2023

A Phase Change Memory and DRAM Based Framework For Energy-Efficient and High-Speed In-Memory Stochastic Computing

Supreeth Mysore
University of Kentucky, ssh366@uky.edu
Digital Object Identifier: https://doi.org/10.13023/etd.2023.191

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Supreeth Mysore, Student

Dr. Ishan Thakkar, Major Professor

Dr. Daniel Lau, Director of Graduate Studies

A Phase Change Memory and DRAM Based Framework For Energy-Efficient and High-Speed In-Memory Stochastic Computing

---

DISSERTATION

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the College of Electrical and Computer Engineering at the University of Kentucky

By
Supreeth Mysore Shivanandamurthy
Lexington, Kentucky

Director: Dr. Ishan G. Thakkar, Assistant Professor of Electrical and Computer Engineering
Co-Director : Dr. Sayed Ahmad Salehi, Assistant Professor of Electrical and Computer Engineering
Lexington, Kentucky 2023

ABSTRACT OF DISSERTATION

A Phase Change Memory and DRAM Based Framework For Energy-Efficient and High-Speed In-Memory Stochastic Computing

Convolutional Neural Networks (CNNs) have proven to be highly effective in various fields related to Artificial Intelligence (AI) and Machine Learning (ML). However, the significant computational and memory requirements of CNNs, especially the multiply-accumulate (MAC) operations that are fundamental building blocks of CNNs, make their processing highly challenging. As the input dataset size increases, the traditional processor-centric, von-Neumann computing architectures become ill-suited for processing CNNs because they exponentially increase the latency and energy costs of processing CNNs.

To overcome these challenges, researchers have explored the Processing-In-Memory (PIM) technique, which involves placing the processing unit inside or near the memory unit. This approach reduces data migration length and utilizes the internal memory bandwidth at the memory bank and subarray levels. However, developing an efficient PIM-based system with minimal hardware complexity remains a significant challenge.

The proposed PIM solution in this thesis report suggests utilizing different memory technologies, such as Dynamic RAM (DRAM) and phase change memory (PCM), with stochastic arithmetic and minimal add-on logic. Stochastic computing is a technique that uses pseudo-random numbers to perform arithmetic operations. This technique reduces hardware requirements for CNNs' arithmetic operations, making it possible to implement them with simple logic gates.

This thesis report details techniques and architectures to enable DRAM and PCM-based PIM systems that can employ stochastic arithmetic to accelerate CNNs. The detailed topics include PCM-based scalable Stochastic Number Generator (SNG), DRAM-based stochastic MAC accelerator for CNNs, non-volatile memory (NVM) class PCRAM-based MAC accelerator for CNNs, and DRAM-based stochastic to binary conversion (StoB). The report also identifies future research directions to enable highly scalable, energy-efficient, and high-speed PIM accelerators for CNNs. for the proposed designs, including in-situ PCRAM-based SNG, ODIN (A Bit-Parallel Stochastic Arithmetic Based Accelerator for In-Situ Neural Network Processing in Phase Change RAM), ATRIA (Bit-Parallel Stochastic Arithmetic Based Accelerator for In-DRAM CNN Processing), and AGNI (In-Situ, Iso-Latency Stochastic-to-Binary Number Conversion for In-DRAM Deep Learning), and presents initial findings for these ideas.

In summary, the proposed solution in the report offers a comprehensive approach to address the challenges of processing CNNs. The proposed designs have the potential to significantly improve the energy and time efficiency of accelerating CNNs. Using Stochastic Computing with different memory technologies enables the development of efficient PIM-based systems with minimal hardware overheads and complexity, providing a promising path for the future of CNN-based applications.

KEYWORDS: In-Memory Computing, Convolution Neural Network, Stochastic Computing, Artificial Intelligence

Author's signature: Supreeth Mysore Shivanandamurthy

Date: May 5, 2023

A Phase Change Memory and DRAM Based Framework For Energy-Efficient and High-Speed In-Memory Stochastic Computing

By
Supreeth Mysore Shivanandamurthy

Director of Dissertation: Dr. Ishan G. Thakkar

Co-Director of Dissertation: Dr. Sayed Ahmad Salehi

Director of Graduate Studies: Dr. Daniel Lau

Date: May 5, 2023

DEDICATED TO MY FAMILY AND TEACHERS
Thanks for leading me from the darkness of ignorance
to the quest for the immortality of knowledge.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my gratitude to several individuals who have contributed significantly to the successful completion of my work. Without their support, guidance, and motivation, this accomplishment would not have been possible.

