to make all of the sum reductions happen in a single lane, so that inefficient *inter-lane SHF* and *lane-FWD* are eliminated. Second, each subarray takes part of the input feature maps. These subarrays work in parallel on the partial results, and an extra subarray sums up the final result (Figure 6.14(b)). Third, for bank and group level parallelism, we take use of the application's parallelism in pixel and batch, i.e., mapping different regions of the feature map to different banks, and different batches to different groups.

Notation	Description
Ru_{kernel}	Convolution data reuse buffering size, none/kernel/lines.
Ln_{ifmap}	Number of input feature maps per lane.
Sr_{act}	Number of active subarrays per Bank.
Bn_{size}	The partition of the feature maps by pixel for computing.

Table 6.2: The design knobs for resource allocation optimization.

With the allocation scheme, we formulate the allocation optimization problem. The design parameters to solve are shown in Table 6.2. Ru_{kernel} denotes the data reuse scheme for convolution computation in a lane. It can buffer (1) nothing, or (2) the kernel while fetching a line in the kernel every time, or (3) lines while fetching only a pixel every time. The formulation of the optimization is described as follows,

$$\textit{To solve:} \quad \{Ru_{\text{kernel}}, Ln_{\text{ifmap}}, Sr_{\text{act}}, Bn_{\text{size}}\} \tag{6.1}$$

$$\textit{Objective:} \quad \max . \, Performance/Watt \tag{6.2}$$

$$\textit{s.t.:} \quad Cap(Ru_{\text{kernel}}, Ln_{\text{ifmap}}) \leq Cap_{\max} \tag{6.3}$$

$$Ln_{\text{ifmap}} \leq \max . \, \text{ifmap}_{number} \tag{6.4}$$

$$Sr_{\text{act}} \leq \max . \, Sr_{\text{act}} \tag{6.5}$$

$$Bn_{\text{size}} \leq \max . \, \text{ifmap}_{\text{size}}. \tag{6.6}$$

In this case study, we maximize performance per watt as the optimization objective (others metric can also apply). We need to solve the design parameters listed in Table 6.2. For the constraints, Equation (6.3) shows that the buffered data should not exceed the memory capacity. Equation (6.4) shows that the number of input feature maps per lane is limited by the total feature map number. Equation (6.5) describes the limitation of the active subarray's quantity (in Section 6.2.4). Equation (6.6) describes the limitation of the feature map size. To solve this

problem, we scan all the design space with the in-house simulator (described in Section 6.4.1).

6.3.3 Mapping Other Applications to *DRISA*

DPU is not limited to only CNN applications. *DRISA* is not limited to CNN inference accelerations. Data quantization in recursive NN (RNN) with 2-bit weights [223] is also an ideal case to run on *DRISA*. Instead of inference, training is also feasible by quantizing gradient data with DoReFa-Net [209].

Furthermore, *DRISA* is not limited to deep learning applications. *DRISA* is designed as a SIMD architecture and can be treated as a vector processor, so a large range of applications can be mapped to *DRISA*. Programs benefit from *DRISA* the most if they have enough data parallelism, if they are both compute and memory intensive, and if they can be mostly computed by integer operations. We are currently working on mapping emerging bioinformatic applications (meta-genome data analysis [191]) to *DRISA*.

6.4 Experiments

In this section, we first describe the experiment setup. Then, overall performance, energy, and area evaluations are presented. The evaluation for the CNN acceleration case study is also presented with comparisons to the state-of-the-art solutions.

6.4.1 Experiment Setups for *DRISA*

We evaluate and compare four *DRISA* designs. The configurations are shown as follows.

3T1C: 8-bit lane, 256 rows and 512 columns per mat, 4 mats (256 rows by 2048 columns, or 256 lanes) per subarray, 16 subarrays per bank, 64 banks per group; in total 4 groups and 2Gb capacity. 22nm DRAM process technology and the 3T1C cell size is $30F^2$ [224].

1T1C-nor/mixed/adder: 1T1C-based solutions with `NOR` logic, or mixed logic gates (including `NAND, NOR, XNOR, INV`), or adder circuit attached to SA (see Section 6.2.1). In total, 8 groups and 4Gb capacity. The cell size is $6F^2$. Rest of the configurations are same as *3T1C*'s.

In order to evaluate the brand new hardware, two in-house simulators are developed. First, a circuit-level simulator is built based on CACTI-3DD [225]. CACTI-3DD is DRAM circuit simulator. It provides DRAM latency, energy, and area parameters, which are validated with fabrication DRAMs. Based on it, our simulator modifies the configuration files to reflect array organization. Then we add extra circuits described in Section 6.2 with APIs provided from CACTI-3DD. The controllers and adders in the *1T1C-adder* solution are synthesized by Design Compiler [226] with an industry library. **The difference between the logic process and DRAM process technologies are capture from parameters in previous research [19].** Second, a behavioral-level simulator is developed from scratch, calculating the latency and energy *DRISA* spends given a certain task like system-C simulation. It also includes a mapping optimization framework for the CNN applications, according to the design space exploration described in Section 6.3.2.

6.4.2 Evaluation for *DRISA* Solutions

Table 6.3 shows the area comparison of the four *DRISA* configurations. First, we observe that even though the memory density of *3T1C* is half as much as others, *3T1C* only takes 17.6% more area than *1T1C-nor*, due to the large cell footprint of the latter. Second, *1T1C-adder* takes the largest area, due to the more complex logic circuit (adders) that are embedded. Third, *DRISA* is almost half as dense as a normal 8Gb DRAM memory. However, *DRISA* is not a cost sensitive memory design. Although it is not as dense as a memory, it very area efficient as an accelerator. Later experiment shows *DRISA* offers the highest performance per area among all kinds of accelerators. Note that though a larger chip with higher performance

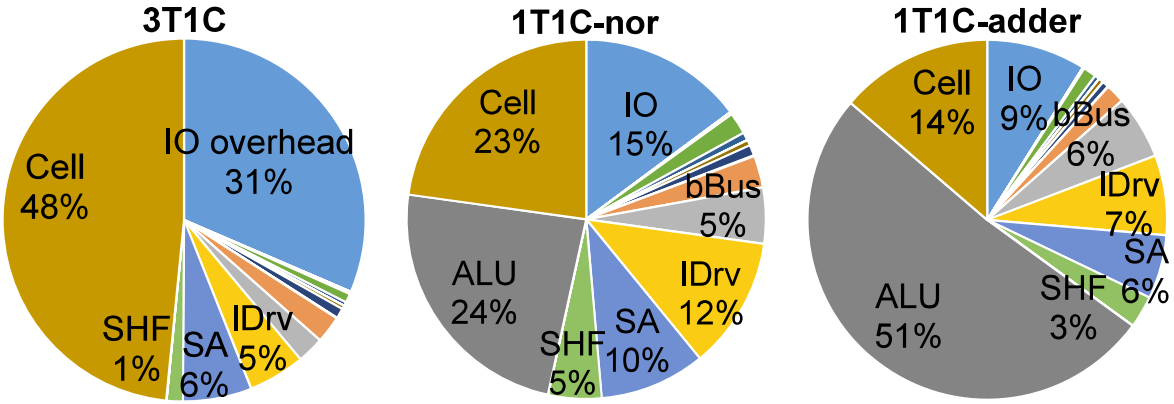

Figure 6.15: The area breakdown of three *DRISA* solutions.

is feasible, the die size impacts the chip cost. *DRISA* thus has a similar die size to commercial DRAM memories in this paper. For a fair comparison, the following results are normalized by area.

Solution	*3T1C*	*1T1C-nor*	*1T1C-mixed*	*1T1C-adder*	*DRAM-8Gb*
Area (mm^2)	64.58	54.90	65.22	90.91	60.44

Table 6.3: The area comparison of *DRISA* solutions, including an 8Gb DRAM memory as a reference.

Figure 6.15 shows the area breakdown. First, we observe that the breakdown of *3T1C* is similar to that of a DRAM memory, where cells and analog IO circuits dominate (79%). The add-on shifters, controllers, buffers, and bus circuits constitute a smaller fraction (less than 5%) in area. Second, the add-on `NOR` and latch circuits in *1T1C-nor* take 24% of the area, almost as much as the memory cell themselves. This is because of the inefficient implementation of logic circuits with DRAM process technologies. Third, the add-on adder circuits in *1T1C-adder* takes 51% of the total area, resulting in 66% more area than *1T1C-nor*. Again, this result supports the observation that **DRAM process technologies are not suitable to build complex logic circuits, with design complexities even for simple adders.**

Figure 6.16 shows the latency (*tStep*) and energy (*eStep*) for a basic computing step (including opening operand rows, computing, closing operand rows, and writing back to result

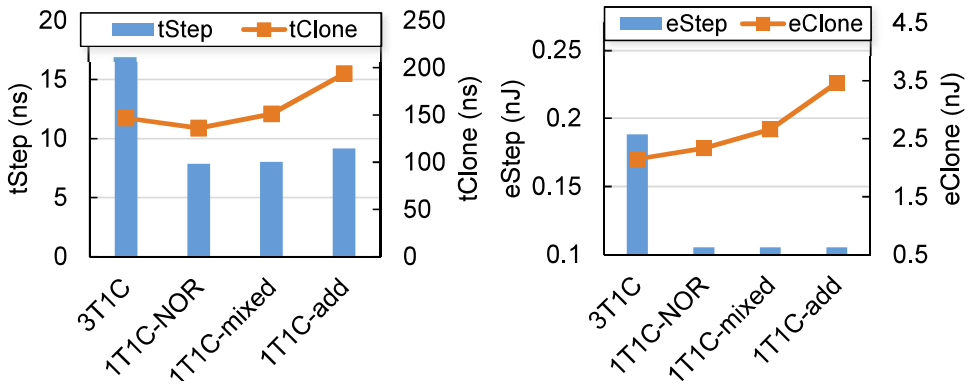

Figure 6.16: The latency and energy comparison among four *DRISA* solutions.

row). It also shows the latency (*tClone*) and energy (*eClone*) to copy one row across subarrays in the same bank. First, we observe that *3T1C* takes 112% more computing latency and 79% more computing energy, due to the longer BLs/WLs and larger cells. Second, the data movement latency/energy are dominated by the latency/energy on wires. Hence, designs with larger areas result in larger latency/energy. Third, a latency breakdown shows that the logics' latency takes less than 1% in all *DRISA* cases, except for *1T1C-adder* which spends 10% latency on the adder.

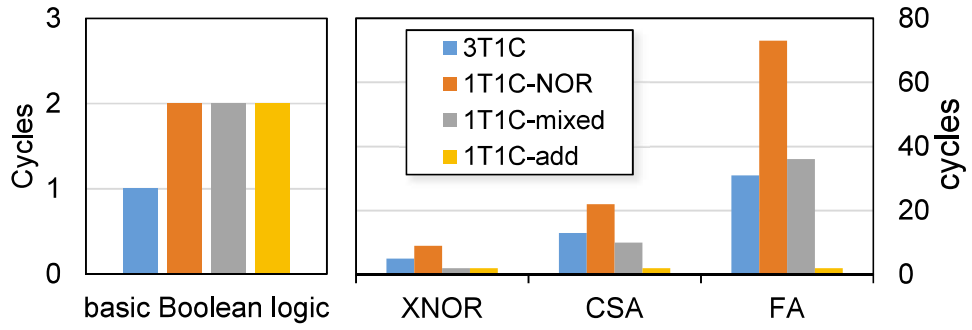

Figure 6.17: Cycles for frequently used operations.

Figure 6.17 shows the cycle count for frequently used operations. It shows that *1T1C-nor* is the slowest since for `AND/OR` the system requires copy-on-operation, which takes 3 more cycles per logic. *1T1C-mixed* appears to be better since every logic operation is based on the add-on

logic gates. However, it still needs 2 cycles for each Boolean logic operation. *1T1C-adder* is clearly the fastest design.

In later sections, we will show effective performance of these four solutions for more comprehensive comparisons before we draw the conclusion in Section 6.5.

6.4.3 The CNN Case Study Results

We compare *DRISA* with the state-of-the-art solutions in the CNN inference acceleration case study.

Baseline setup: We also compare with state-of-art accelerating platforms for the CNN application. They are described as follows.

ASIC: This is a DaDianNao-like [78] ASIC design but optimized for binary weight CNN with 8-bit activation data. There are two versions with either 8x8 tiles (33MB eDRAM) or 16x16 tiles (129MB eDRAM). An advanced on-chip data reuse scheme as in ShiDianNao [227] is adopted. The design is synthesized with Design Compiler [226] and scaled to 22nm. The eDRAM and SRAM are calculated from CACTI [15]. An in-house behaviorial-level simulator is built to evaluate the performance and energy given a certain CNN task.

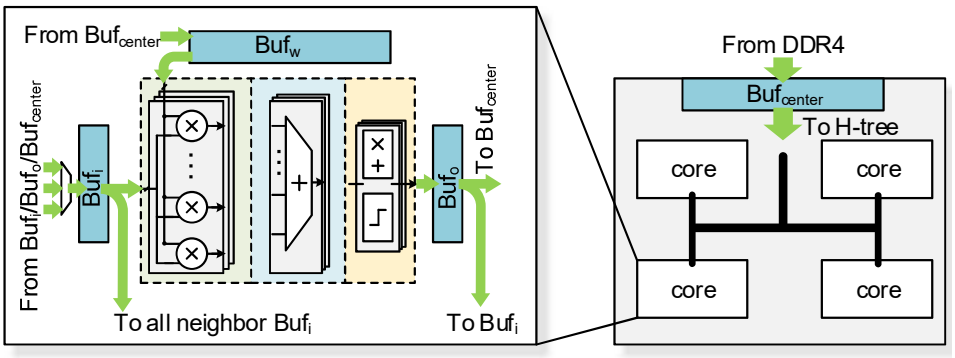

Figure 6.18: The architecture of the ASIC baseline.

GPU: We use two TITAN X (Pascal) [190]. Each GPU has 3584 CUDA cores running at 1.5GHz (11TFLOPs peak performance). *GPU-FP* is achieved by running Torch 7 [203] with

cuDNN [228] using floating point data. We measure the power consumption with NVIDIA's system management interface [229]. The results are conservatively scaled by 50% to exclude the power cost by cooling, voltage regulators, etc. We then aggressively scale the *GPU-FP* result by $\times$4 for the quantized CNN[1], since it claims $\times$4 peak performance running with 8-bit integers instead of floating point data.

In the case study, we consider four CNN applications (including both convolution layers and fully connected layers): 8-layer AlexNet [216], 16-layer VGG-16, 19-layer VGG-19 [219], and 152-layer ResNet-152 [217]. Note that as another advantage, *DRISA* does not have refresh overhead. Even in the most complex CNN case, one iteration of the task is done within 8ms, which means every row have already been read and restored at least once within 64ms.

Performance evaluation: Figure 6.19 shows the peak performance (w/ and w/o normalization by area) for all the solutions (We assume *DRISA* has the same power budget as GPUs). It shows that the best *DRISA* is still 54% slower than *GPU-INT*. Note that *DRISA*'s area is $\sim$14% of GPUs, larger sized *DRISA* with more active subarrays provides higher performance. Therefore, area-normalized results (performance per area) turns to be a fairer performance metric. With this metric, *DRISA* (*1T1C-adder*) outperforms ASIC and GPU by 1.9$\times$ and 12.7$\times$, respectively. Note that peak performance is only part of the picture, and the resource utilization shown later is another key factor.

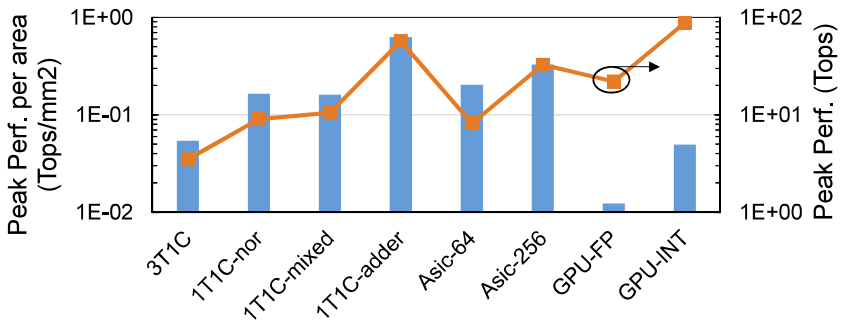

Figure 6.19: The peak performance (w/ and w/o normalization by area) comparison.

[1]No framework supports fixed point CNN on GPU yet, and the real scale ratio should be less than $\times$4 due to the unperfect utilization.

Figure 6.20 shows the on-chip memory capacity and bandwidth (buffer bandwidth for ASIC and register file bandwidth for GPU are counted). First, it shows that *DRISA* has 387× more memory capacity and 54× more bandwidth than GPU and 6.8× and 15× more than ASIC solutions. This is because of the **in-situ computing architecture**. The majority of *DRISA* is DRAM cells for large capacity, and multiple subarrays are activated simultaneously for large bandwidth. Second, *3T1C* and *1T1C-adder* have lower capacity and bandwidth density among *DRISA* solutions, due to the large cell size and large add-on circuit overheads.

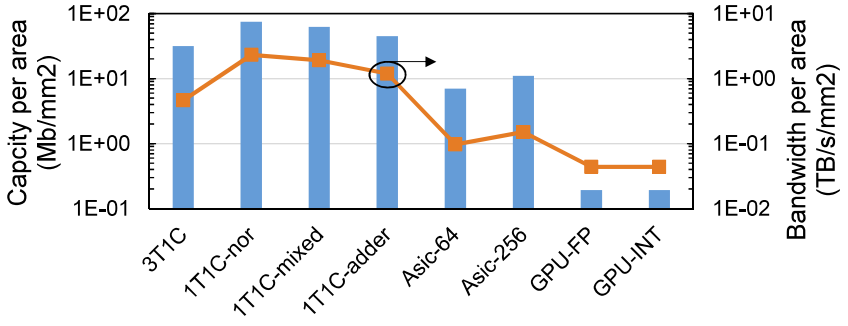

Figure 6.20: The on-chip memory capacity and peak on-chip bandwidth comparison (normalized by area).