First and foremost, I would like to extend my heartfelt gratitude to my advisor, Dr. Ishan Thakkar, for his unwavering support and guidance throughout my research. Dr. Thakkar has consistently provided me with the latest ideas, relevant literature, and infrastructure, which have been critical to the completion of my work. His constant encouragement and constructive feedback have been invaluable, and I am grateful for his mentorship.

I would also like to acknowledge my family, who has been my pillar of strength throughout this journey. Their unconditional love, support, and sacrifice have been an inspiration to me, particularly during the challenging times of the pandemic. Their unwavering encouragement and motivation have been the driving force behind my accomplishments, and I am grateful for their unwavering support.

Moreover, I would like to express my appreciation to my fellow labmates at UCAT lab, UKY, for their insightful discussions and collaborative efforts. Their support, camaraderie, and shared knowledge have been instrumental in making my research journey an enjoyable and memorable one.

Finally, I would like to thank Dr. Salehi for his support and guidance during my research studies. His valuable insights, suggestions, and expertise have been instrumental in shaping my work, and I am grateful for his mentorship.

Once again, I extend my sincere appreciation to all those who have contributed to the successful completion of my work, and I look forward to continuing my research journey with their support and guidance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

The advent of big data has transformed the way we use technology [5]. Smart gadgets and applications rely on massive datasets that require advanced algorithms like machine learning, deep learning, and neural networks to extract meaningful information. Convolutional neural networks (CNNs) have emerged as powerful tools for visual image analysis and are extensively used in many fields, including computer vision, speech recognition, and natural language processing. A detailed explanation of CNNs will be provided in Chapter 2.

However, the high computational demands of CNNs pose a significant challenge for traditional computing systems. The convolution and fully connected layers, which are essential to CNNs, consist of many multiply and accumulate (MAC) operations that are both computation and data-intensive (refer to Chapter 2). Traditional computing systems are not designed to handle this massive amount of data, leading to extravagant energy and latency costs. They are facing significant challenges in keeping up with the computational needs of modern CNN applications. Moreover, frequent data movements between and processing and memory cores of these systems also add substantial energy and latency costs while processing CNNs. In Chapter 2, we discuss in detail the drawbacks of traditional computing systems in the context of processing CNN applications.

To mitigate these issues, there is a pressing need for alternative computing systems that are simpler to realize and have low-overhead hardware organization, with increased energy efficiency for arithmetic operations. The envisioned alternative computing systems should also address the issue of performance degradation and enable faster processing of CNNs.

To that end, to address the challenge of frequent data movements in traditional von-Neumann architectures, researchers have proposed using the processing-in-memory (PIM) technique and near-data computing [6]. This work is concentrated on the PIM technique, wherein the memory units are instilled with some in-situ processing capabilities. Due to this, the required data movements reduce drastically, resulting in increased processing throughput, latency and energy consumption. However, designing PIM architectures have its own challenges. First, the fabrication cost involved in the hardware modifications to enable PIM inside traditional main memory should be minimized. Second, the hardware circuitry that powers PIM architectures should be energy- and area-efficient.

To address this hardware efficiency challenge, researchers have come up with alternative arithmetic options such as stochastic computing and approximate computing [7] [8] [6]. Our research focuses on implementing the computations required for processing CNNs in the stochastic domain. Stochastic arithmetic operations are simpler to implement and have less hardware footprint compared to binary computation. However, stochastic computing suffers from limitations, namely the high area and energy overheads, and low performance of conventional circuits used for binary to stochastic number conversion. We address these shortcomings to enable stochastic computing in PIM architectures.