Figure 6.21 shows the performance (frames per second) results on CNN applications with a batch size of 1/8/64, which are normalized with area. It shows that *DRISA* is 8.7× and 7.7× faster than the ASIC and GPU solutions, respectively. **It also shows a flipped result:** *1T1C-adder* with higher peak performance (Figure 6.19) is 12.4% slower than *1T1C-mixed*. This is because the computing and data movement costs are not balanced in *1T1C-adder*. Even though the computing is fast, the data movement turns out to be the bottleneck. For the same reason, an ASIC with more tiles is not necessarily better.

Figure 6.22 shows the fraction of time when either on-chip or off-chip data movement blocks computing (data from GPU is not achievable). **It explains why *DRISA* performs better:** First, we observe *DRISA* (except for *1T1C-adder*) only spends ∼10% time on memory access while others spend more than 90% time waiting for the loading data either from off-

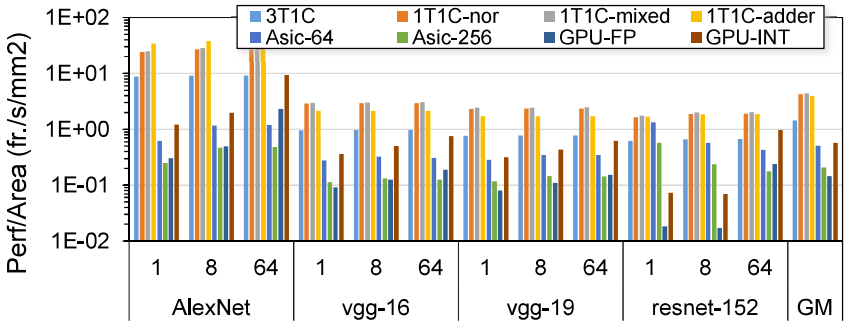

Figure 6.21: The performance comparison (normalized by area).

chip memory or on-chip caches[2]. The low memory bottleneck ratio is then transferred as a high resource utilization in Figure 6.23, which benefits from the in-situ computing architecture. Second, although both *1T1C-adder* and ASICs have more than 90% memory bottleneck ratio, *1T1C-adder* is still 7.8× faster than ASICs (Figure 6.21). This is because *DRISA*'s superiority also stems from its massive parallelism, not only its merging of computing and memory resources.

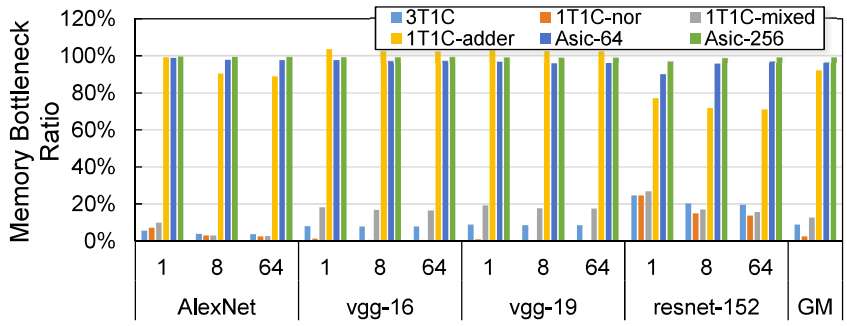

Figure 6.22: The memory bottleneck ratio (when computing has to wait for data).

Figure 6.23 shows the resource utilization (in regard to the peak performance), which strengthens the conclusions drawn from Figure 6.22. *DRISA* (except for *1T1C-adder*) has an average of 45% utilization. The utilization is lower than 50% because it spends at least half of the resources on the data movement, while others are lower than 20%.

[2]The large percentage is not surprising since the computing and memory access are pipelined, and the computing latency is usually hidden by the data movement.

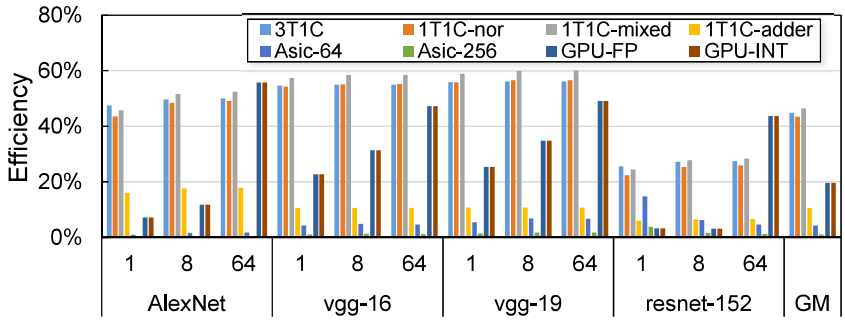

Figure 6.23: The resource utilization efficiency.

Energy Evaluation: Figure 6.24 shows the area-normalized energy efficiency (frames per Joule, higher is better) comparison. First, we observe that GPUs are still the most energy-hungry solutions[3]. Second, *DRISA* is even 1.4× better than ASICs, thanks to the efficient in-situ computing architecture. In addition, DRAM process technologies have less leakage compared with the logic process, especially when considering memory cell's leakage(DRAM retention time is 64ms while eDRAM is $\sim$100μs [20]). Third, *3T1C* is 1.94× better than *1T1C-adder*, because the logic implemented by DRAM process technologies hurts the energy efficiency.

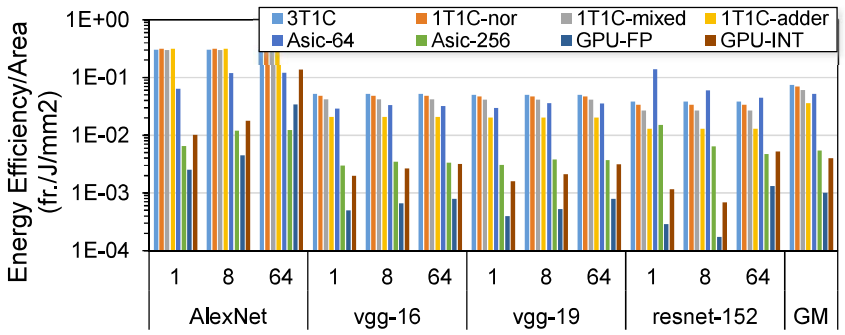

Figure 6.24: The energy efficiency comparison (normalized by area).

Figure 6.25 shows the percentage of energy spent on memory, which explains **why *DRISA* has better energy efficiency**. First, we observe that *DRISA* (except for *1T1C-adder*) spends

[3] We conservatively take 50% of total GPU board power as that actually spent on the GPU chip and GDDR memories.

45% energy on memory, 1.15× smaller than others, thanks to the in-situ computing architecture. Second, *1T1C-adder* spends 92% energy on memory, due to the inefficient DRAM-implemented logics and longer wires induced by the large area overhead.

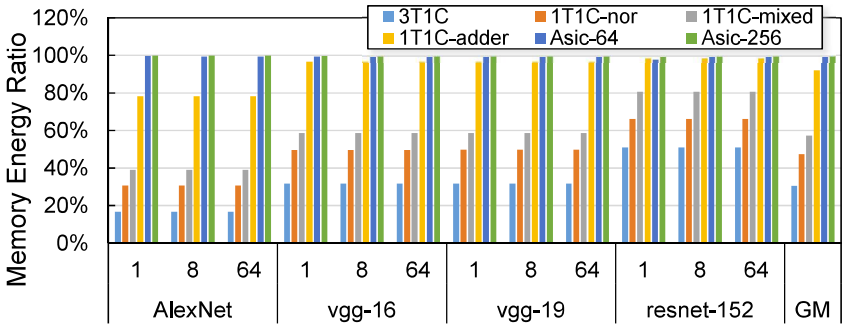

Figure 6.25: The data movement energy ratio.

Figure 6.26 shows the power consumption. It shows that even though *DRISA*-based solutions activate multiple rows simultaneously, the power consumption is still within the power budget and 54% lower than GPUs. This is due to the power budget-aware active subarray number controlling, as described in Section 6.2.4. We also evaluated the power density with the Hotspot tool [230]. The core temperature is well under DRAM's 85°*C* constraint, and existing cooling solutions are sufficient [231]. In addition, when integrating *DRISA* with PCIe like the case of GPUs, the power delivery is not a problem. When integrating with DIMM, *DRISA* needs to shut down parts of the activate subarrays in order to stay within DIMM's power budget.

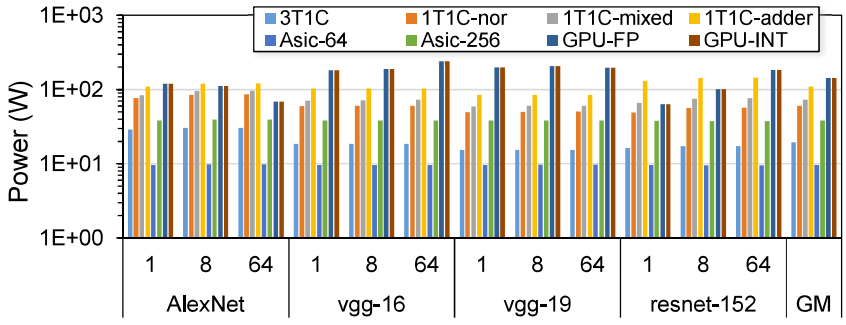

Figure 6.26: The power comparison.

Cost Analysis: Besides the performance and energy benefits, *DRISA* can also potentially offer

lower cost. Not only does *DRISA* have high area efficiency, but also three more reasons contribute to lower cost. First, a DRAM process has only 3 to 4 metal layers [232], while GPUs or ASICs with logic processes usually have more than 10 layers. Second, *DRISA* has fewer pins since it does not require connections to large external memories. This could result in a reduction in packaging cost. Combining these two factors, an industry cost analysis tool [233] shows that *DRISA* can be $\sim$6$\times$ more cost efficient (normalized by area) than GPUs. Third, it has fewer requirements for extra memory chips (like GDDR5), since *DRISA* itself is a memory.

6.5 Discussion: Which DRISA is Better

1T1C-adder is the least effective design. Though *1T1C-adder* has the best (3.84$\times$ better) peak performance, its effective performance is 11% lower than others because its resource utilization is 77% lower. It is also 40% less energy efficient. In addition, it requires the largest area with 51% overhead to build adders, which becomes more difficult to manufacture. *1T1C-adder*'s problems lie in (1) large energy/area overhead of building logics with DRAM process technologies and (2) unbalanced computing and data movement capability (data movement is costly due to longer wires caused by area overhead). As a conclusion, **building too large logic with DRAM process technologies is not feasible**.

On the opposite end of the spectrum from the *1T1C-adder*, *3T1C* has minimal extra logic circuits built in the DRAM process technologies. However, it is also not the most effective design. Though it has 5.7% better energy efficiency, it suffers from 68% lower effective performance. For the area, it is 49% less dense. *3T1C*'s problem is its large cell size. As a conclusion, it shows that **only relying on memory cells for computing is not feasible, either, due to significant performance loss**, though it brings the best energy efficiency.

1T1C-nor/mixed stand somewhere between *1T1C-adder* and *3T1C* and prove to be **the best designs**. They add a few logics in DRAM process technologies but not in excess. *1T1C-mixed*

is 4.7% faster than *1T1C-nor* due to its flexibility to build logics, while *1T1C-nor*'s energy efficiency is 16% better due to more memory cell based computing.

6.6 Conclusion

To address the "memory wall" challenge, we propose a DRAM-based PIM design with simple Boolean logic operations to enable in-situ computing inside DRAM. To overcome the challenges induced by building accelerators with DRAM process technologies, we use simple Boolean logic operations to compute complex functions by running serially. The Boolean logic operations are provided either by the BLs themselves or by extra circuits added to the SAs. We compare four different *DRISA* designs and conclude that *1T1C-nor/mixed* are the best choices. We then present a case study where we evaluate CNN applications on *DRISA*. With the benefit of in-situ computing, *DRISA* shows 8.8× speedup and 1.2× better energy efficiency when compared with ASICs, and 7.7× speedup and 15× better energy efficiency than GPUs.

Chapter 7

SCOPE: A Stochastic Computing Engine for DRAM-based In-situ Accelerator

The DRAM-based in-situ accelerator [234] is proposed to address the challenges faced by both the memory-rich processors and the compute-capable memory architecture. *This in-situ accelerator is different from conventional PIM architectures.* It is a standalone accelerator instead of part of system memory, so it is highly optimized for speed instead of low cost. DRAM, with its smaller memory cells ($6F^2$), allows for large on-chip memory capacity. Furthermore, it tightly combines the compute and memory by attaching a simple processing unit to each memory bitline (BL), and leverages the large number of BLs inside memory for massive parallelism. However, due to the inefficiency of building processing logic circuits with the DRAM technology, it only supports simple bitwise operations (`NOR`) and bitwise shifts [197, 234, 235]. In order to support complex computations such as additions, it breaks those instructions into multiple bitwise Boolean logic operations and processes them one by one. This scheme takes hundreds of cycles for multiplication (MUL) instructions, which raises the challenge for high performance.

To address this challenge, we adopt stochastic computing arithmetic for the DRAM-based

in-situ accelerator. Stochastic computing is an approximate computing method that has been studied for decades [236]. Stochastic computing converts a MUL to a simple bitwise AND operation, dramatically reducing the complexity. It reduces the MUL's latency on the DRAM-based in-situ accelerator by 48× since the latter can do very long vector bitwise operations in parallel. However, the benefit comes at the cost of *data explosion*, since typically stochastic computing represents an n-bit fixed point integer by the probability of the appearance of "1" in a 2^n-bit bitstream, demanding larger memory bandwidth. Note that this challenge would offset the performance improvement when stochastic computing is adopted in a conventional Von Neumann architecture, but not for a DRAM-based in-situ accelerator, which tightly couples large memory with compute units.

In this chapter, we propose SCOPE, the StoChastic cOmPuting Engine for DRAM-based in-situ accelerators. SCOPE tailors and adapts the stochastic computing arithmetic to such in-situ accelerator architecture. It converts the complex MULs into simple AND operations, thus dramatically reducing latency. In return, the in-situ architecture offers large on-chip memory with wide internal bandwidth to address the problem of long bitstreams in stochastic computing, which makes a perfect match between the arithmetic and the architecture. To further overcome stochastic computing challenges (i.e., exponentially long bitstreams, low throughput, and unpredictable numerical precision loss), we propose a novel Hierarchical and Hybrid Deterministic (H^2D) improved arithmetic, introducing three techniques. First, the hierarchical method separately converts the binary number's MSBs part and LSBs part into two shorter stochastic bitstreams, in order to reduce the total bitstream length. Second, we propose a hybrid representation with both a dense binary representation and a compute-friendly stochastic representation, which further reduces the bitstream length. Third, we develop a deterministic approach for the stochastic number generator, significantly improving the numerical precision and reducing the area overhead. Even more importantly, it makes the error reproducible for debugging purposes. The contributions of this paper are summarized as follows:

- We propose the idea of applying stochastic computing to the DRAM-based in-situ accelerator architecture to synergistically reinforce the strengths and address the weaknesses of both paradigms.
- We demonstrate SCOPE, the holistic architecture design that supports stochastic computing in the in-situ accelerator.
- We propose a novel Hierarchical and Hybrid Deterministic (H^2D) stochastic computing arithmetic, providing better performance, lower area overhead, and better numerical precision.
- We evaluate CNN/RNN inference and CNN training tasks on SCOPE as a case study and compare SCOPE with state-of-the-art solutions.

7.1 Background

This section briefly describes the background of stochastic computing, the DRAM-based in-situ accelerator architecture, and deep learning applications.

Stochastic Computing (SC) trades precision and representation density for simpler logic design and lower power. It has been proposed as an alternative low power computing method [237], and is widely applied to image/signal processing, control systems, or general purpose computing [236, 238]. Stochastic computing uses stochastic bitstreams to represent and compute. The probability of the appearance of "1" in the bitstream ($\{x_i\}$) represents the value of the binary number (X), as shown in Equation (7.1) and example of in Equation (7.3).

$$X\ (Binary) \rightarrow \{x_i\}\ (Stoch.),\ X = P(x_i = 1), \tag{7.1}$$

$$X \cdot Y = P(x_i = 1) \cdot P(y_i = 1) = P(x_i = 1\ \&\ y_i = 1). \tag{7.2}$$

Stochastic computing simply AND two bitstreams to calculate a multiplication (MUL). The resulting bitstream's probability of the appearance of "1" gives the result of multiplication, as shown in Equation (7.2) and the example in Equation (7.4). To calculate addition (ADD), stochastic computing uses a multiplexer to evenly select bits from two operand bitstreams.

$$X = \frac{3}{6}\,(Binary) \rightarrow \{x_i\} = \{0,1,0,1,1,0\}\,(Stoch.), \tag{7.3}$$

$$Y = \frac{2}{6}\,(Binary) \rightarrow \{y_i\} = \{0,0,1,1,0,0\}\,(Stoch.),$$

$$X \cdot Y = \frac{1}{6}\,(Binary) \rightarrow \{x_i \& y_i\} = \{0,0,0,1,0,0\}\,(Stoch.). \tag{7.4}$$

There are three major components of stochastic computing. They are (1) a stochastic number generator (SNG), which converts binary data to stochastic bitstreams, (2) logic gates, which calculate MULs and ADDs, and (3) a popcount (PC) unit, which converts the result bitstreams back to binary numbers. For the first part, conventionally, a random number generator (RNG) is adopted, and the binary number is used as the threshold to generate bitstream. It takes 2^n cycles[1] to convert an n-bit binary into a 2^n-bit stochastic bitstream. To reduce energy consumption and improve numerical precision, linear-feedback shift register (LFSR) based SNG method was proposed [239]. Parallel SNGs [240] were proposed based on LFSR to further reduce the conversion latency. For the second part, logic gates, existing work [241, 242] has shown that binary addition is preferred over stochastic additions to obtain better accuracy. This work follows this conclusion and only runs MUL operations in the stochastic domain. For the third part, PC counts the number of "1"s in the bitstream. An approximate PC (APC) [243] is proposed to sacrifice result accuracy for reduced energy consumption and latency.

[1]Throughout this paper, unless otherwise stated, we use 2^n-bit bitstream to represent n-bit binary.

7.2 Motivation

In this section, we introduce the motivation and the key ideas of this work. We describe the challenges of DRAM-based in-situ architecture, and show how adopting stochastic computing can help solve those challenges. Furthermore, the problems of stochastic computing arithmetic and features of the in-situ accelerator which can address these problems are highlighted.

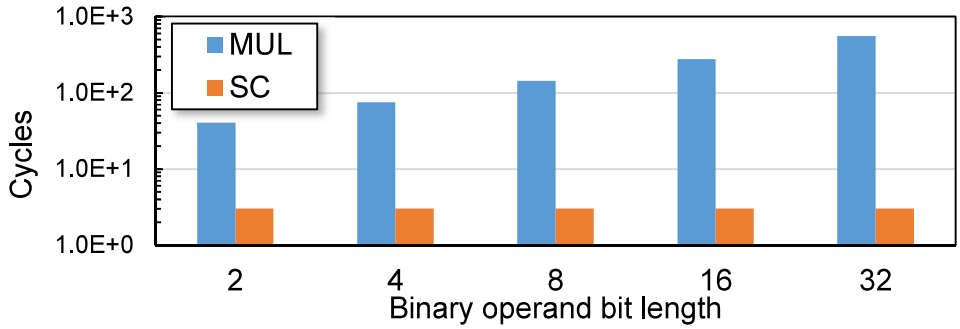

Figure 7.1: Latency (cycles per MUL) comparison of binary and stochastic computing in DRAM-based in-situ accelerator architecture.

Limitation of DRAM-based In-situ Accelerators Building computing units in DRAM processes presents several limitations. Different from a logic process, the DRAM process is optimized for high density and low leakage power, so building logic gates in the DRAM process is not efficient (80% area and 22% performance overhead [19]). Therefore, the in-situ architecture can only build simple bitwise logic gates such as `AND` gates. In order to calculate MULs and ADDs, simple bitwise logic operations are performed repeatedly. The key problem is the slow MUL operation. As shown in Figure 7.1 (blue bars), an 8-bit multiplication takes 143 cycles, and the cycle count increases exponentially with the operand's bit-length (x-axis). This severely degrades the performance, especially the latency. Consequently, applications that can be used with DRAM-based in-situ accelerators are limited to those mainly having ADD operations.

Opportunities and Challenges of Stochastic Computing The multiplication operation becomes a single bitwise AND operation in stochastic computing. The in-situ accelerator computes thousands of AND operations in parallel, taking only 3 cycles. As shown in Figure 7.1 (orange bars), the 8-bit MUL latency is reduced 47.33× by using stochastic computing. The stochastic computing not only boosts the performance, but also enables adoption of the architecture by more applications. For example, previously, DRAM-based in-situ processors could only accelerate binary weight neural networks to avoid MUL operations, whereas when stochastic computing is adopted, they can support more applications like neural network training [244].

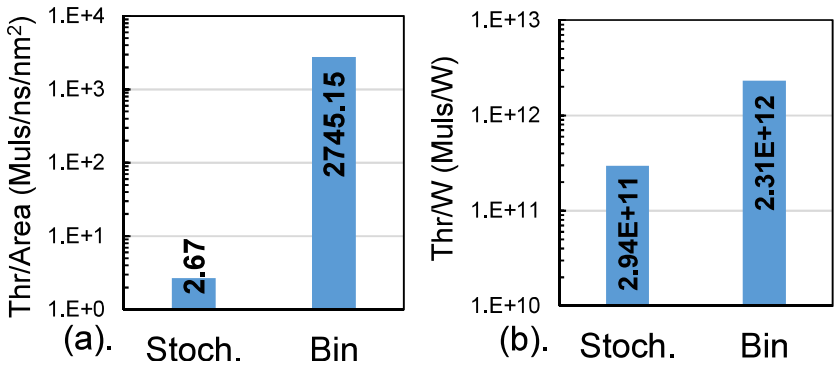

Figure 7.2: Comparison of (a) performance per area and (b) performance per Watt for 10-bit binary multiplier and 1024-bit stochastic computing multiplier.

However, adopting stochastic computing arithmetic is not straightforward. There are many *challenges*. First, the throughput is degraded, although the MUL latency is reduced. Second, the area and power overheads to support stochastic computing are also downsides. Figure 7.2 shows the comparison between the stochastic computing and binary arithmetic[2] in terms of 10-bit multiplier throughput normalized by (a) area and (b) power. Both of them have dedicated registers for data buffering. Stochastic computing computes in parallel (1024 ANDs in one cycle), representing the architecture of the in-situ accelerator. Unfortunately, the stochastic computing case only provides 0.1% throughput/area and 12.7% throughput/power of the conventional binary datapath case.

[2]For the SC-multiplier, we design one-cycle fully parallel circuit synthesized with 45nm FreePDK, integrating parameters from state-of-the-art SNG [240] and PC [243].

There are two reasons behind the low throughput problem. First, data representations in stochastic computing are very inefficient. An n-bit binary is represented with 2^n-length bit-stream in the stochastic domain. The required data storage capacity and bandwidth exponentially increases, demanding more on-chip buffer. As shown in Figure 7.3(a) and (b), 88% of the power in the stochastic computing circuit is spent on data buffering, whereas it is only 13% in the conventional binary case. Second, stochastic computing incurs large area overhead for the supporting circuits, e.g., the SNG and the PC. As shown in Figure 7.3(c), 95% area is dedicated to the SNG [240]. Previous work has also shown that stochastic computing increases energy consumption by 3× compared with binary data path due to the SNG overhead [245].

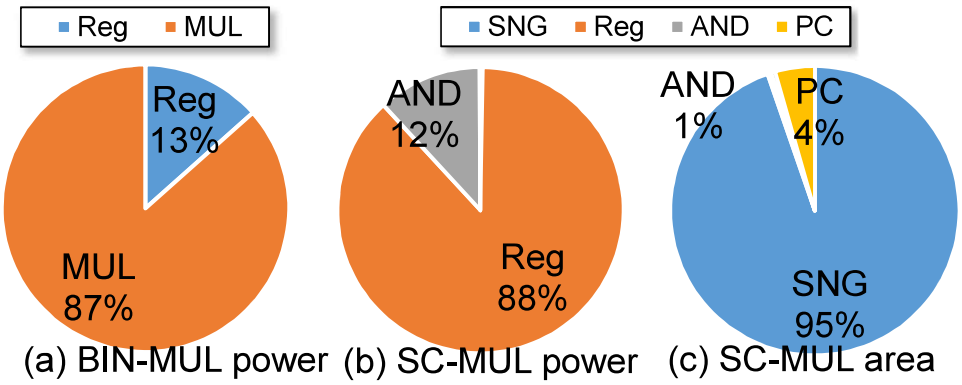

Figure 7.3: (a) Binary MUL Power breakdown, (b) stochastic computing MUL power breakdown, and (c) stochastic computing MUL area breakdown.

Running stochastic computing on *the DRAM-based in-situ accelerator relieves these problems*. Unlike conventional accelerators, which use low density SRAM or eDRAM, the in-situ accelerator has a large amount of DRAM-based on-die memory space, thus it provides high energy efficiency. For example, the DRAM-based in-situ processor [234] spends 33% less energy for buffering 1Mb data than conventional processors using SRAM [15]. Therefore, stochastic computing's requirement for more data buffering is met. In addition, all computing units operate at the BL level, offering ultra high bandwidth so that the stochastic computing can run fully parallel. As a result, *the in-situ accelerator architecture and stochastic computing perfectly match each other. Their advantages compensate each other's disadvantages.*

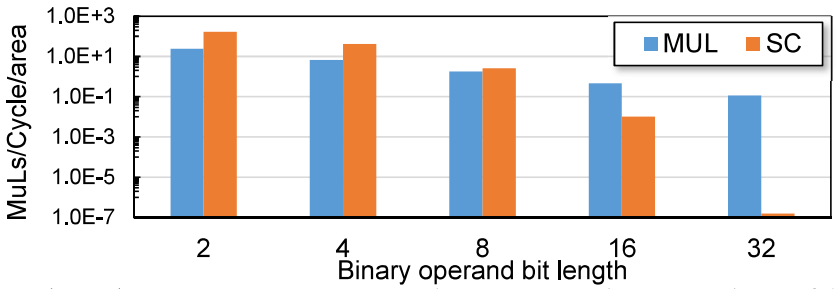

Figure 7.4: Throughput/Area (MULs per cycle per area unit) comparison of binary and stochastic computing MUL in DRAM-based in-situ accelerator architecture.

Towards A Better Stochastic Computing Arithmetic Even when stochastic computing is adopted, certain challenges still remain, although the nature of the in-situ architecture partially solves some issues. Figure 7.4 shows a comparison between binary MUL operations and stochastic computing MUL operations using the in-situ accelerator. Different from the latency in Figure 7.1, this data shows the throughput results normalized by the resource area used. Stochastic computing provides limited throughput benefit only if the operand bit length is less than 8. There are three reasons. **Challenge-1:** The bitstream length needs to be reduced. The long bitstream requires large storage and takes more computing resources. This problem is highlighted in Figure 7.2. These resources could have otherwise been used for more computing parallelism but are wasted to support long bitstreams. **Challenge-2:** The area overhead of the supporting hardware, i.e., the SNG and the PC, as shown in Figure 7.3(c). **Challenge-3:** stochastic computing's numerical precision needs improvement, which is pointed out as a major challenge in the field [236]. In addition, the error incurred by the stochastic computing would be better to be deterministic, so that the system error could be predictable and controllable. The error should also be reproducible to allow for system debugging. In summary, *a better stochastic computing arithmetic with reduced bitstream length, simplified support circuitry, and improved precision is needed to address these problems.*

7.3 SCOPE Architecture

In this section, we describe the SCOPE architecture. We first show how stochastic computing is tailored and adapted to the DRAM-based in-situ accelerator architecture, and then we introduce a series of hardware designs to support stochastic computing.

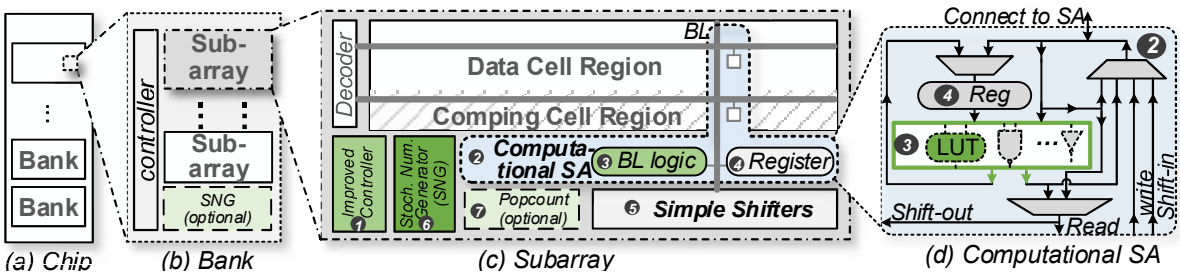

Figure 7.5: SCOPE's architecture overview (Additional hardware to support stochastic computing is colored green).

7.3.1 Customizing Stochastic Computing

The original stochastic computing arithmetic is tailored to better fit with the DRAM-based in-situ accelerator. First, only MULs is performed with stochastic computing, while ADDs still uses the binary data path. This is done to avoid error accumulation, even though this incurs latency and power overheads for converting data back and forth between stochastic bitstreams and binary. Previous work [241, 242] have also shown that stochastic computing-based ADD seriously degrades the application-level computing accuracy. Second, the unipolar stochastic computing [237], which is similar with the unsigned integers, is adopted with an extra bit as the sign-bit. The bipolar stochastic computing arithmetic (like signed integers) is not used to avoid incompatibility with the proposed H^2D method which will be introduced in Section 7.4. Third, fully parallel stochastic computing is used, which is different from most of the conventional serial stochastic computing data path (e.g., taking 1024 cycles to compute with a 1024-bit bitstream), because the DRAM-based in-situ accelerator already provides massively parallel bitwise operation hardware.

7.3.2 Supporting Stochastic Computing

Figure 7.5 shows the overview of the SCOPE architecture. SCOPE includes additional supporting hardware, which are colored green. These include the controller, Stochastic Number Generator (SNG), popcount (PC) unit, and the logic gates on each BL. The whole stochastic computing workflow using SCOPE is described as follows. Step-1, after the instructions are read and consecutively decoded at the bank and the subarray level, SNG is used to convert the binary numbers into stochastic bitstreams, which are then stored along the memory row. Step-2, arithmetic operations are applied on these bitstreams using the bitwise operations provided by each BL. Step-3, the result bitstreams are converted back to binary integers using the PC. In the rest of this subsection, we describe SCOPE's major components following the order of this working flow, and explain how they coordinate and support stochastic computing.

Binary-to-Stochastic Conversion As shown in Figure 7.5-❻, SNG converts binary numbers into stochastic bitstreams. Conventionally, an energy-efficient parallel LFSR-based SNG [240] can be adopted, generating 32-bit bitstream per cycle. However, after our stochastic computing arithmetic improvement is introduced later in Section 7.4, an even simpler and faster SNG solution can be used in SCOPE. The improved arithmetic allows us to use a lookup table (LUT)-based SNG, which generates the whole bitstream in just one cycle.

We introduce two architecture-level optimizations for the conversion, in addition to the SNG hardware itself. First, we choose to convert binary data to stochastic bitstreams before storing them in the subarray. An alternative approach is storing binary data and not converting them until being issued for logic operations. The former approach stores stochastic bitstreams and occupies 2.5× more subarray capacity, but it saves 50176× conversion tasks since the data is heavily reused[3]. Since SCOPE offers large on-chip memory, the proposed approach is preferred to obtain better performance and energy efficiency. Second, we design one SNG for each

[3]The result is for the weight of a convolution layer in CNN (VGG16's first layer) after H^2D optimizations.

subarray instead of each bank, so that binary data is not converted to stochastic bitstreams until the inter-subarray data transfer is done. This design choice reduces data movement overhead by transferring dense binary data instead of the large stochastic bitstreams. Otherwise, moving 1KB data takes 2.5× more latency and energy.

Arithmetic Operation Arithmetic operations are carried out with BL logic operations in computational SAs (Figure 7.5(c)-❷). Figure 7.5(d) shows the components of the computational SA, which mainly include a set of logic gates (e.g., `AND` in ❸) and a register (❹). Three multiplexers (MUXs) are used to select the inputs of the register, the inputs of the logic gates, the restore data source, and the shift data source.

MUL is simplified as an `AND` operation in the stochastic domain. SCOPE needs three cycles to compute `AND`, as shown in Figure 7.6(a). In Cycle-1, row X is read and latched in the register, and this row is then restored (closed). In Cycle-2, row Y is read, and $X\&Y$ is calculated and latched, after which row Y is restored (closed) . In Cycle-3, the result row is activated, and is restored by the result value from the register. Note that we intentionally protect the input operands since they are usually reused in the future. Since DRAM is read destructive, extra cycles is needed to restore original values (X and Y) to the operand rows. ADD in SCOPE is still calculated with binary data path. Figure 7.6(b) shows the working flow to compute a carry-save addition (CSA). This step takes four cycles. In Cycle-1, operand X is loaded. In Cycle-2 and 3, Y and *Cin* are read, intermediate results are calculated accordingly, and *Sum* is restored to the memory. In Cycle-4, carry-out *Co* is calculated and restored back to memory. Note that since operands in sum-reduction are rarely reused in the future, the steps that save operands are skipped, in order to improve performance.

Compared with the baseline architecture [234], we have two optimizations from the hardware design perspective. First, an optimized set of logic gates (Figure 7.5-❸) was selected so that *two* results of any operation can be obtained. For example, $X\&Y$ and $X\wedge Y$ can be computed

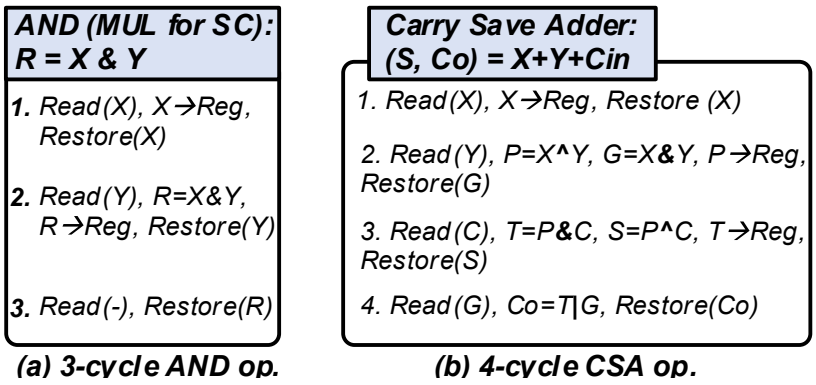

Figure 7.6: (a) Steps to perform MUL. (b) Steps for carry save addition.

simultaneously. This optimized two-output design improves addition operations by 1.75× with 19% area overhead, compared with single-output logic sets. Second, an LUT data path to support the improved H^2D stochastic computing arithmetic is added (described in Section 7.4).

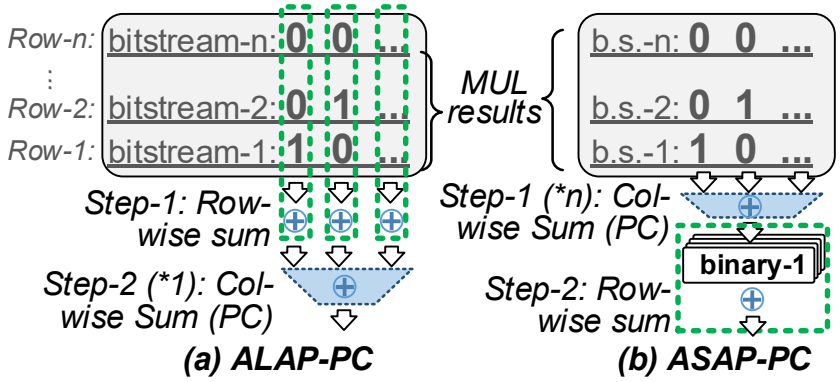

Figure 7.7: Calculating the sum of partial MUL results in MAC operations (a) PC as Late as possible, and (b) PC as Soon as possible.