This dissertation focuses on developing alternative computing systems for faster processing of CNNs with high energy efficiency. Specifically, we explore two types of CNN accelerators: volatile memory-based (i.e., DRAM-based) and phase-change memory (PCM)-

based PIM designs. Our initial work involves designing a novel stochastic computing-based PIM framework for CNN acceleration using mature DRAM technology. This framework utilizes bit-parallel stochastic computing, as described in Chapter 5. To further enhance the performance of our DRAM-based PIM accelerators for CNNs, we propose a new iso-latency stochastic-to-binary conversion substrate, which is detailed in Chapter 6. Further, one potential alternative to DRAM is PCM, which offers scalability and energy efficiency benefits. Considering the popularity of PCM technology, we extended our idea of using the stochastic computing-based acceleration of CNNs to realize a PCM-based PIM architecture. This architecture is explained in detail in Chapter 4. Furthermore, we also contributed to the development of a scalable stochastic number generator (SNG) using the inherent stochasticity of PCM. This is explained in Chapter 3.

The contributions of this report are enumerated in Section 1.1. The report outline is provided in Section 1.2.

1.1 Contributions

This thesis report makes the following contributions.

Scalable Stochastic Number Generator for Phase Change Memory (PCM) Based In-Memory Stochastic Processing

In this study, we present the design of a high-performance and low-area footprint SNG that utilizes the inherently stochastic nature of PCM. We demonstrate the effectiveness of our PCM-based SNG by implementing it for different binary input lengths and comparing it with state-of-the-art Linear Feedback Shift Register (LFSR)-based SNG designs. We also identify the workflow towards achieving SNG scalability with minimal hardware footprint and discuss the challenges and constraints associated with the PCM-based SNG design. In Chapter 3, we provide a detailed explanation of our proposed SNG's energy and area-saving, which are 250× and 300×, respectively, compared to conventional LFSR-based SNG designs at 14-bit binary input length.

ODIN: A Bit-Parallel Stochastic Arithmetic-Based Accelerator for In-situ Neural Network Processing in Phase Change RAM

We introduce a new PIM engine, named ODIN, that combines the advantages of stochastic computing and PIM architecture to facilitate the low-overhead in-situ acceleration of essential CNN functions such as multiply-accumulate (MAC), nonlinear activation, and pooling. ODIN employs hybrid binary-stochastic bit-parallel arithmetic and Phase Change RAM (PCRAM) as the underlying memory technology to achieve energy-efficient computation.

To assess ODIN's performance, we mapped four CNN benchmark applications on the proposed accelerator and compared it with a conventional processor-centric design and a crossbar-based in-situ CNN accelerator from prior research. Our analysis results indicate that the ODIN accelerator can achieve at least 5.8x faster and 23.2× more energy-efficient computation compared to the processor-centric design, and up to 90.8× faster and 1554×

more energy-efficient computation compared to the crossbar-based in-situ CNN accelerator from prior work. A detailed description of these findings is presented in Chapter 4.

ATRIA: Low-latency, Energy-efficient In-DRAM CNN Acceleration With Bit-parallel Unary Computing

We introduce ATRIA, a novel in-DRAM accelerator that utilizes bit-parallel rate-coded unary computing for energy-efficient and high-speed inference of CNNs. ATRIA employs lightweight modifications to DRAM cell arrays to implement stochastic computing and accelerate MAC operations inside DRAM. By performing 16 MAC operations in only two consecutive memory operation cycles, *ATRIA* significantly enhances the latency, throughput, and efficiency of CNN inference processing. Chapter 5 provides a detailed explanation of these improvements.

AGNI:In-situ, ISO-latency Stochastic-to-Binary Number Conversion for In-DRAM Deep Learning

Additionally, we present AGNI, a new technique for in-DRAM stochastic-to-binary number conversion for deep learning applications. AGNI makes minor modifications to existing DRAM peripherals and enhances the performance of existing in-DRAM stochastic-to-binary conversion circuits by reducing area, energy-delay product, and latency. Our SPICE simulations demonstrate that AGNI can achieve a 3.9× improvement in performance across four deep CNN models. Chapter 6 provides a comprehensive explanation of AGNI and its simulation results.

1.2 Report outline

The report is thoughtfully structured to present a comprehensive account of our research work. In Chapter 1, we provide an introduction to the report, highlighting the motivation and objectives of our research. Chapter 2 offers a review of the existing literature and the required background.