Stochastic-to-Binary Conversion After calculating MULs in the stochastic domain, we convert the data back to binary data before any further computations, in order to avoid error accumulation. We choose not to adopt the prevailing approximate popcount (APC) proposed in previous work [243] (Figure 7.5-❼). Although the APC reduces the popcount latency, it significantly decreases the numerical precision (see Figure 7.13). Moreover, APC requires de-

signing extra hardware in SCOPE, which takes 11.71% area overhead per subarray. Therefore, we leverage the already existing column-wise addition operations in SCOPE for the conversion, instead of paying extra hardware overhead and suffering precision loss if APC is used.

In order to improve the performance of stochastic-to-binary conversion, we propose a method called ALAP-PC, which calculates the PC as late as possible. This is an effective optimization for vectored multiply-and-accumulate (MAC) operations, which are widely used in many applications such as those that require matrix multiplication. In such situations, there are two types of sum tasks. One is the sum of the bits within the stochastic bitstream to convert it back to binary data, i.e., the popcount. It requires addition across columns, since a bitstream is stored along in one memory row. The other type is the sum of all the MUL results. It requires addition across rows, since the results are stored in different rows. The key idea of the ALAP-PC is to perform the row-wise sum-reduction first, and the column-wise sum-reduction (PC) as late as possible. Figure 7.7 shows (a) the ALAP-PC and (b) the opposite, ASAP-PC, in which column-wise sum-reduction is done first and then row-wise sum-reduction is executed.

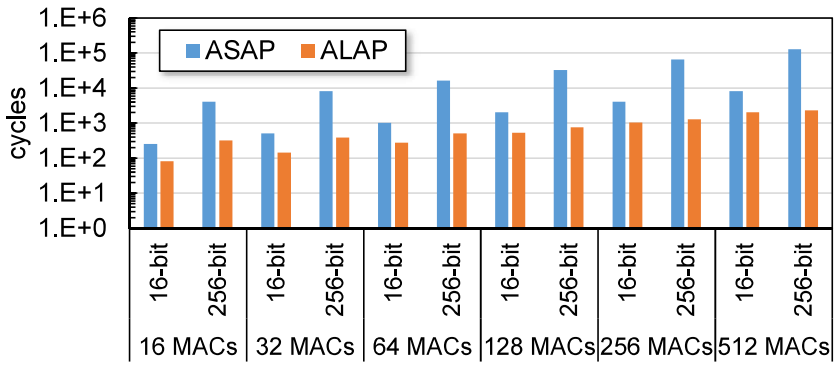

Figure 7.8: ASAP-PC vs. ALAP-PC.

Figure 7.8 shows the performance comparison between these two methods. The x-axis represents the number of MAC operations, with either 16-bit or 256-bit stochastic bitstream. The data shows that the ALAP-PC achieves up to 56× improvement over ASAP-PC. This is because row-wise sum-reduction can use CSA, which has no carry prorogation and can be

calculated efficiently by lock-step bitwise operations within 4 cycles. On the other hand, the column-wise sum-reduction needs full adders (FAs), which takes 22 cycles for 8-bit operands. The ALAP-PC uses less of the inefficient column-wise sum-reduction operation and hence saves latency.

7.4 H^{2}D Arithmetic

The conventional stochastic computing arithmetic suffers from the exponential bitstream length and the numerical precision loss, as discussed in Section 7.2. In this section, we describe our proposed improved stochastic computing arithmetic composed of three key techniques.

7.4.1 Hierarchical Stochastic Bitstreams

In order to reduce the length of stochastic bitstreams, the hierarchial method divides the binary data into two parts and separately converts these parts into stochastic bitstreams. As shown in Figure 7.9, a binary number is divided into the most significant bits (MSBs) part and the least significant bits (LSBs) part. Taking an 8-bit binary number as an example, the MSBs-part is its higher 4 bits and the LSBs-part is its lower 4 bits. Instead of converting the whole n-bit binary data into (2^n-1)-bit stochastic bitstream, the proposed hierarchial method separately converts the $\frac{n}{2}$-bit MSB/LSB parts into two ($2^{\frac{n}{2}}$-1)-bit stochastic bitstreams (referred to as partial bitstreams). Consequently, the stochastic bitstream length is reduced by $2^{\frac{n}{2}-1}$. The MUL operation in the context of the hierarchical method is not a bitwise `AND` anymore, as shown in the lower right corner of Figure 7.9. It contains three partial bitstream MULs, one shift, and two addition operations. Note that it is unnecessary to consider the MUL of two LSB-parts because in stochastic computing, the result is truncated from $2n$-bit to n-bit for an n-bit MUL.

The benefit of the hierarchical method is the reduction of the bitstream length (8$\times$ for 8-bit

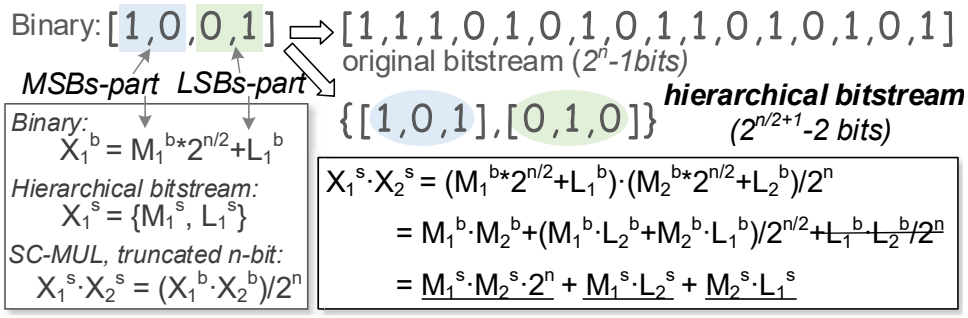

Figure 7.9: Hierarchical stochastic representation.

integer), which then translates to throughput improvement. One drawback of this method is that it makes MUL more complex. For the 8-bit integer example, 6× more cycles are required. However, after considering the MUL overhead, it still brings 5.40× throughput improvement. Another downside is that it degrades the numerical precision because it introduces more error to the MSB-parts. However, we show that this disadvantage can be compensated by other methods, and we show it obtains 60% accuracy improvement when all these techniques are combined in Section 7.7.3.

7.4.2 Hybrid Binary-Stochastic Bitstreams

Here, we discuss the proposed hybrid binary-stochastic data representation and arithmetic. As shown in Figure 7.10, the original stochastic bitstream is divided into groups with three bits. Then, binary representation is used inside each group, while a set of these groups makes up the hybrid binary-stochastic bitstream. To perform MUL operations on the hybrid bitstreams, 2-bit binary MUL is executed within each group, and then a series of the result groups forms as the result hybrid bitstream. The 2-bit MUL for each group can be implemented with simple bitwise operations, as shown in the lower part of Figure 7.10. Note that we choose 3-bit stochastic subsequence to group as a 2-bit binary instead of a longer subsequence. This is because a longer subsequence requires more-than-2-bit binary MUL operations, which results

in larger latency and area overhead that may cancel out the benefit of a short bitstream.

The benefit of the hybrid representation is a 1.5× reduction of bitstream length since it condenses every 3-bit stochastic subsequence into 2-bit binary, which can lead to higher throughput and energy efficiency. In addition, the hybrid method improves the numerical precision by 33%, since part of the MUL calculation is performed in the accurate binary domain. The only drawback is the 2-bit binary MUL calculation. A custom LUT is designed for this operation as shown in Figure 7.5-❸. It takes one more cycle than the original 3-cycle stochastic MUL. However, hybrid representation still offers 11% MUL throughput improvement even after considering this overhead.

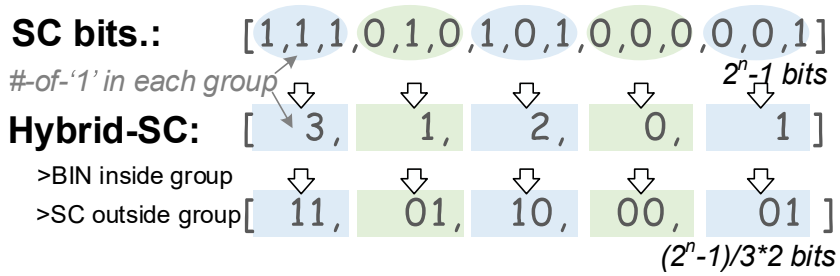

Figure 7.10: Hybrid binary-stochastic representation.

7.4.3 Deterministic SNG

Previous work [246, 247] have shown that the stochastic numbers are not necessarily completely randomized. Consequently, we propose a deterministic SNG. Instead of using conventional random number generator hardware (e.g., RNG or LFSR) for the SNG, we use a pre-programmed lookup table (LUT). As shown in Figure 7.11, one operand is converted into stochastic bitstream in a "0"s-"1"s-"0"s style. The number of "0"s before "1"s are preset as an offset, which is looked-up from the LUT. The number of "1"s is equal to the binary data. The other operand is converted into stochastic bitstream in a periodic style. In other words, "1"s are evenly distributed in the stochastic bitstream. For both of the operands, the number of "1"s are set to the original binary number. Designing the deterministic approach is easier in

SCOPE, because the error propagation is limited by only computing one MUL operation in the stochastic domain and converting stochastic bitstreams back to binary immediately.

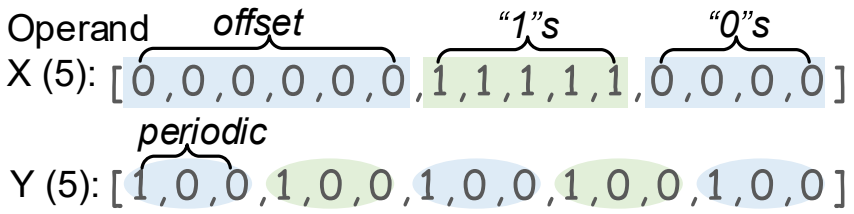

Figure 7.11: A deterministic approach for stochastic number generation.

The deterministic approach has two advantages. First, it simplifies the SNG as a simple LUT, thus it provides 53.2× area reduction. Second, it improves the stochastic computing numerical precision by 23%, as shown in Figure 7.13. Moreover, it makes stochastic computing error predictable and reproducible, which makes it possible to control the error and debug the program.

7.4.4 H^2D: Putting all together

The three techniques (*Hierarchial, Hybrid, and Deterministic*) are orthogonal to each other, so all of them can be applied together. We collectively name these techniques H^2D. For an 8-bit integer, H^2D can reduce the bitstream length by 12× and improve the throughput by 5.2×. The numerical precision is also improved by 60%, as will be shown in Section 7.7.3.

7.5 Discussions

In this subsection, we discuss SCOPE's system integration, programming interface, and limitations.

System integration SCOPE is positioned as an accelerator or co-processor. It is integrated using the PCIe bus, similar to the TPU [13], GPUs [248], or Micron's Automata [249]. SCOPE

has sufficiently large on-chip memory so that no additional memory (DRAM) is required. In addition, SCOPE can be scaled out by connecting multiple chips together, as in the case of multiple GPUs.

Programming interface Although the programming interface is beyond the scope of this paper, here, we briefly discuss the hypothetical framework. We assume a high level framework is built as the user interface, similar to the case of Tenserflow and TPU [13], where the hardware accelerator is transparent to the user. However, to build this framework, lower level APIs that send SCOPE instructions from the host and drivers to fetch data to and from SCOPE are required. We leave the complete software stack as our future work.

Limitations SCOPE has two major limitations. First, although the proposed H^2D improves the numerical precision, it is still limited to applications that can tolerate approximate computing. In addition, the data is limited to fixed-point integer format, since a stochastic bitstream is an alternative representation of an integer, not a floating-point number. Second, SCOPE operations within the same subarrays are executed in lock-step, which limits the programming flexibility and resource utilization. Considering these limitations, the preferred usage is for applications that are data intensive, have significant data parallelism, and can tolerate approximate computing. In this paper, we consider deep learning applications as a case study. However, broader application fields such as image/signal processing and bioinformatics can also potentially be used with SCOPE.

7.6 A Case Study: Deep Learning

In this section, we consider CNN/RNN inference tasks and CNN training tasks as a case study.

7.6.1 Tailoring DNN for Stochastic Computing

Deep learning applications are well-known for being error tolerant. Data quantization [209–215,223] and compression [208] methods have been well studied. Even aggressively quantized CNN training using 1-bit weight, 2-bit activation, and 4-bit gradient shows tolerable recognition accuracy degradation [209]. Previous work have also evaluated applying stochastic computing to CNN inference [241, 242, 245, 250] and training [244]. In this case study, we run CNN/RNN inference task and CNN training task on SCOPE as follows. We apply the quantization DNN methods, but replace the integer MUL with stochastic computing methods. In the back-propagation process, the numerical error induced by stochastic computing is modeled as part of the quantization noise, which is taken into consideration during the gradient calculation. In Section 7.7.4, we will justify that SCOPE is capable for use in deep learning applications.

7.6.2 Mapping DNN on SCOPE

Mapping DNN applications to SCOPE is non-trivial. The general guidelines are: (1) to fully explore task-level or data-level parallelism so that SCOPE's resources are well-utilized, (2) to carefully exploit data reuse so that communication overhead is minimized, and (3) to minimize operations that runs slowly on SCOPE such as the row-wise sum-reduction operation. Following these guidelines, we propose mapping strategies for both the feedforward and back-propagation processes.

Figure 7.12(a) shows how SCOPE maps feedforward tasks (either convolution or MLP layers). The weights are pre-stored in the subarrays to avoid unnecessary data movement. This scheme saves significant latency and energy especially for MLP layers. The activation data (either feature maps of convolution layers or neuron output for MLP layers) is broadcasted along a subarray. Then, each element of the result vector from that subarray becomes an output feature map or a neuron. Finally, SCOPE's subarray-level parallelism is leveraged to process

different parts of the input feature map in parallel, and SCOPE's independent banks are also explored for batch-level parallelism.

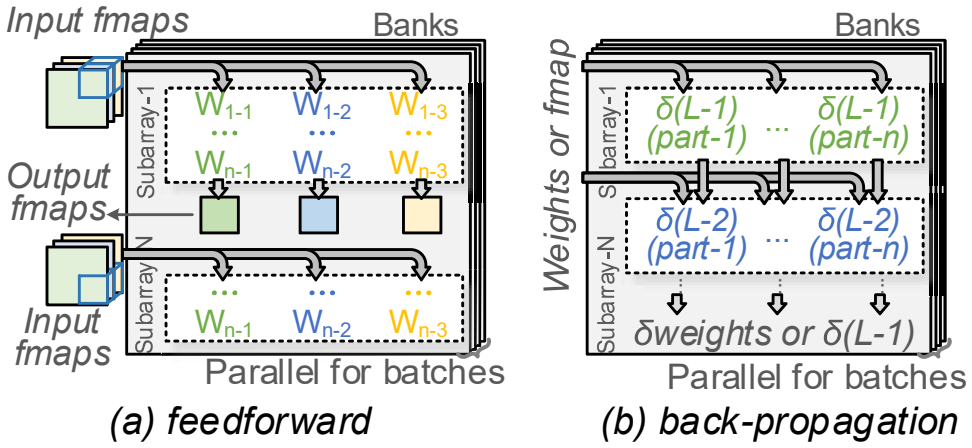

Figure 7.12: Mapping both feedforward and back-propagation of DNN to SCOPE.

Figure 7.12(b) shows the mapping scheme for back-propagation. Different from the feed-forward case, the gradients are stored in the computing subarrays, since both of the two matrix multiplication tasks in back-propagation use the gradient data. Then, the weight are input to the subarray when calculating the gradient of the previous layer, or the activation data are fed in when calculating the weight gradient. Finally, the result of multiple subarrays are merged to get the final result. Different batches are mapped to independent banks for higher throughput. The weight update calculation, if demanding floating-point operations, is offloaded to the host or other hardware.

7.7 Experiments

In this section, after the description of the experimental setup, we demonstrate the evaluation of SCOPE. Then we evaluate the numerical precision improvement of the proposed H^2D method. Finally, we show the performance and energy evaluation when running deep learning applications on SCOPE.

7.7.1 Experimental Setup

The configuration is described as follows. SCOPE is evaluated under 22nm DRAM technology, and has 8Gb DRAM, which is split into 1024 banks, and these banks are further grouped into 64 groups. For each bank, there are 16 subarrays, and each subarray has 256 rows and 2048 columns, an SNG, and a 16-bit adder. We compare SCOPE with baselines described in Table 7.1 in the following experiments.

DRISA [234]	The DRAM-based in-situ acc. w/o. stoch. comp.
SCOPE-vanilla	The SCOPE w/o. H^2D using original stoch. comp.
SCOPE-hyb **SCOPE-hier**	SCOPE w/o. the whole H^2D, both w/. the deterministic SNG but only w/. either *hybrid* or the *hierarchial* .
SCOPE-H^2D	SCOPE with the whole H^2D design **(our proposal)**
GPU	Pascal TITAN X [248] with 40.4 TOPS peak INT8 performance and 12GB GDDR5 device memory.

Table 7.1: Baseline descriptions.

To evaluate SCOPE's circuit, we layout the logic gates for each computational SA (Figure 7.5-❸). We use 45nm FreePDK [251], because this logic process has the similar pitch size with the 22nm DRAM's peripheral region [252]. We then integrate the post-layout compute unit parameters to a heavily-modified CACTI-3DD [225], in order to model the SCOPE architecture. We also develop behavioral level simulators to evaluate SCOPE's effective performance and energy consumption running given applications, according to the mapping techniques described in Section 7.6. The GPU results are measured from dual Pascal TITAN-X with FP32 precision. We conservatively scale these results by 4× as *GPU-INT8* for a fair comparison. We also conservatively exclude one third of the GPU board power consumption for cooling purpose. We include the 12GB device memory as part of GPU's area, since SCOPE's equivalent device memory is on-die.