We present our research contributions in Chapters 3, 4, 5, and Chapter 6, with each chapter focusing on a specific contribution of our research. Chapter 3 elaborates on our proposed workflow for a PCM-based SNG and discusses potential future work. Chapter 4 discusses ODIN, a stochastic MAC engine using PCM memory structure, along with our future plans for this research. Similarly, Chapter 5 focuses on ATRIA. In Chapter 6, we discuss our AGNI substrate.Finally, Chapter 7 concludes our report by summarizing our findings and contributions. We also highlight future scopes related to precision and correlation manipulation techniques to address stochastic inaccuracy in Chapter 8, Chapter 9, and Chapter 10.

Chapter 2 Background and Related Work

2.1 Convolution Neural Networks (CNNs)

In the present day, cloud computing and big data are very popular. It consists of an immense dataset to process and extract meaningful information. For applications, such as image recognition, computer vision, and other augmented reality-based technology, scholars found that convolution neural networks are a very favorable candidate and accelerated the growth of big data significantly. For our dissertation, we are focusing on the application related to the convolution neural network. Thus, it is necessary to give a brief overview of the generic CNNs, the structure of neural network layers, and the arithmetical operation involved in each neural network layer. Generally speaking, CNNs consist of four major types of layers, as shown in Figure 1, namely convolution (CONV), local response normalization (LRN), max pooling (POOL), and fully connected (FC) layers [9]. From all these layers, the major portion of information extraction of the input data takes place in the CONV and FC layers, where a very large number of MAC operations occur on the synaptic weights and input biases. These MAC operations typically involve many steps for intermediate value preparation and storage. Therefore, they can be highly time-consuming and energy-consuming. To give better insight, consider Equation 2.1, related to an FC layer, which consists of MAC operations between $w_{i,j's}$ and $a_{i's}$.

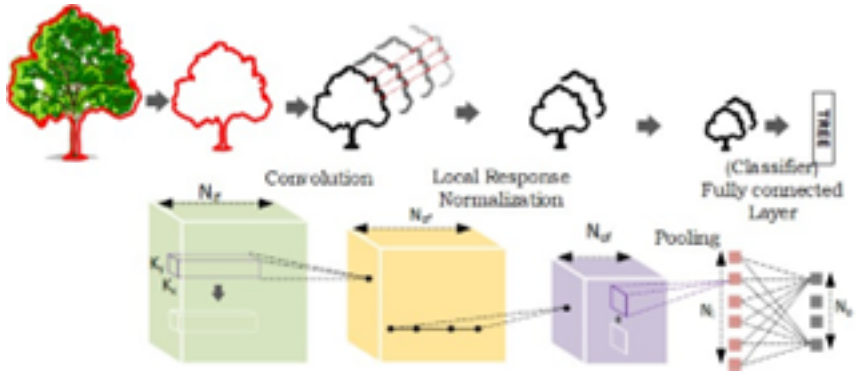

Figure 2.1: Four major types of layers found in Convolutional Neural Networks (CNNs) [1].

Here in Equation 2.1, $w_{i,j's}$ is the synaptic weight connecting ith node in layer 'a'to j^{th} node in layer 'b', and 'f' is a non-linear activation function (in our dissertation we use ReLU activation function).

$$\text{out}(x,y)^{f_0} = \sum_{f_i=0}^{N_i} \sum_{k_x=0}^{K_x} \sum_{k_y=0}^{K_y} (w_{f_i,f_0}(k_x,k_y) \cdot \text{in}(k_x, y+k_y)^{f_i}) \tag{2.1}$$

2.2 State of art machine learning technique

In this subsection, we are explaining the layers and NN's mathematical operations.

Convolution layer

This layer is used to identify the characteristic elements of input data. For example, see Figure 2.1. The filter is formed by $K_x \times K_y$ coefficient kernel; these kernels are learned and form the synaptic weights for the next layer. The formula below shows the concrete method to find the output neuron at position (x, y) of output feature map f^0:

From the above Equation 2.2 input feature map fi can have multiple feature maps and the kernel is usually three dimensional i.e., $K_x \times K_y \times N_{if}$.

$$\text{out}(x, y)^{f_0} = \max_{0 \le k_x \le K_x, 0 \le k_y \le K_y} \tag{2.2}$$

Pooling layer (POOL)

This layer computes the simple operation of max and average over the neighboring points. See Figure 2.1. Unlike convolutional or a classifier layer, a pooling layer has no learned synaptic weight and is represented as follows:

Local Response Normalization (LRN)

LRN implements the competition between multiple neurons at the same location. The function is like the lateral inhibition found in biological neurons. The impact of the LRN on the accuracy is negligible. Our work is not focused on implementing the LRN layers as evidenced by [10] [11] as shown in the below equation 2.3.

$$\text{out}(j) = t\left(\sum_{i=0}^{N_i} w_{i,j} \cdot \text{in}(i)\right) \tag{2.3}$$

Classification layer

Finally, the results of the sequence of Pooling, CONV, and LRN layers are fed to the classification layer. This layer is also called a fully connected layer. Check-in Figure 2.2, each N_i node is connected to output node No. The main function of this layer is to correlate the different features extracted from the filtering, pooling, and normalization to predict the output categories. Where $w_{i,j}$ is the synaptic weights, t( ) is a transfer function. It can be tanh(x), max(0,x) and ReLU. Our research work considered the ReLU activation function as shown in equation 2.1.

2.3 Processing in-memory (PIM)

Introduction

In this subchapter, we are going to discuss the conventional computer architecture and its drawback for big data applications. We also discuss the mitigation technique with the in-memory processing methodology especially using DRAM and NVM-based memory architecture.

Motivation

As explained in the earlier Section, CNN is being adopted by a wide spectrum of application domains such as natural language processing, image recognition, and computer vision. Nowadays, NN models employ increasingly enormous numbers of datasets and parameters. For example, Alexnet [12] [13] and VGG [14] are exercised in image classification with almost 61M and 138M parameters respectively. Keeping in mind this enormous data, the training of such a complex model demands enormous computation, memory resources, latency time, and energy. One critical aspect of this computation is the energy and large bandwidth requirement. These needs keep growing with the increase in parameters of dataset classification. But in present days the NN models are becoming deeper and larger; the data set and pressure on the faster runtime system need to keep growing. To mitigate this requirement, investigators come up with the low precision CNN model [15] and prune CNN [16], owing to the tradeoff in accuracy. Even these techniques try to improve the computation speed but have not addressed the key issue of CNN training model data migration requirement.

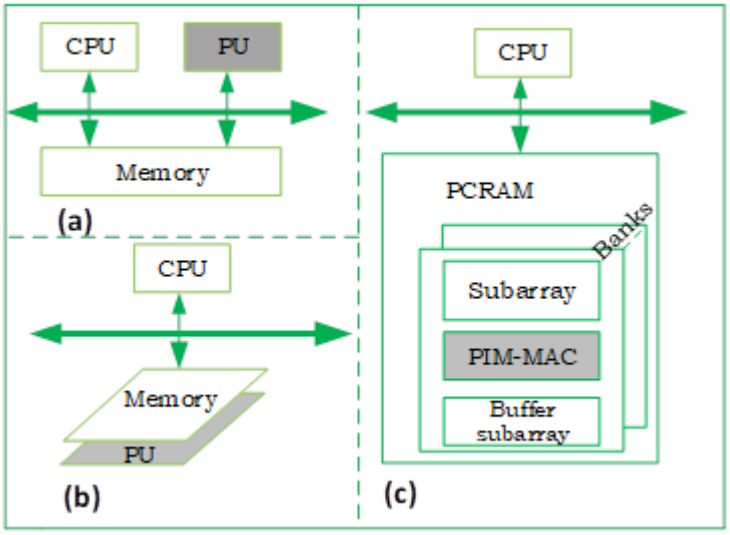

Figure 2.2: a) Generic computer architecture b) Generic PIM approach to implement processing unit inside main memory c) Proposed design framework for PCRAM-based PIM implementation (can also be implemented in DRAM).