7.7.2 Performance and Area Evaluation

Table 7.2 shows the comparison between baselines in terms of the MUL latency, the fused MUL-ADD peak throughput, chip area, and performance per unit area. We highlight three key observations from these results. First, simply adopting stochastic computing to the DRAM-based in-situ accelerator decreases the MUL latency by 47.6×. However, this degrades the throughput by 1.21× because of the long bitstream (see Section 7.2). Second, hierarchical and hybrid methods both increase the throughput by reducing the bitstream length at a reasonable cost of longer latency and/or extra area overhead. These methods increase the performance per unit area of SCOPE-vanilla by 4.4× and 1.1×, respectively. Third, putting all the arithmetic optimizations together, *the* H^{2}D *shows* 4.2× *and* 6.16× *better performance and performance per unit area compared with the SCOPE-vanilla*.

	DRISA[a]	**SCOPE**				**GPU**
		vanilla	**hier**	**hybrid**	***H^{2}D***	**INT8**
MUL latency[b]	143	3	17	4	21	0.5
Peak TOPs[c]	1.65	1.36	5.98	1.55	7.08	40.4
Area (mm^2)	258.2	259.42	258.2	273.38		1631[d]
Peak GOPs/Area	6.39	5.24	23.16	5.67	**25.90**	24.76

[a] We re-evaluate DRISA [234] with 8Gb setup and integrating more accurate post-layout logic gates parameters; [b]Cycle; [c]INT8-fused MUL-ADD operation, including SNG/PC; [d]Normalized to 22nm, including 12GB DRAM.

Table 7.2: Peak Performance Comparison.

Table 7.3 shows the SCOPE-*H^{2}D*'s area, latency, and energy, compared with commodity DRAMs with same capacity. As expected, SCOPE's area is larger than the commodity DRAM. However, note that this work is targeted as a high performance accelerator rather than system main memory. Enabling wider computing capability and higher performance is a priority for this type of accelerator. One reason for the area overhead is the extra hardware, e.g., computational SA in Figure 7.5-❷. Another reason is the array reorganization. SCOPE adopts a large number of smaller yet less dense arrays with short BL and WL [253], which contributes to both

a much shorter latency and increased parallelism. Consequently, SCOPE is 5.03× faster with 19.36× less energy consumption.

Area (mm^2)		Latency (ns)		Energy (nJ)	
SCOPE	**DRAM**	**SCOPE**	**DRAM**	**SCOPE**	**DRAM**
273.38	61.73	8.02	40.35	0.11	2.13

Table 7.3: The area, latency, and energy of SCOPE and commodity DRAM.

7.7.3 Precision Evaluation

As one of the approaches to approximate computing, the numerical precision in SCOPE is very important. In this subsection, we evaluate SCOPE's precision, and show how it can be improved by the proposed H^2D method. Then, we show how SCOPE impacts the accuracy of neural networks in our case study described in Section 7.6.

Figure 7.13 shows the error root mean square (RMS) of all possible INT8 operand combinations. In this figure, the x-axis is different arithmetic configurations, in which the baseline is the vanilla stochastic computing (SCOPE-vanilla). Applying hybrid and deterministic methods alone (the second and fourth bar) reduces the error by 33% and 23%, respectively. By replacing the approximate APC used in previous work with the accurate PC (the fifth bar), an extra 25% error reduction is achieved. Even though the hierarchial method increases the error by 275% (the third bar), putting them together with the accurate PC, *H^2D improves the overall precision by 60% over baseline*.

In the experiment, we also evaluate the impact of numerical precision on deep learning applications. The stochastic computing-based CNN evaluation based on previous DNN quantization methods [209, 223]. stochastic computing is simulated by adding noise to integer MULs according to the error RMS data from Figure 7.13. The numerical error induced by stochastic computing is then modeled as part of the quantization noise in the back-propagation process. In the CNN inference experiments, we conservatively quantize both weight and acti-

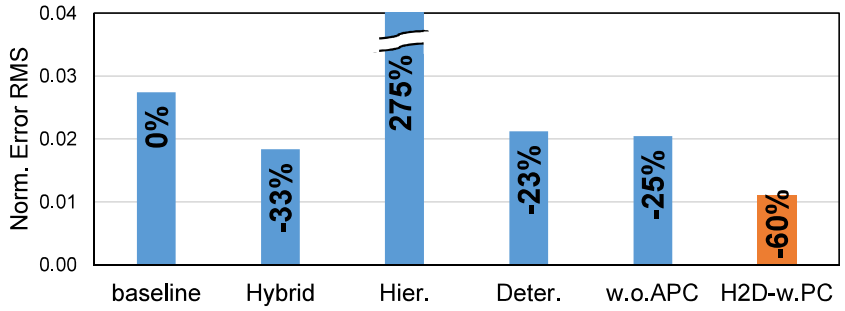

Figure 7.13: The error RMS of 8-bit MUL for the proposed H^2D method (normalized to range $[0, 1]$).

vation data to 8-bit integers while using floating-point gradient for the offline training. Then, we apply stochastic computing for all integer MULs in the feedforward process. In the CNN training experiments, in addition to the 8-bit weights and activation data, we also quantize gradients as 8-bit integers. Then, we apply stochastic computing for all integer MULs in both the feedforward and back-prorogation process. For this case study, we have evaluated LeNet [254] on MNIST [255]. It is difficult to examine a larger neural network, because random number generation required to simulate stochastic computing on current GPUs is inefficient, resulting in extremely long training periods. However, we observe that SCOPE offers even smaller accuracy loss compared with quantized neural network solutions, so we believe that the performance of SCOPE on larger scale neural networks is also similar with other neural network quantization work [209, 223].

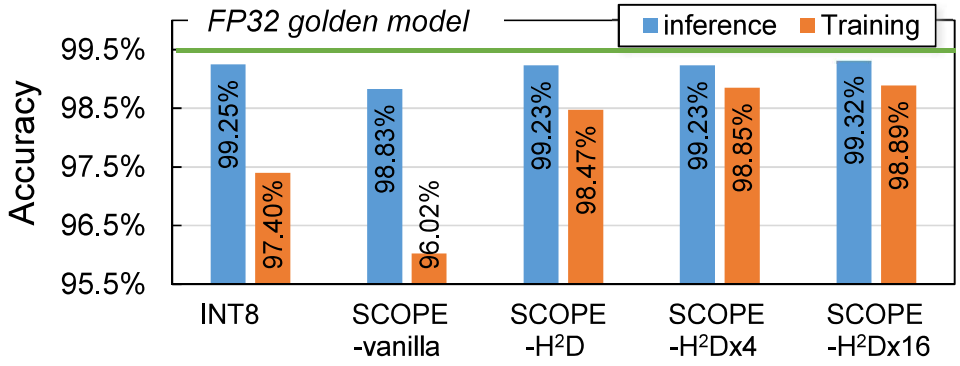

Figure 7.14: The stochastic computing based CNN inference and training recognition accuracy.

Figure 7.14 shows both the inference and training recognition accuracy for integer quantization, SCOPE-vanilla, and the proposed SCOPE-H^2D method. For inference (blue bars), first, we observe that H^2D-based CNN only has 0.27% accuracy degradation than the golden model (using FP32, 99.5% accuracy), and only 0.02% degradation compared with the integer quantization method. Second, the proposed H^2D method is better than SCOPE-vanilla by 0.4%, which is a even larger gap than that between H^2D and the golden model. For training (orange bars), we observe that although H^2D-based training has 1.03% accuracy degradation compared with the golden model, it is 1.07% and 2.45% better than the integer-based solution and the SCOPE-vanilla, respectively. In addition, we observe that having longer stochastic bitstreams, which can help to improve the numerical precision, is not necessary in this situation. The last two bars show that 4× and 16× longer bitstreams only improve the accuracy by 0% and 0.09% in the inference experiment, and 0.38% and 0.42% in the training experiment.

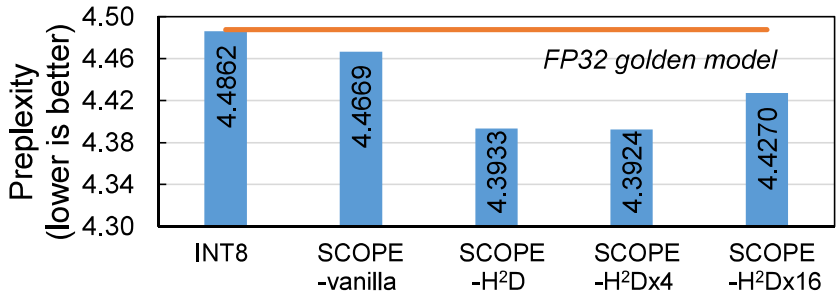

Figure 7.15: The stochastic computing based vanilla RNN inference for character-level language model.

We also evaluate vanilla RNN inference applications in the experiment, which justify that SCOPE can be widely adopted to various deep learning applications. The perplexity is the RNN accuracy metric, in which case lower perplexity is better. We still use 8-bit weights and activation data, and floating-point gradient, and we use stochastic computing for integer MULs in the feedforward processes. We use a vanilla RNN model with 3 layers and 256 neurons per layer running a character-level language model [256]. As shown in Figure 7.15, we observe that although the integer-based quantization method has a similar result as the golden model,

both the SCOPE-vanilla and the proposed SCOPE-H^2D have a better result than the golden model. The SCOPE-vanilla has a 0.02-lower perplexity, and SCOPE-H^2D is even better with a 0.09-lower perplexity. In summary, *we have shown that SCOPE is effective for some of the deep learning applications*.

7.7.4 Evaluating DNN on SCOPE

In this subsection, we evaluate SCOPE with deep learning applications as a case study. We consider four benchmarks, as shown in Table 7.4, in which all the CNN applications run on ImageNet [220] data set and the RNN runs on a character-level language model [256]. The batch size is set to 64 for all the experiments.

vgg	VGG-16 [219], CNN inference
resnet	ResNet-152 [217], CNN inference
rnn	Vanilla RNN, 3-layer×256-neuron, inference
Alex-train	AlexNet [216], CNN training

Table 7.4: Benchmark descriptions.

Figure 7.16 shows the performance per unit area results, which are normalized to the *DRISA* baseline. On average, the proposed SCOPE-H^2D is 2.3× better than *DRISA* and 3.8× better than *GPU-INT*. Although *GPU-INT* provides high performance when its resource utilization is high, the data movement may offset its advantages. In addition, compared with the methods without H^2D or with partial H^2D, SCOPE-H^2D always provides better performance. SCOPE-H^2D is 11.6×, 9.0×, and 1.1× better than the SCOPE-vanilla, SCOPE-hyb, SCOPE-hier, respectively. Moreover, the trend stays the same for each particular benchmark. The superior performance of the SCOPE-H^2D comes from both the bitstream length reduction and the simplified hardware.

Figure 7.17 shows the energy efficiency (performance per Watt) results, which are normalized to the *DRISA* baseline. On average, the proposed SCOPE-H^2D's energy efficiency

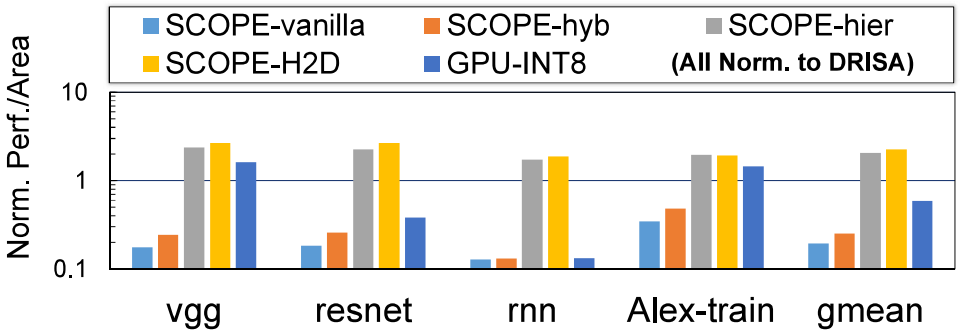

Figure 7.16: Performance per unit area (task/s/mm^2) normalized to *DRISA [234]*.

is 4.4× better than the baseline *DRISA*, and is 1.7× better compared with *GPU-INT*. SCOPE performs better for memory-intensive inference benchmarks, e.g., *ResNet* and *RNN*, since data movement overhead is significantly reduced by storing data within its large on-chip memory. In addition, we also demonstrate the importance of the H^2D method. With the H^2D method, it provides 7.1× better energy efficiency than the SCOPE-vanilla. Compared with the partially adopted H^2D (SCOPE-hyb and SCOPE-hier) baselines, the SCOPE-H^2D is 5.4× and 1.2× better on average, and the same trend applies to every benchmark.

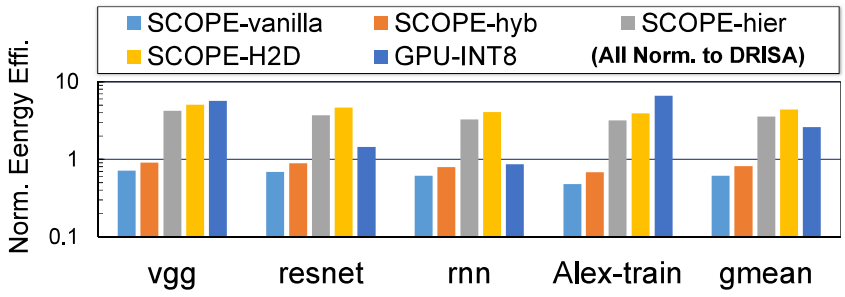

Figure 7.17: Energy efficiency (task/J) normalized to *DRISA [234]*.

Figure 7.18 shows the power consumption (lower is better), which are normalized to the *DRISA* baseline. On average, SCOPE-H^2D has lower power consumption than the baseline *DRISA* by 1.8×, because of stochastic computing's high power efficiency. It also uses lower power when compared with *GPU-INT* by 5.2×, contributed by the reduction of data movement energy. Again, the data also shows that the power efficiency is improved by the proposed H^2D

method. SCOPE-H^2D reduces power by 1.7× and 1.6× compared with the SCOPE-vanilla and SCOPE-hier. Adopting the hybrid method increases the power consumption because of the extra LUT logic operations in the computational SA (Figure 7.5-❸).

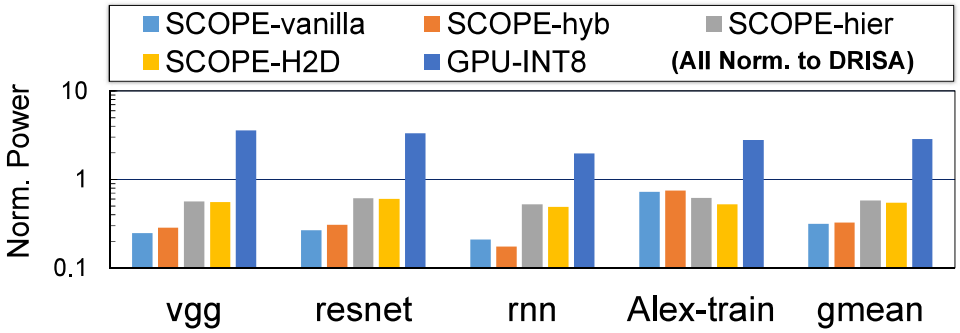

Figure 7.18: Power consumption (Watt) normalized to *DRISA [234]*.

7.8 Conclusion

In this chapter, we design SCOPE, a holistic architecture which adopts stochastic computing for the DRAM-based in-situ accelerator architecture, two paradigms that synergistically complement each other. Stochastic computing simplifies the complex MUL operations so that MULs can be efficiently calculated with the simple Boolean logic gates available in the DRAM-based in-situ accelerator. In return, the in-situ architecture offers large on-chip memory with wide internal bandwidth to address the problem of long bitstreams in stochastic computing. To further improve the performance and energy efficiency, we propose the H^2D arithmetic optimization methods. We evaluated the proposed architecture with a case study of deep learning applications. The experimental results show that SCOPE is 2.3× better than the DRAM-based in-situ accelerator baseline, and 3.8× better than GPU, in terms of area-normalized performance. The proposed H^2D arithmetic optimization improves the performance by 11.6× and boosts the energy efficiency by 5.4×. In addition, H^2D improves stochastic computing's numerical precision by 60% on average, which translates into accuracy

improvement when adopted to deep learning applications.

Chapter 8

Summary

Data movement between the processing units and the memory in traditional von Neumann architecture is creating the "memory wall" and "power wall" challenge. To bridge the gap between the computing and the memory, two approaches, the *compute-capable memory* and the *memory-rich processor* have been studied in this thesis.

We first explore the compute-capable memory architecture. As a highlight, we leverage emerging NVM for this study, which leads to two benefits. First, as the next generation system memory technology, NVM is turning mature and offering better scalability and lower cost, compared with the prevailing 3D stacking DRAM solutions. Second, we leverage NVM's special feature (e.g., resistive cell structure) and utilize the memory cell themselves for computing, so that the performance is maximized with reasonable overhead. Specifically, there are two architecture designs from Chapter 3 and 4.

First, PINATUBO, a processing in non-volatile memory architecture for bulk bitwise operations, is proposed. While most of the recent work focused on PIM in DRAM memory with 3D die-stacking technology, we propose to leverage the unique features of emerging non-volatile memory (NVM), such as resistance-based storage and current sensing, to enable efficient PIM design in NVM. Instead of integrating complex logic inside the cost-sensitive mem-

ory, PINATUBO redesigns the read circuitry so that it can compute the bitwise logic of two or more memory rows very efficiently, and support one-step multi-row operations.

Then, PRIME, a novel processing-in-memory architecture for neural network computation in reram-based main memory, is proposed. In this work, we propose a novel PIM architecture, called PRIME, to accelerate NN applications in ReRAM based main memory. In PRIME, a portion of ReRAM crossbar arrays can be configured as accelerators for NN applications or as normal memory for a larger memory space. We provide microarchitecture and circuit designs to enable the morphable functions with an insignificant area overhead. We also design a software/hardware interface for software developers to implement various NNs on PRIME. Benefiting from both the PIM architecture and the efficiency of using ReRAM for NN computation, PRIME distinguishes itself from prior work on NN acceleration, with significant performance improvement and energy saving.

We then explore the memory-rich accelerator architecture. The uniqueness of our study is that we provide ultra large on-chip memory capacity by utilizing either NVM or DRAM, which offer much higher density than conventional SRAM or eDRAM. Specifically, there are three architecture design from Chapter 5, 6, and 7.

First, we proposed we propose a circuit-level model and develop a simulation tool, NVSIM-CAM, which helps researchers to make early design decisions, and to evaluate device/circuit innovations. The tool is validated by HSPICE simulations and data from fabricated chips. We also present a case study to illustrate how NVSIM-CAM benefits the nvTCAM design. In the case study, we propose a novel 3D vertical ReRAM based TCAM cell, the 3DvTCAM. We project the advantages/disadvantages and explore the design space for the proposed cell with NVSIM-CAM. We also show the potential architecture innovations that facilitated by the 3DvTCAM.

Then, *DRISA*, a DRAM-based reconfigurable in-situ accelerator architecture, to provide *both* powerful computing capability and large memory capacity/bandwidth. *DRISA* is primar-

ily composed of DRAM memory arrays, in which every memory bitline can perform bitwise Boolean logic operations (such as NOR). *DRISA* can be reconfigured to compute various functions with the combination of the functionally complete Boolean logic operations and the proposed hierarchical internal data movement designs. We further optimize *DRISA* to achieve high performance by simultaneously activating multiple rows and subarrays to provide massive parallelism, unblocking the internal data movement bottlenecks, and optimizing activation latency and energy. We explore four design options and present a comprehensive case study to demonstrate significant acceleration of convolutional neural networks.

Finally, We propose SCOPE, the stochastic computing engine with a novel in-memory computing architecture and improved arithmetic. In stochastic computing, binary numbers are converted into stochastic bitstreams, which turns integer multiplications into simple bitwise AND operations, but at the expense of larger memory capacity/bandwidth demands. Stochastic computing is a perfect match for the DRAM-based in-situ accelerators because it addresses the in-situ accelerator's low performance problem by simplifying the operations, while leveraging the in-situ accelerator's advantage of large memory capacity/bandwidth. To further boost the performance and compensate for the numerical precision loss, we propose a novel Hierarchical and Hybrid Deterministic (H^2D) stochastic computing arithmetic. Finally, we consider quantized deep neural network inference and training applications as a case study.

We hope the work in this thesis would be useful and inspirational for both accelerator and memory system design, especially in the context of emerging applications such as deep learning.

Bibliography

[1] Top500, "The list of top500 super computers." https://www.top500.org/statistics/, 2017.

[2] Nvidia, *Nvidia tesla v100 gpu architecture*, *Technology Report* (2017).

[3] K. Rupp, *Cpu, gpu and mic hardware characteristics over time*, 2013. https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/.

[4] R. Williams, T. Sze, D. Huang, S. Pannala, and C. Fang, *Server memory road map*, *Memory Forum* (2012).

[5] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, *Tiered-latency DRAM: A Low Latency and Low Cost DRAM Architecture*, in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, (Washington, DC, USA), pp. 615–626, IEEE Computer Society, 2013.

[6] E. Computing, *The opportunities and challenges of exascale computing*, .

[7] O. Villa, D. R. Johnson, M. O'Connor, E. Bolotin, D. Nellans, J. Luitjens, N. Sakharnykh, P. Wang, P. Micikevicius, A. Scudiero, S. W. Keckler, and W. J. Dally, *Scaling the Power Wall: A Path to Exascale*, in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 830–841, IEEE Press, 2014.

[8] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, *A scalable processing-in-memory accelerator for parallel graph processing*, in *International Symposium on Computer Architecture (ISCA)*, (New York, New York, USA), pp. 105–117, ACM Press, 2015.

[9] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim in *International Symposium on Microarchitecture (MICRO)]*, pp. 1–13, ACM, 2016.

[10] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, *Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory*, in *International Symposium on Computer Architecture (ISCA)*, pp. 380–392, IEEE, jun, 2016.

[11] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. 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*, pp. 469–480, Dec, 2009.

[12] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, *DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning*, in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 269–284, 2014.

[13] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, *In-Datacenter Performance Analysis of a Tensor Processing Unit*, .

[14] C. Gonzalez, E. Fluhr, D. Dreps, D. Hogenmiller, R. Rao, J. Paredes, M. Floyd, M. Sperling, R. Kruse, V. Ramadurai, R. Nett, S. Islam, J. Pille, and D. Plass, *POWER9: A processor family optimized for cognitive computing with 25Gb/s accelerator links and 16Gb/s PCIe Gen4*, in *IEEE International Solid-State Circuits Conference*, pp. 50–51, IEEE, feb, 2017.

[15] N. P. Muralimanohar, Naveen and Balasubramonian, Rajeev and Jouppi, *CACTI 6.0: A tool to model large caches*, *HP Lab.* (2009) 22–31.

[16] G. Fredeman, D. W. Plass, A. Mathews, J. Viraraghavan, K. Reyer, T. J. Knips, T. Miller, E. L. Gerhard, D. Kannambadi, C. Paone, D. Lee, D. J. Rainey, M. Sperling, M. Whalen, S. Burns, R. R. Tummuru, H. Ho, A. Cestero, N. Arnold, B. A. Khan, T. Kirihata, and S. S. Iyer, *A 14 nm 1.1 Mb Embedded DRAM Macro With 1 ns Access*, *IEEE Journal of Solid-State Circuits* **51** (jan, 2016) 230–239.

[17] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, *Demystifying gpu microarchitecture through microbenchmarking*, in *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, pp. 235–246, IEEE, 2010.

[18] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, *Fine-grained DRAM: Energy-efficient DRAM for Extreme Bandwidth Systems*, in *International Symposium on Microarchitecture (MICRO)*, pp. 41–54, ACM, 2017.

[19] Y.-B. Kim and T. W. Chen, *Assessing merged DRAM/Logic technology*, *Integration, the VLSI Journal* **27** (jul, 1999) 179–194.

[20] Mu-Tien Chang, P. Rosenfeld, Shih-Lien Lu, and B. Jacob, *Technology comparison for large last-level caches (L3Cs): Low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM*, in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 143–154, IEEE, feb, 2013.

[21] H.-S. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, *Phase change memory*, *Proceedings of the IEEE* **98** (Dec, 2010) 2201–2227.

[22] Y. Huai, *Spin-transfer torque mram (stt-mram): Challenges and prospects*, *AAPPS Bulletin* **18** (2008), no. 6 33–40.

[23] H.-S. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. Chen, and M.-J. Tsai, *Metal oxide rram*, *Proceedings of the IEEE* **100** (June, 2012) 1951–1970.

[24] G. De Sandre *et. al.*, *A 90nm 4mb embedded phase-change memory with 1.2v 12ns read access time and 1mb/s write throughput*, in *ISSCC*, pp. 268–269, Feb, 2010.

[25] C. Kim, K. Kwon, C. Park, S. Jang, and J. Choi, *A covalent-bonded cross-coupled current-mode sense amplifier for stt-mram with 1t1mtj common source-line structure array*, in *Solid- State Circuits Conference - (ISSCC), 2015 IEEE International*, pp. 1–3, Feb, 2015.

[26] M.-F. Chang, J.-J. Wu, T.-F. Chien, Y.-C. Liu, T.-C. Yang, W.-C. Shen, Y.-C. King, C.-J. Lin, K.-F. Lin, Y.-D. Chih, S. Natarajan, and J. Chang, *Embedded 1mb reram in 28nm cmos with 0.27-to-1v read using swing-sample-and-couple sense amplifier and self-boost-write-termination scheme*, in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 332–333, Feb, 2014.

[27] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, J. Shin, Y. Rho, C. Lee, M. G. Kang, J. Lee, Y. Kwon, S. Kim, J. Kim, Y.-J. Lee, Q. Wang, S. Cha, S. Ahn, H. Horii, J. Lee, K. Kim, H. Joo, K. Lee, Y.-T. Lee, J. Yoo, and G. Jeong, *A 20nm 1.8v 8gb pram with 40mb/s program bandwidth*, in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pp. 46–48, Feb, 2012.

[28] T. yi Liu, T. H. Yan, R. Scheuerlein, Y. Chen, J. Lee, G. Balakrishnan, G. Yee, H. Zhang, A. Yap, J. Ouyang, T. Sasaki, S. Addepalli, A. Al-Shamma, C.-Y. Chen,

M. Gupta, G. Hilton, S. Joshi, A. Kathuria, V. Lai, D. Masiwal, M. Matsumoto, A. Nigam, A. Pai, J. Pakhale, C. H. Siau, X. Wu, R. Yin, L. Peng, J. Y. Kang, S. Huynh, H. Wang, N. Nagel, Y. Tanaka, M. Higashitani, T. Minvielle, C. Gorla, T. Tsukamoto, T. Yamaguchi, M. Okajima, T. Okamura, S. Takase, T. Hara, H. Inoue, L. Fasoli, M. Mofidi, R. Shrivastava, and K. Quader, *A 130.7mm2 2-layer 32gb reram memory device in 24nm technology*, in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pp. 210–211, Feb, 2013.

[29] *Intel and micron have new class of non-volatile memory that is 1000 times faster and 10 times denser than nand flash memory*, .

[30] T. yi Liu, T. H. Yan, R. Scheuerlein, Y. Chen, J. Lee, G. Balakrishnan, G. Yee, H. Zhang, A. Yap, J. Ouyang, T. Sasaki, A. Al-Shamma, C. Chen, M. Gupta, G. Hilton, A. Kathuria, V. Lai, M. Matsumoto, A. Nigam, A. Pai, J. Pakhale, C. H. Siau, X. Wu, Y. Yin, N. Nagel, Y. Tanaka, M. Higashitani, T. Minvielle, C. Gorla, T. Tsukamoto, T. Yamaguchi, M. Okajima, T. Okamura, S. Takase, H. Inoue, and L. Fasoli, *A 130.7-mm 2-layer 32-gb reram memory device in 24-nm technology*, *Solid-State Circuits, IEEE Journal of* **49** (Jan, 2014) 140–153.

[31] F. Bedeschi, R. Fackenthal, C. Resta, E. Donze, M. Jagasivamani, E. Buda F. Pellizzer, D. Chow, A. Cabrini, G. Calvi, R. Faravelli, A. Fantini, G. Torelli, D. Mills, R. Gastaldi, and G. Casagrande, *A multi-level-cell bipolar-selected phase-change memory*, in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pp. 428–625, Feb, 2008.

[32] X. Lou, Z. Gao, D. V. Dimitrov, and M. X. Tang, *Demonstration of multilevel cell spin transfer switching in mgo magnetic tunnel junctions*, *Applied Physics Letters* **93** (2008), no. 24 242502.

[33] S.-S. Sheu, M.-F. Chang, K.-F. Lin, C.-W. Wu, Y.-S. Chen, P.-F. Chiu, C.-C. Kuo, Y.-S. Yang, P.-C. Chiang, W.-P. Lin, C.-H. Lin, H.-Y. Lee, P.-Y. Gu, S.-M. Wang, F. Chen, K.-L. Su, C.-H. Lien, K.-H. Cheng, H.-T. Wu, T.-K. Ku, M.-J. Kao, and M.-J. Tsai, *A 4mb embedded slc resistive-ram macro with 7.2ns read-write random-access time and 160ns mlc-access capability*, in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pp. 200–202, Feb, 2011.

[34] M.-F. Chang, S.-S. Sheu, K.-F. Lin, C.-W. Wu, C.-C. Kuo, P.-F. Chiu, Y.-S. Yang, Y.-S. Chen, H.-Y. Lee, C.-H. Lien, F. Chen, K.-L. Su, T.-K. Ku, M.-J. Kao, and M.-J. Tsai, *A high-speed 7.2-ns read-write random access 4-mb embedded resistive ram (reram) macro using process-variation-tolerant current-mode read schemes*, *Solid-State Circuits, IEEE Journal of* **48** (March, 2013) 878–891.

[35] J. Meza, J. Li, and O. Mutlu, *A case for small row buffers in non-volatile main memories*, in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pp. 484–485, Sept, 2012.

[36] S. Cho and H. Lee, *Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance*, in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pp. 347–357, Dec, 2009.

[37] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, *A durable and energy efficient main memory using phase change memory technology*, in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp. 14–23, ACM, 2009.

[38] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, *Architecting phase change memory as a scalable dram alternative*, in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp. 2–13, ACM, 2009.

[39] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, *Scalable high performance main memory system using phase-change memory technology*, in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp. 24–33, ACM, 2009.

[40] P. Nair, C. Chou, B. Rajendran, and M. Qureshi, *Reducing read latency of phase change memory via early read and turbo read*, in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 309–319, Feb, 2015.

[41] P. Chi, C. Xu, T. Zhang, X. Dong, and Y. Xie, *Using multi-level cell stt-ram for fast and energy-efficient local checkpointing*, in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, pp. 301–308, 2014.

[42] J. Wang, X. Dong, and Y. Xie, *Enabling high-performance lpddrx-compatible mram*, in *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, pp. 339–344, ACM, 2014.

[43] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, *Evaluating stt-ram as an energy-efficient main memory alternative*, in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pp. 256–267, April, 2013.

[44] M. Qureshi, M. Franceschini, and L. Lastras-Montano, *Improving read performance of phase change memories via write cancellation and write pausing*, in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pp. 1–11, Jan, 2010.

[45] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras, *Preset: Improving performance of phase change memories by exploiting asymmetry in write times*, in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, pp. 380–391, IEEE Computer Society, 2012.

[46] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, *A case for intelligent ram*, *IEEE micro* **17** (1997), no. 2 34–44.

[47] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava, *et. al.*, *Mapping irregular applications to diva, a pim-based data-intensive architecture*, in *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, p. 57, ACM, 1999.

[48] M. Oskin, F. T. Chong, and T. Sherwood, *Active pages: A computation model for intelligent memory*, vol. 26. IEEE Computer Society, 1998.

[49] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, *Flexram: Toward an advanced intelligent memory system*, in *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pp. 5–14, IEEE, 2012.

[50] J. Lee and J. H. Ahn and K. Choi, *Buffered compares: Excavating the hidden parallelism inside DRAM architectures with lightweight logic*, in *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1243–1248, 2016.

[51] J. Ahn, S. Yoo, and K. Choi, *AIM: Energy-Efficient Aggregation Inside the Memory Hierarchy*, *ACM Transactions on Architecture and Code Optimization* **13** (oct, 2016) 1–24.

[52] Y. Wang, Y. Han, L. Zhang, H. Li, and X. Li, *Propram: Exploiting the transparent logic resources in non-volatile memory for near data computing*, in *Proceedings of the 52Nd Annual Design Automation Conference*, pp. 47:1–47:6, ACM, 2015.

[53] P. Trancoso, *Moving to Memoryland: In-memory Computation for Existing Applications*, in *International Conference on Computing Frontiers*, pp. 32:1—-32:6, ACM, 2015.

[54] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, *NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules*, in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 283–295, feb, 2015.

[55] M. Gao and C. Kozyrakis, *HRL: Efficient and flexible reconfigurable logic for near-data processing*, in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 126–137, IEEE, mar, 2016.

[56] D. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, *TOP-PIM: Throughput-oriented Programmable Processing in Memory*, in *International Symposium on High-performance Parallel and Distributed Computing*, pp. 85–98, ACM, 2014.

[57] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C. Cher, C. H. A. Costa, J. Doi, C. Evangelinos, B. M. Fleischer, T. W. Fox, D. S. Gallo, L. Grinberg, J. A. Gunnels, A. C. Jacob, P. Jacob, H. M. Jacobson, T. Karkhanis, C. Kim, J. H.

Moreno, J. K. O'Brien, M. Ohmacht, Y. Park, D. A. Prener, B. S. Rosenburg, K. D. Ryu, O. Sallenave, M. J. Serrano, P. D. M. Siegl, K. Sugavanam, and Z. Sura, *Active Memory Cube: A processing-in-memory architecture for exascale systems*, *IBM Journal of Research and Development* **59** (mar, 2015) 17:1–17:14.

[58] S. G. Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand and O. Mutlu, *Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation*, in *International Conference on Computer Design (ICCD)*, 2016.

[59] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, *PIM-enabled Instructions: A Low-overhead, Locality-aware Processing-in-memory Architecture*, in *International Symposium on Computer Architecture (ISCA)*, pp. 336–348, ACM, 2015.

[60] B. Akin, F. Franchetti, and J. C. Hoe, *Data Reorganization in Memory Using 3D-stacked DRAM*, in *International Symposium on Computer Architecture (ISCA)*, pp. 131–143, ACM, 2015.

[61] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, *NDC: Analyzing the Impact of 3D-Stacked Memory+ Logic Devices on MapReduce Workloads*, in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014.

[62] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, *Near-Data Processing: Insights from a MICRO-46 Workshop*, in *Micro, IEEE*, vol. 34, pp. 36–42, IEEE, jul, 2014.

[63] Q. Guo, T.-M. Low, N. Alachiotis, B. Akin, L. Pileggi, J. C. Hoe, and F. Franchetti, *Enabling portable energy efficiency with memory accelerated library*, in *International Symposium on Microarchitecture (MICRO)*, (New York, New York, USA), pp. 750–761, ACM Press, dec, 2015.

[64] M. Gao, G. Ayers, and C. Kozyrakis, *Practical Near-Data Processing for In-memory Analytics Frameworks*, *Parallel Archit. Compil. Tech. (PACT), 2015 IEEE Int. Conf.* (2015) 113–124.

[65] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, *Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities*, in *International Conference on Parallel Architectures and Compilation (PACT)*, (New York, New York, USA), pp. 31–44, ACM Press, 2016.

[66] B. Hong, G. Kim, J. H. Ahn, Y. Kwon, H. Kim, and J. Kim, *Accelerating Linked-list Traversal Through Near-Data Processing*, in *International Conference on Parallel Architectures and Compilation (PACT)*, (New York, New York, USA), pp. 113–124, ACM Press, 2016.

[67] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C.-Y. Cher, C. H. Costa, J. Doi, C. Evangelinos, *et. al.*, *Active memory cube: A processing-in-memory architecture for exascale systems*, *IBM Journal of Research and Development* **59** (2015), no. 2/3 17–1.