Need of processing-in-memory computing

To address the data migration length challenges as explained before, researchers are considering the option of processing-in-memory technique (PIM). PIM is not a recent technique. It was actually proposed by IBM research in 2004, but most of the applications during previous years do not involve high data set volumes. One of the earliest PIM architecture examples is Stone's logic-based in-memory computer. Also, the cost of adding the extra circuitry (i.e., processing unit) inside the memory is not feasible due to the fabrication issues [17] [18]. However, big data with a conventional computer system (i.e., von-Neumann

architecture) is not able to suffice the system performance requirement. Thus, it is inevitable for today's digital world to incorporate the PIM architecture to meet the computing system requirements and accelerate the big data and cloud computing applications. There exist a few PIM architectures proposed by the researchers such as NON-VON [2], EXECUBE [19], Terasys [20], Computational RAM [21], and IRAM [22]. This architecture constitutes the add-on logic inside the DRAM to perform the addition operation in parallel. But many of these architectures were hindered by the setbacks/physical limitations of memory technology (i.e., DRAM) that prevents the integration of processing unit (PU) inside the memory.

New opportunity in modern memory

The researcher communities are of the opinion that the demand for large memory density by modern applications such as data learning with DRAM scaling has been pushed to their complete physical limits [23] [24]. It is becoming more problematic to increase the memory density [9] with reduced latency [25] [26] and improve the energy efficiency of DRAM-based memory architecture [27]. Consequentially, memory-related research companies are exploring the alternative to DRAM and also addressing the prior issues. Researchers have developed two solutions to overcome the limitations exhibited in prior memory systems for PIM implementation.

The first invention is the 3D stacked memory [18] [28]. In this technology, multiple layers of memory (in general DRAM) are stacked one above the other [29]. In addition to the multiple layers-based stacked 3D-DRAM memory architecture, the researchers have incorporated PU in one of the layers. Thus, the 3D-stacked-based technique reduces the data migration length thereby improving the system performance.

Second, researchers have found the major breakthrough of emerging non-volatile memory (NVM) can be a replacement for main memory subsystems. The main advantage of this NVM technology is the near-zero leakage current compared to the high leakage current in DRAM. Hence the NVM-based design exhibits better scalability and improves the memory density for data storage. The memory manufacturers are mainly considering three types of NVMs to replace the DRAM system at the main memory level:

- Phase change memory (PCM) [30]
- Magnetic RAM (MRAM) [31]
- Metal-oxide resistive RAM (RRAM) / memristor [32] [33]

All of the above mentioned NVM-based techniques are expected to provide the same memory latencies and energy consumption close to DRAM from literature studies. Furthermore, recent literature related to the NVM state [1] that the NVM is not only used for storing the data but can be used as the processing unit. In addition, each cell of NVM mimics the characteristics of a human neuron, which resemble the basic functionality of neural network character. Consequently, the hardware circuitry to implement the NN by using NVM-based memory technology can reduce the area requirement significantly. This idea attracted much research in PIM architecture using NVM-based systems. In addition,

Table 2.1: Advantages and disadvantages of SC

Features	Advantages	Disadvanatages
Operating Speed	short clock period Massive Parallelism	Long bit stream
Result quality	high error tolerance progressive precision	Low precision random number fluctuations correlations included inaccuracies
Circuit size and power	Tiny arithmetic components	More random number sources and stochastic to binary conversion ckts
Design Issues	Rich set of arithmetic components	Little CAD tool support at present

numerous PIM works related to the NVM-based cells are processed in recent times to perform Boolean logic operations [34] [35].

Some examples include deep learning (DL), artificial intelligence (AI), voice recognition etc., used in healthcare, data-science management, and statistical investigation. These applications consist of tremendous data information for processing. von-Neumann led to a lot of data migration from the main memory to the processor and the final processed information is stored back again in the main memory. From recent research studies, it is calculated that the data migration is 100× greater than the ADD operation [26] [20]. Also, this system suffers from memory-wall issues due to limited memory channel bandwidth for data transmission. To mitigate this drawback, we are reusing the concept of processing in-memory/compute-in-memory concepts [36] [37]. In PIM the processing unit is made closer or within the main memory causing reduced data migration length.

Consequently, this modification in PIM with reference to the baseline memory system leads to the enhanced performance of the computing system in multifold [38] [39] [40]. Processing in memory (PIM) is not a new research area. It was started in early 2004 [20] by IBM research members, but during that time the fabrication issue and high cost hindered the research progress. However, nowadays with technology's progression the fabrication cost have been drastically reduced. It is stable to fabricate a computing unit/processing unit inside the main memory.