[68] Q. Guo, T.-M. Low, N. Alachiotis, B. Akin, L. Pileggi, J. C. Hoe, and F. Franchetti, *Enabling portable energy efficiency with memory accelerated library*, in *Proceedings of the 48th International Symposium on Microarchitecture*, pp. 750–761, ACM, 2015.

[69] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, *Tetris: Scalable and efficient neural network acceleration with 3d memory*, in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 751–764, ACM, 2017.

[70] V. Seshadri, *Simple DRAM and Virtual Memory Abstractions to Enable Highly Efficient Memory Systems*, *CoRR* **abs/1605.06483** (2016).

[71] V. Seshadri, M. A. Kozuch, T. C. Mowry, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, and P. B. Gibbons, *RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization*, in *International Symposium on Microarchitecture (MICRO)*, (New York, New York, USA), pp. 185–197, ACM Press, 2013.

[72] V. Seshadri, K. Hsieh, A. Boroumand, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, *Fast Bulk Bitwise AND and OR in DRAM*, *Computer Architecture Letters* **PP** (2015), no. 99 1.

[73] M. Gao, C. Delimitrou, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and C. Kozyrakis, *DRAF: A Low-Power DRAM-Based Reconfigurable Acceleration Fabric*, in *International Symposium on Computer Architecture (ISCA)*, pp. 506–518, IEEE, jun, 2016.

[74] AMD, "Radeon R9 Series Graphics Cards." `www.amd.com/r9`, 2015.

[75] Nvidia, "Pascal GPU Architecture ." `http://www.nvidia.com/object/gpu-architecture.html`, 2016.

[76] Intel, "Intel Xeon Phi Processor: Your Path to Deeper Insight." `http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-processor-product-brief.html`, 2016.

[77] Xilinx, "Xilinx Virtex UltraScale+ HBM device family ." `https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html`, 2016.

[78] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, *DaDianNao: A Machine-Learning Supercomputer*, in *International Symposium on Microarchitecture (MICRO)*, pp. 609–622, IEEE, dec, 2014.

[79] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, *Deep Image: Scaling up Image Recognition*, *CoRR* **abs/1501.02876** (2015).

[80] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, *From High-Level Deep Neural Models to FPGAs*, in *International Symposium on Microarchitecture (MICRO)*, IEEE, Oct, 2016.

[81] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, *A million spiking-neuron integrated circuit with a scalable communication network and interface*, *Science* **345** (2014), no. 6197 668–673.

[82] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, *A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm*, in *Custom Integrated Circuits Conference (CICC)*, pp. 1–4, IEEE, sep, 2011.

[83] S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla, S. Chandra, N. Basilico, S. Carpin, T. Zimmerman, F. Zee, R. Alvarez-Icaza, J. A. Kusnitz, T. M. Wong, W. P. Risk, E. McQuinn, T. K. Nayak, R. Singh, and D. S. Modha, *Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores*, in *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10, IEEE, aug, 2013.

[84] J.-s. Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. A. Tierno, L. Chang, D. S. Modha, and D. J. Friedman, *A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons*, in *Custom Integrated Circuits Conference (CICC)*, pp. 1–4, IEEE, sep, 2011.

[85] P. Dlugosch, D. Brown, P. Glendenning, M. Leventhal, and H. Noyes, *An efficient and scalable semiconductor architecture for parallel automata processing*, in *Parallel and Distributed Systems, IEEE Transactions on*, p. 99, IEEE, 2014.

[86] A. Akerib and E. Ehrman, *Non-volatile in-memory computing device*, may, 2015. US Patent App. 14/588,419.

[87] A. Akerib, O. AGAM, E. Ehrman, and M. Meyassed, *Using storage cells to perform computation*, dec, 2014. US Patent 8,908,465.

[88] A. Akerib and E. Ehrman, *In-memory computational device*, nov, 2014. US Patent App. 14/555,638.

[89] Y. Yang, J. Mathew, M. Ottavi, S. Pontarelli, and D. K. Pradhan, *2t2m memristor based tcam cell for low power applications*, in *Design Technology of Integrated Systems in Nanoscale Era (DTIS), 2015 10th International Conference on*, pp. 1–6, April, 2015.

[90] M.-F. Chang, C.-C. Lin, A. Lee, C.-C. Kuo, G.-H. Yang, H.-J. Tsai, T.-F. Chen, S.-S. Sheu, P.-L. Tseng, H.-Y. Lee, and T.-K. Ku, *A 3t1r nonvolatile tcam using mlc reram with sub-1ns search time*, in *Solid- State Circuits Conference - (ISSCC), 2015 IEEE International*, pp. 1–3, Feb, 2015.

[91] J. Li, R. Montoye, M. Ishii, and L. Chang, *1 mb 0.41 um2 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing*, *Solid-State Circuits, IEEE Journal of* **49** (April, 2014) 896–907.

[92] S. Matsunaga, S. Miura, H. Honjou, K. Kinoshita, S. Ikeda, T. Endoh, H. Ohno, and T. Hanyu, *A 3.14 um2 4t-2mtj-cell fully parallel tcam based on nonvolatile logic-in-memory architecture*, in *VLSI Circuits (VLSIC), 2012 Symposium on*, pp. 44–45, June, 2012.

[93] S. Matsunaga, A. Katsumata, M. Natsui, T. Endoh, H. Ohno, and T. Hanyu, *Design of a 270ps-access 7-transistor/2-magnetic-tunnel-junction cell circuit for a high-speed-search nonvolatile ternary content-addressable memory*, *Journal of Applied Physics* **111** (2012), no. 7 07E336.

[94] X. Guo, E. Ipek, and T. Soyata, *Resistive computation: Avoiding the power wall with low-leakage, stt-mram based computing*, in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pp. 371–382, ACM, 2010.

[95] T. Hanyu, D. Suzuki, N. Onizawa, S. Matsunaga, M. Natsui, and A. Mochizuki, *Spintronics-based nonvolatile logic-in-memory architecture towards an ultra-low-power and highly reliable vlsi computing paradigm*, in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1006–1011, 2015.

[96] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, *Ac-dimm: associative computing with stt-mram*, in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pp. 189–200, ACM, 2013.

[97] H.-J. Tsai, K.-H. Yang, Y.-C. Peng, C.-C. Lin, Y.-H. Tsao, M.-F. Chang, and T.-F. Chen, *Energy-efficient non-volatile tcam search engine design using priority-decision in memory technology for dpi*, in *Proceedings of the 52Nd Annual Design Automation Conference*, pp. 100:1–100:6, ACM, 2015.

[98] C.-Y. Wen, J. Li, S. Kim, M. Breitwisch, C. Lam, J. Paramesh, and L. Pileggi, *A non-volatile look-up table design using pcm (phase-change memory) cells*, in *VLSI Circuits (VLSIC), 2011 Symposium on*, pp. 302–303, June, 2011.

[99] M.-F. Chang, S.-M. Yang, C.-C. Kuo, T.-C. Yang, C.-J. Yeh, T.-F. Chien, L.-Y. Huang, S.-S. Sheu, P.-L. Tseng, Y.-S. Chen, F. Chen, T.-K. Ku, M.-J. Tsai, and M.-J. Kao, *Set-triggered-parallel-reset memristor logic for high-density heterogeneous-integration friendly normally off applications*, *Circuits and Systems II: Express Briefs, IEEE Transactions on* **62** (Jan, 2015) 80–84.

[100] L. Xie, H. A. Du Nguyen, M. Taouil, S. Hamdioui, and K. Bertels, *Fast boolean logic mapped on memristor crossbar*, .

[101] H. Li *et. al.*, *A learnable parallel processing architecture towards unity of memory and computing*, *Scientific reports* **5** (2015).

[102] B. Chen, F. Cai, W. Ma, P. Sheridan, and U. W. Lu, *Efficient in-memory computing architecture based on crossbar arrays*, in *Electron Devices Meeting, 2015. IEDM'15 Technical Digest. IEEE International*, Dec, 2015.

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

[104] 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*, in *Proceedings of the 43Nd Annual International Symposium on Computer Architecture*, ISCA '16, 2016.

[105] S. Hamdioui, L. Xie, H. A. D. Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike, *et. al.*, *Memristor based computation-in-memory architecture for data-intensive applications*, in *DATE*, pp. 1718–1725, 2015.

[106] K. Wu, *Fastbit: an efficient indexing technology for accelerating data-intensive science*, in *Journal of Physics*, vol. 16, p. 556, 2005.

[107] S. Beamer, K. Asanović, and D. Patterson, *Direction-optimizing breadth-first search*, in *SC*, pp. 1–10, Nov, 2012.

[108] M. Pedemonte, E. Alba, and F. Luna, *Bitwise operations for gpu implementation of genetic algorithms*, in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation (GECCO)*, pp. 439–446, ACM, 2011.

[109] J. Bruce, T. Balch, and M. Veloso, *Fast and inexpensive color image segmentation for interactive robots*, in *Intelligent Robots and Systems (IROS)*, vol. 3, 2000.

[110] M.-F. Chang, S.-J. Shen, C.-C. Liu, C.-W. Wu, Y.-F. Lin, Y.-C. King, C.-J. Lin, H.-J. Liao, Y.-D. Chih, and H. Yamauchi, *An offset-tolerant fast-random-read current-sampling-based sense amplifier for small-cell-current nonvolatile memory*, *Solid-State Circuits, IEEE Journal of* **48** (March, 2013) 864–877.

[111] K. Suzuki and S. Swanson, *The non-volatile memory technology database (nvmdb)*, Tech. Rep. CS2015-1011, Department of Computer Science & Engineering, University of California, San Diego, May, 2015. http://nvmdb.ucsd.edu.

[112] J. Li *et. al.*, *1 mb 0.41 um 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing*, *JSSC* **49** (April, 2014) 896–907.

[113] J. Ahn *et. al.*, *Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture*, in *ISCA*, pp. 336–348, ACM, 2015.

[114] V. Seshadri *et. al.*, *Fast bulk bitwise and and or in dram*, *CAL* **PP** (2015), no. 99 1–1.

[115] K. Chen *et. al.*, *Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory*, in *DATE*, pp. 33–38, 2012.

[116] "Laboratory for web algorithmics." `http://law.di.unimi.it/`.

[117] "The star experiment." `http://www.star.bnl.gov/`.

[118] X. Dong *et. al.*, *Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory*, *TCAD* **31** (July, 2012) 994–1007.

[119] T. E. Carlson *et. al.*, *Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation*, in *SC*, pp. 52:1–52:12, ACM, 2011.

[120] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, *Overcoming the challenges of crossbar resistive memory architectures*, in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pp. 476–488, Feb, 2015.

[121] 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* **PP** (6, 2015) 1–1.

[122] J. Li, C.-I. Wu, S. Lewis, J. Morrish, T.-Y. Wang, R. Jordan, T. Maffitt, M. Breitwisch, A. Schrott, R. Cheek, H.-L. Lung, and C. Lam, *A novel reconfigurable sensing scheme for variable level storage in phase change memory*, in *Memory Workshop (IMW), 2011 3rd IEEE International*, pp. 1–4, IEEE, 2011.

[123] M. Hu, H. Li, Q. Wu, and G. Rose, *Hardware realization of bsb recall function using memristor crossbar arrays*, in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pp. 498–503, June, 2012.

[124] B. Li, Y. Shan, M. Hu, Y. Wang, Y. Chen, and H. Yang, *Memristor-based approximated computation*, in *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pp. 242–247, Sept, 2013.

[125] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, *Training and operation of an integrated neuromorphic network based on metal-oxide memristors*, *Nature* (2014).

[126] Y. Kim, Y. Zhang, and P. Li, *A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing*, *J. Emerg. Technol. Comput. Syst.* **11** (Apr., 2015) 38:1–38:25.

[127] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, *High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm*, *Nanotechnology* **23** (2012), no. 7 075201.

[128] F. A. L. Gao and D. B. Strukov, *A high resolution nonvolatile analog memory ionic devices*, in *4th Annual Non-Volatile Memories Workshop, NVMW 2013*, pp. paper–57, 2013.

[129] M. Hu, J. P. Strachan, E. Merced-Grafals, Z. Li, and R. S. Williams, *Dot-product engine: Programming memristor crossbar arrays for efficient vector-matrix multiplication*, in *ICCAD'15 Workshop on "Towards Efficient Computing in the Dark Silicon Era"*, Nov, 2015.

[130] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, *Proceedings of the IEEE* **86** (Nov, 1998) 2278–2324.

[131] Y. B. M. Courbariaux and J.-P. David, *Low precision storage for deep learning*, *CoRR* **abs/1412.7024** (2014).

[132] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, in *Proceedings of the International Conference on Learning Representations (ICLR)*, pp. 1–14, May, 2015.

[133] F. Alibart, E. Zamanidoost, and D. B. Strukov, *Pattern classification by memristive crossbar circuits using ex situ and in situ training*, *Nature communications* **4** (2013).

[134] M. Hu, H. Li, Y. Chen, Q. Wu, and G. S. Rose, *Bsb training scheme implementation on memristor-based circuit*, in *Computational Intelligence for Security and Defense Applications (CISDA), 2013 IEEE Symposium on*, pp. 80–87, IEEE, 2013.

[135] B. Li, Y. Wang, Y. Wang, Y. Chen, and H. Yang, *Training itself: Mixed-signal training acceleration for memristor-based neural network*, in *ASP-DAC*, pp. 361–366, 2014.

[136] B. Liu, M. Hu, H. Li, Z.-H. Mao, Y. Chen, T. Huang, and W. Zhang, *Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine*, in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2013.

[137] B. Liu, H. Li, Y. Chen, X. Li, T. Huang, Q. Wu, and M. Barnell, *Reduction and ir-drop compensations techniques for reliable neuromorphic computing systems*, in *Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on*, pp. 63–70, Nov, 2014.

[138] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, *Rowclone: Fast and energy-efficient in-DRAM bulk data copy and initialization*, in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, (New York, NY, USA), pp. 185–197, ACM, 2013.

[139] R. A. P. D. D. B. A. A. J. A. A. C. M. F. P. M. S. C. N. B. S. C. T. Z. F. Z. R. A.-I. J. K. T. W. W. R. E. M. T. N. R. S. S. K. Esser, A. Andreopoulos and D. Modha, *Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores*, in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10, Aug, 2013.

[140] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum, *Operating system support for improving data locality on cc-numa compute servers*, in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VII, (New York, NY, USA), pp. 279–289, ACM, 1996.

[141] N. Agarwal, D. Nellans, M. Stephenson, M. O'Connor, and S. W. Keckler, *Page placement strategies for GPUs within heterogeneous memory systems*, in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, (New York, NY, USA), pp. 607–618, ACM, 2015.

[142] M. K. Qureshi, M. M. Franceschini, L. A. Lastras-Montaño, and J. P. Karidis, *Morphable memory system: A robust architecture for exploiting multi-level phase change memories*, in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, (New York, NY, USA), pp. 153–162, ACM, 2010.

[143] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, *Dynamic tracking of page miss ratio curve for memory management*, in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XI, (New York, NY, USA), pp. 177–188, ACM, 2004.

[144] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, *Dot-product engine for neuromorphic computing: Programming 1t1m crossbars for efficient vector-matrix multiplication*, *Tech Reports* (2015) HPL–2015–55.

[145] S. Cho and H. Lee, *Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance*, in *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 347–357, Dec, 2009.

[146] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo, U.-I. Chuang, I.-K. Yong, and K. Kim, *A fast, high-endurance and scalable non-volatile memory device made from asymmetric ta2o5-x/tao2-x bilayer structures*, *Nature Materials* **10** (2011), no. 8 625–630.

[147] C.-W. Hsu, I.-T. Wang, C.-L. Lo, M.-C. Chiang, W.-Y. Jang, C.-H. Lin, and T.-H. Hou, *Self-rectifying bipolar TaOx/TiO2 RRAM with superior endurance over $10^1$2 cycles for 3D high-density storage-class memory*, in *VLSI Technology (VLSIT), 2013 Symposium on*, pp. T166–T167, June, 2013.

[148] S. R. Lee, Y.-B. Kim, M. Chang, K. M. Kim, C. B. Lee, J. H. Hur, G.-S. Park, D. Lee, M.-J. Lee, C. J. Kim, U.-I. Chung, I.-K. Yoo, and K. Kim, *Multi-level switching of triple-layered taox rram with excellent reliability for storage class memory*, in *VLSI Technology (VLSIT), 2012 Symposium on*, pp. 71–72, June, 2012.

[149] D. Niu, C. Xu, N. Muralimanohar, N. P. Jouppi, and Y. Xie, *Design trade-offs for high density cross-point resistive memory*, in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, (New York, NY, USA), pp. 209–214, ACM, 2012.

[150] Y. Yang, J. Mathew, M. Ottavi, S. Pontarelli, and D. Pradhan, *Novel complementary resistive switch crossbar memory write and read schemes*, *IEEE Transactions on Nanotechnology* **14** (March, 2015) 346–357.

[151] A. Kawahara, R. Azuma, Y. Ikeda, K. Kawai, Y. Katoh, K. Tanabe, T. Nakamura, Y. Sumimoto, N. Yamada, N. Nakai, S. Sakamoto, Y. Hayakawa, K. Tsuji, S. Yoneda, A. Himeno, K. Origasa, K. Shimakawa, T. Takagi, T. Mikawa, and K. Aono, *An 8Mb multi-layered cross-point ReRAM macro with 443MB/s write throughput*, in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pp. 432–434, Feb, 2012.

[152] T. yi Liu, T. H. Yan, R. Scheuerlein, Y. Chen, J. Lee, G. Balakrishnan, G. Yee, H. Zhang, A. Yap, J. Ouyang, T. Sasaki, S. Addepalli, A. Al-Shamma, C.-Y. Chen, M. Gupta, G. Hilton, S. Joshi, A. Kathuria, V. Lai, D. Masiwal, M. Matsumoto, A. Nigam, A. Pai, J. Pakhale, C. H. Siau, X. Wu, R. Yin, L. Peng, J. Y. Kang, S. Huynh, H. Wang, N. Nagel, Y. Tanaka, M. Higashitani, T. Minvielle, C. Gorla, T. Tsukamoto, T. Yamaguchi, M. Okajima, T. Okamura, S. Takase, T. Hara, H. Inoue, L. Fasoli, M. Mofidi, R. Shrivastava, and K. Quader, *A 130.7mm2 2-layer 32Gb ReRAM memory device in 24nm technology*, in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pp. 210–211, Feb, 2013.