Motivation

Traditionally, the main memory in recent computing systems uses DRAM. However, due to the scaling and reliability issue in DRAM, high leakage current caused the hindrance in development at the sub-nanometer regime of 22nm [9]. Also, a conventional volatile memory system (i.e., DRAM) suffers from repetitive refresh requirements to retain the data because the stored memory data in DRAM is volatile. To overcome these challenges in recent times, researchers have conducted many experiments to replace DRAM with non-volatile emerging memory (NVM) such as Phase change memory (PCM/PCRAM), spintronic devices, and magnetic tunnel junction devices [41] [42] [43]. In addition to this NVM (especially memristive devices) have near zero leakage current and, it is easy to scale the size even fabricated to 7nm by IBM research [43]. Also, the memristive devices mimic the character of a neuron network thus reducing the hardware required to a very large extent. To give an example, using the CMOS logic the gate requirement to implement

a single neuron node of NN is very high (22 transistors). In contrast, we can replace this with a single memristive cell for a great hardware footprint reduction.

Background

Multiply-accumulate (MAC) is the most widely used arithmetic operation, which is both a data and compute-intensive operation. In this chapter of our dissertation, we are concentrating on the PIM architecture in recent times to perform the MAC operation and use of data-intensive applications such as deep learning, convolution neural network, and relevant works.

PIM are classified based on the data interpretation and stored in the memory array into two classes. They are PIM-A [44] and PIM-P 54 [45]. First, consider some of the following applications that use the resistance changes to compute and transmit data, and output is produced in the memristive array called PIM-A [46]. Also, these PIM-A architectures such as Snider [38], IMPLY, and FBL [47], use the resistive changes as their input and output of the memristive crossbar. Further, hybrid PIM-Ah is proposed, which consists of resistive accumulator [46], and majority logic [48]. But these architectures suffer from the reliability issue of DAC and ADC components.

Second, in some PIM operations, the output is produced in the periphery of the memory unit as voltage changes and this class of architecture is called PIM-P. Depending on the voltage representation in PIM-P, memory systems are broadly classified into resistive PIM-Pr and PIM-Cr [25] [49]. Some of the recent CIM-Pr work is Pinatubo [50] [51], scouting [15], and HieIM [52] where the output peripherals are very simple and less complex to design. Whereas in PIM-Cr the output peripheral circuits are complex to design [53] [54]. One key point to be remembered is that PIM-Cr uses ADC/DAC units at the output to perform the calculation which consumes almost 85% of area and 90% of total energy consumption.

Work related to MAC operation

To maintain brevity and consistency, we narrow down our documentation related to the PIM-based accelerator with only arithmetic and arithmetic with logical operation accelerator for MAC operation. Related work is tabulated in Table 2.2. Also, some of the recent works related to the arithmetic operation using PIM architecture, such as 3D-DRAM to perform convolution operation proposed by Wang [55]. However, this hardware implementation cost (i.e., complex fabrication steps) is expensive with a complex arithmetic processing unit (PU). Ahead HMC-MAC [27], NeuralHMC [56] [18] are 3-D DRAM-based accelerators for increasing the MAC operations, but the processing unit (PU) design is a complex and expensive fabrication process to implement.

To perform the massive MAC operations in parallel for a neural network using DRAM, literature such as McDRAM [57], where the processing unit (PU) of MAC is placed close to the column decoder and it suffers from the area overhead issues and timing violation.

Few recent articles use the LUT-based MAC operation such as LAcc [58] [59], where each MAC operation is disintegrated into smaller units that decompose the multiplication operation size and LUT size. However, the accuracy is very low in LAcc architecture.