[153] M. Jung, J. Shalf, and M. Kandemir, *Design of a large-scale storage-class rram system*, in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, (New York, NY, USA), pp. 103–114, ACM, 2013.

[154] C. Xu, P.-Y. Chen, D. Niu, Y. Zheng, S. Yu, and Y. Xie, *Architecting 3D vertical resistive memory for next-generation storage systems*, in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '14, (Piscataway, NJ, USA), pp. 55–62, IEEE Press, 2014.

[155] P. Gu, B. Li, T. Tang, S. Yu, Y. Cao, Y. Wang, and H. Yang, *Technological exploration of rram crossbar array for matrix-vector multiplication*, in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, pp. 106–111, Jan, 2015.

[156] P.-Y. Chen, D. Kadetotad, Z. Xu, A. Mohanty, B. Lin, J. Ye, S. Vrudhula, J.-s. Seo, Y. Cao, and S. Yu, *Technology-design co-optimization of resistive cross-point array for accelerating learning algorithms on chip*, in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 854–859, 2015.

[157] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, *Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning*, in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, (New York, NY, USA), pp. 269–284, ACM, 2014.

[158] Xiangyu Dong *et. al.*, *NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory*, *TCAD* **31** (jul, 2012) 994–1007.

[159] 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 *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 33–38, EDA Consortium, 2012.

[160] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas, *Cacti-io: Cacti with off-chip power-area-timing models*, in *Proceedings of the International Conference on Computer-Aided Design*, pp. 294–301, ACM, 2012.

[161] N. M. N. P. J. C. Xu, D. Niu and Y. Xie, *Understanding the trade-offs in multi-level cell ReRAM memory design*, in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pp. 1–6, IEEE, 2013.

[162] A. Fritsch *et. al.*, *A 4GHz, low latency TCAM in 14nm SOI FinFET technology using a high performance current sense amplifier for AC current surge reduction*, in *ESSCIRC*, pp. 343–346, sep, 2015.

[163] S. Matsunaga *et. al.*, *A 3.14 um2 4T-2MTJ-cell fully parallel TCAM based on nonvolatile logic-in-memory architecture*, in *VLSIC*, pp. 44–45, jun, 2012.

[164] J. Li *et. al.*, *1 Mb 0.41 um2 2T-2R Cell Nonvolatile TCAM With Two-Bit Encoding and Clocked Self-Referenced Sensing*, *JSSC* **49** (apr, 2014) 896–907.

[165] C.-C. Lin *et. al.*, *A 256b-Wordlength ReRAM-based TCAM with 1ns Search-Time and 14x Improvement in WordLength- EnergyEfficiency-Density Product using 2.5T1R cell*, in *ISSCC*, pp. 136–138, 2016.

[166] H.-J. Tsai *et. al.*, *Energy-efficient non-volatile TCAM search engine design using priority-decision in memory technology for DPI*, in *DAC*, pp. 1–6, june, 2015.

[167] Q. Guo *et. al.*, *AC-DIMM: associative computing with STT-MRAM*, in *ISCA*, pp. 189–200, 2013.

[168] N. Muralimanohar *et. al.*, *CACTI 6.0: A tool to model large caches*, *HP Lab.* (2009) 22–31.

[169] S. Li *et. al.*, *CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques*, in *ICCAD*, pp. 694–701, nov, 2011.

[170] B. Agrawal and T. Sherwood, *Ternary CAM Power and Delay Model: Extensions and Uses*, *TVLSI* **16** (may, 2008) 554–564.

[171] I. Bayram and Y. Chen, *NV-TCAM: Alternative interests and practices in NVM designs*, in *NVMSA*, pp. 1–6, aug, 2014.

[172] K. Pagiamtzis *et. al.*, *Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey*, *JSSC* **41** (mar, 2006) 712–727.

[173] S. Matsunaga *et. al.*, *Fully parallel 6T-2MTJ nonvolatile TCAM with single-transistor-based self match-line discharge control*, in *VLSIC*, pp. 298–299, jun, 2011.

[174] M.-F. F. Chang *et. al.*, *A 3T1R nonvolatile TCAM using MLC ReRAM with Sub-1ns search time*, *JSSC* **58** (feb, 2015) 318–319.

[175] L.-Y. Huang *et. al.*, *ReRAM-based 4T2R nonvolatile TCAM with 7x NVM-stress reduction, and 4x improvement in speed-wordlength-capacity for normally-off instant-on filter-based search engines used in big-data processing*, in *VLSIC*, pp. 1–2, jun, 2014.

[176] M.-F. Chang *et. al.*, *An Offset-Tolerant Fast-Random-Read Current-Sampling-Based Sense Amplifier for Small-Cell-Current Nonvolatile Memory*, *JSSC* **48** (mar, 2013) 864–877.

[177] S. Matsunaga *et. al.*, *Implementation of a perpendicular MTJ-based read-disturb-tolerant 2T-2R nonvolatile TCAM based on a reversed current reading scheme*, in *ASP-DAC*, pp. 475–476, jan, 2012.

[178] Chung-Hsun Huang *et. al.*, *Design of high-performance CMOS priority encoders and incrementer/decrementers using multilevel lookahead and multilevel folding techniques*, *JSSC* **37** (2002), no. 1 63–76.

[179] "Predictive Technology Model." `http://ptm.asu.edu/`.

[180] S. Zuloaga *et. al.*, *Scaling 2-layer RRAM cross-point array towards 10 nm node: A device-circuit co-design*, in *ISCAS*, pp. 193–196, may, 2015.

[181] C. Xu *et. al.*, *Architecting 3D vertical resistive memory for next-generation storage systems*, in *ICCAD*, pp. 55–62, nov, 2014.

[182] I. G. Baek *et. al.*, *Realization of vertical resistive memory (VRRAM) using cost effective 3D process*, in *IEDM*, pp. 31.8.1–31.8.4, dec, 2011.

[183] C. Xu *et. al.*, *Modeling and design analysis of 3D vertical resistive memory: A low cost cross-point architecture*, in *ASP-DAC*, pp. 825–830, jan, 2014.

[184] E. Cha *et. al.*, *Nanoscale (10nm) 3D vertical ReRAM and NbO2 threshold selector with TiN electrode*, in *IEDM*, pp. 10.5.1–10.5.4, dec, 2013.

[185] "BLAST." `http://blast.ncbi.nlm.nih.gov/Blast.cgi`.

[186] "GenBank." `http://www.ncbi.nlm.nih.gov/genbank/statistics/`.

[187] L. Gwennap, *Skylake speedshifts to next gear*, *Microprocessor Report* **29** (2015), no. 9 6–10.

[188] *NVIDIA Volta*, 2017.

[189] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, *A case for intelligent RAM*, *Micro, IEEE* **17** (1997), no. 2 34–44.

[190] *NVIDIA TITAN X (pascal)*, 2016. http://www.geforce.com/hardware/10series/titan-x-pascal.

[191] K. Chen and L. Pachter, *Bioinformatics for whole-genome shotgun sequencing of microbial communities*, *PLoS Comput Biol* **1** (2005), no. 2 e24.

[192] F. Hamzaoglu, U. Arslan, N. Bisnik, S. Ghosh, M. B. Lal, N. Lindert, M. Meterelliyoz, R. B. Osborne, J. Park, S. Tomishima, Y. Wang, and K. Zhang, *13.1 A 1Gb 2GHz embedded DRAM in 22nm tri-gate CMOS technology*, in *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 230–231, IEEE, feb, 2014.

[193] G. Sideris, *INTEL 1103-MOS memory taht defied cores*, *ELECTRONICS* **46** (1973), no. 9 108–113.

[194] D. H. Neil Weste, *CMOS VLSI Design: A Circuits And Systems Perspective, 3/E*. Pearson, 2006.

[195] D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*. Newnes, 2013.

[196] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, *A Case for Exploiting Subarray-level Parallelism (SALP) in DRAM*, in *International Symposium on Computer Architecture (ISCA)*, pp. 368–379, IEEE Computer Society, 2012.

[197] 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 *Proc. 53rd Annu. Des. Autom. Conf. - DAC '16*, (New York, New York, USA), pp. 1–6, ACM Press, 2016.

[198] *8Gb B-die DDR4 SDRAM*, 2016. http://www.samsung.com/semiconductor/global/file/product/2016/06/DS_K4A8G085WB-B_Rev1_61-0.pdf.

[199] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, *Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation*, in *International Symposium on Computer Architecture (ISCA)*, pp. 349–360, jun, 2014.

[200] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. H. Ahn, *Reducing memory access latency with asymmetric DRAM bank organizations*, *International Symposium on Computer Architecture (ISCA)* **41** (jul, 2013) 380.

[201] "Micron Automata Processor." https://www.micronautomata.com/.

[202] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, *AC-DIMM: associative computing with STT-MRAM*, in *nternational Symposium on Computer Architecture (ISCA)*, pp. 189–200, ACM, 2013.

[203] "Torch 7." http://torch.ch/.

[204] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, *Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM*, *CoRR* **abs/1611.09988** (2016).

[205] J. M. Park, Y. S. Hwang, S. W. Kim, S. Y. Han, J. S. Park, J. Kim, J. W. Seo, B. S. Kim, S. H. Shin, C. H. Cho, S. W. Nam, H. S. Hong, K. P. Lee, G. Y. Jin, and E. S. Jung, *20nm DRAM: A new beginning of another revolution*, in *2015 IEEE International Electron Devices Meeting (IEDM)*, pp. 26.5.1–26.5.4, 2015.

[206] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM Circuit Design: Fundamental and High-Speed Topics*. Wiley-IEEE Press, 2nd ed., 2007.

[207] P. J. Nair, D.-H. Kim, and M. K. Qureshi, *ArchShield: architectural framework for assisting DRAM scaling by tolerating high error rates*, in *International Symposium on Computer Architecture*, (New York, New York, USA), pp. 72–83, ACM Press, 2013.

[208] S. Han, H. Mao, and W. J. Dally, *Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding*, *CoRR* **abs/1510.00149** (2015).

[209] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, *DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients*, *CoRR* **abs/1606.06160** (2016).

[210] F. Li and B. Liu, *Ternary Weight Networks*, *CoRR* **abs/1605.04711** (2016).

[211] G. Venkatesh, E. Nurvitadhi, and D. Marr, *Accelerating Deep Convolutional Networks using low-precision and sparsity*, *CoRR* **abs/1610.00324** (2016).

[212] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, *Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations*, *CoRR* **abs/1609.07061** (2016).

[213] M. Courbariaux, Y. Bengio, and J. David, *BinaryConnect: Training Deep Neural Networks with binary weights during propagations*, *CoRR* **abs/1511.00363** (2015).

[214] M. Courbariaux and Y. Bengio, *BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1*, *CoRR* **abs/1602.02830** (2016).

[215] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks*, *CoRR* **abs/1603.05279** (2016).

[216] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105. Curran Associates, Inc., 2012.

[217] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, *CoRR* **abs/1512.03385** (2015).

[218] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going Deeper with Convolutions*, *CoRR* **abs/1409.4842** (2014).

[219] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, *CoRR* **abs/1409.1556** (2014).

[220] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, *International Journal of Computer Vision (IJCV)* **115** (2015), no. 3 211–252.

[221] Intel, "Intel Instruction Set Architecture Extensions." `https://software.intel.com/en-us/intel-isa-extensions`.

[222] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, *Cambricon: An Instruction Set Architecture for Neural Networks*, in *International Symposium on Computer Architecture (ISCA)*, pp. 393–405, IEEE, jun, 2016.

[223] J. Ott, Z. Lin, Y. Zhang, S. Liu, and Y. Bengio, *Recurrent Neural Networks With Limited Numerical Precision*, *CoRR* **abs/1608.06902** (2016).

[224] S.-M. Kang and Y. Leblebici, *CMOS digital integrated circuits*. Tata McGraw-Hill Education, 2003.

[225] Ke Chen, Sheng Li, N. Muralimanohar, Jung Ho 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)*, pp. 33–38, EDA Consortium, IEEE, mar, 2012.

[226] "Design Compiler, Synopsys Inc.." http://www.synopsys.com/Tools/Implementation/RTLSynthesis/ DesignCompiler.

[227] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, *ShiDianNao: shifting vision processing closer to the sensor*, in *International Symposium on Computer Architecture (ISCA)*, (New York, New York, USA), pp. 92–104, ACM Press, 2015.

[228] "NVIDIA cuDNN." https://developer.nvidia.com/cudnn.

[229] "NVIDIA System Management Interface." https://developer.nvidia.com/nvidia-system-management-interface.

[230] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, *HotSpot: a compact thermal modeling methodology for early-stage VLSI design*, *IEEE ransactions on Very Large Scale Integration (VLSI) Systems* **14** (may, 2006) 501–513.

[231] Yasuko Eckert Nuwan Jayasena and G. Loh, *Thermal Feasibility of Die-Stacked Processing in Memory*, in *WoNDP: 2nd Workshop on Near-Data Processing, International Symposium on Microarchitecture*, IEEE, 2014.

[232] T. Vogelsang, *Understanding the Energy Consumption of Dynamic Random Access Memories*, in *International Symposium on Microarchitecture (MICRO)*, pp. 363–374, IEEE, dec, 2010.

[233] *IC Cost and Price Model, Revision 1506, IC Knowledge LLC.*, 2015. http://www.icknowledge.com/.

[234] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, *DRISA: A DRAM-based Reconfigurable In-Situ Accelerator*, in *International Symposium on Microarchitecture (MICRO)*, pp. 288–301, ACM, 2017.

[235] 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 *International Symposium on Microarchitecture (MICRO)*, pp. 273–287, ACM, 2017.

[236] A. Alaghi and J. P. Hayes, *Survey of Stochastic Computing*, *ACM Transactions on Embedded Computing Systems* **12** (may, 2013) 1–19.

[237] B. R. Gaines, *Stochastic computing*, in *Proceedings of the spring joint computer conference*, 1967.

[238] J. P. Hayes, *Introduction to stochastic computing and its challenges*, in *Proceedings of the Annual Design Automation Conference*, 2015.

[239] J. H. Anderson, Y. Hara-Azumi, and S. Yamashita, *Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy*, in *Design, Automation & Test in Europe Conference & Exhibition*, pp. 1550–1555, IEEE, 2016.

[240] K. Kim, J. Lee, and K. Choi, *An energy-efficient random number generator for stochastic circuits*, in *Asia and South Pacific Design Automation Conference*, pp. 256–261, IEEE, jan, 2016.

[241] H. Sim, D. Nguyen, J. Lee, and K. Choi, *Scalable stochastic-computing accelerator for convolutional neural networks*, in *Asia and South Pacific Design Automation Conference*, pp. 696–701, IEEE, jan, 2017.

[242] A. Ren, Z. Li, C. Ding, Q. Qiu, Y. Wang, J. Li, X. Qian, and B. Yuan, *SC-DCNN: Highly-Scalable Deep Convolutional Neural Network using Stochastic Computing*, *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems* (2017) 405–418.

[243] K. Kim, J. Lee, and K. Choi, *Approximate de-randomizer for stochastic circuits*, in *International SoC Design Conference*, pp. 123–124, IEEE, nov, 2015.

[244] S. Gupta, V. Sindhwani, and K. Gopalakrishnan, *Learning Machines Implemented on Non-Deterministic Hardware*, *arXiv Prepr. arXiv1409.2620* (2014).

[245] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, *Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks*, in *Proceedings of the Annual Design Automation Conference*, 2016.

[246] D. Braendler, T. Hendtlass, and P. O'Donoghue, *Deterministic bit-stream digital neurons*, *IEEE Transactions on Neural Networks* **13** (nov, 2002) 1514–1525.

[247] D. Jenson and M. Riedel, *A deterministic approach to stochastic computation*, in *Proceedings of International Conference on Computer-Aided Design*, 2016.

[248] *NVIDIA GPU*, 2016. `http://www.nvidia.com`.

[249] Micron, *Micron announces development of new parallel processing architecture*, .

[250] B. Y. Zhe Li, Ao Ren, Ji Li, Qinru Qiu, Yanzhi Wang, *DSCNN: Hardware-Oriented Optimization for Stochastic Computing Based Deep Convolutional Neural Networks*, in *Internetional Conference on Comupter Design*, 2016.

[251] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, *FreePDK: An open-source variation-aware design kit*, in *International Conference on Microelectronic Systems Education (MSE)*, pp. 173–174, IEEE, 2007.

[252] M. Sung, S. A. Jang, H. Lee, Y. H. Ji, J. I. Kang, T. O. Jung, T. H. Ahn, Y. I. Son, H. C. Kim, S. W. Lee, S. M. Lee, J. H. Lee, S. B. Baek, E. H. Doh, H. J. Cho, T. Y. Jang, I. S. Jang, J. H. Han, K. B. Ko, Y. J. Lee, S. B. Shin, J. S. Yu, S. H. Cho, J. H. Han, D. K. Kang, J. Kim, J. S. Lee, K. D. Ban, S. J. Yeom, H. W. Nam, D. K. Lee, M. M. Jeong, B. Kwak, J. Park, K. Choi, S. K. Park, N. J. Kwak, and S. J. Hong, *Gate-first high-k/metal gate dram technology for low power and high performance products*, in *2015 IEEE International Electron Devices Meeting (IEDM)*, pp. 26.6.1–26.6.4, Dec, 2015.

[253] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, *Low-Cost Inter-Linked Subarrays (LISA): Enabling fast inter-subarray data movement in DRAM*, in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 568–580, IEEE, mar, 2016.

[254] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik, *Comparison of learning algorithms for handwritten digit recognition*, in *International conference on artificial neural networks*, vol. 60, pp. 53–60, Perth, Australia, 1995.

[255] Y. LeCun, C. Cortes, and C. J. Burges, *Mnist handwritten digit database*, *AT&T Labs* **2** (2010).

[256] "Efficient, reusable RNNs and LSTMs for torch." `https://github.com/jcjohnson/torch-rnn`.