Table 2.2: The literature survey on PIM for arithmetic and logic operation with different memory class

Technology	year	Name	Data precision	Large Data Mapping	In Memory Bitwise op.	In memory large Data Op.
DRAM-3D	2017	wang [64]	16b	Bit parallel	N/A	MAC
RERAM	2016	chi [65] / prime	6b	crossbar	N/A	MAC
MRAM/RERAM	2017	du [66]	8-bit	Crossbar	AND/OR/NOR	ADD/MUL/MAC
MCRAM AND PCM	2018	mimi [67]	BWNN/ 8-bit	Bit-serial	AND/XOR	MAC
THYRISTOR DEVICES	2018	joonseopsim [68]/lupis	6-bit	Bit-parallel	AND/NOR/SHIFT	ADD,MUL,MAC
RRAM	2018	halwani [69]	BCNN/Var Width	Bit-Parallel	AND/SHIFT/OR/NOT	MAC/SHIFT
CMOL	2018	madhav [70]	8-bit	Bit-serial	AND/NOR/SHIFT	MAC
RERAM	2018	haiyu [71] / lergan	16-bit	Crossbar	AND/OR	MAC
RERAM	2019	gupta [61] / nnpim	16-bit	Crossbar	AND/OR	MAC
RERAM	2019	RAPID [72]	16-bit	Crossbar	AND/OR	MAC
RERAM	2019	angizi [21] [73]	10-bit	Crossbar	AND/OR	COMPARISION
STT-MRAM	2019	hao [41] /pj-axmt	8-bit	Crossbar	AND/OR	MAC
DRAM	2020	roy [74]	8-bit	Bit-parallel	AND/OR	MAC
MCRAM/ PCM	2018	xie [75]/ aim	BCNN	Crossbar	XNOR	N/A
MEMRISTIVE CROSS BAR	2016	shafiee [10] / isaac	16-bit	Crossbar	N/A	MAC
RRAM	2018	chen [28]	DCNN	Crossbar	N/A	MAC
MEMRISTIVE CROSS BAR	2017	ankit [26]/ resparc	Var. Width	Crossbar	AND/OR/NOT	MAC
SOT-MRAM	2017	angizi [30] / imc	8-bit	Bit-Parallel	AND/OR	N/A
SOT-MRAM	2018	angizi [31]/ cmp-pim	8-bit	Bit-Parallel	AND/OR	COMPARISON
NVM	2018	nihar [76]	Var. Width	Crossbar	AND/OR/NOT	MAC
HMC	2019	chuhan [18]/ neuralhmc	16-bit	Bit-Parallel	AND/OR/NOT	MAC
SRAM	2019	gao [77] / tangram	16-bit	Bit-Parallel	AND/OR/NOT	MAC
HMC DRAM	2018	jeon [27]/ hmc-mac	Var. Width	Bit-Parallel	AND/OR/NOT	MAC
NANOWIRE WITH NVM	2017	liu	Var. Width	Bit-Parallel	AND/OR/NOT	ADD, MUL
RERAM	2018	long [78]	10-bit	Crossbar	AND/OR	MAC
DRAM	2018	shin [79] / mcdram	6-bit	Bit-Parallel	AND/OR/NOT	ADD, MUL
DRAM	2019	wang [60]	BCNN	Bit-Parallel	AND/OR/NOT	MAC
DRAM	2019	deng / LAcc	8-bit	Crossbar	XNOR	N/A
MEMRISTOR	2017	john [10]	8-bit	Crossbar	NOR	NOR/SHIFT
SOT-MRAM	2017	zhezhi [44]	8-bit	Bit-Parallel	AND/OR	MAC
WIDE IO2 DRAM	2017	lei [80] / xnor-pop	BCNN	Crossbar	XNOR	N/A
SOT-MRAM	2018	angizi [81] /imce	8-bit	Bit-Parallel	AND/OR	MAC
STT-MRAM	2018	jain [82]	8-bit	Crossbar	NOR	NOR/SHIFT
NANOWIRE SPINTRONICS	2018	angizi [81]	BCNN	Crossbar	AND/NOR	N/A
RERAM	2018	ammer [16]/imaging	8-bit	Crossbar	NOR	NOR/SHIFT
NVM	2018	said [83] [84]	16-bit	Crossbar	AND/NOR/NOT/SHIFT	MAC/COMPARISION
DRAM	2018	li [85] [86]/ scope	8-bit	Bit-Parallel	AND/OR	ADD/MULT(STOCH)
SOT-MRAM	2018	angizi [87] [88]/ dima	8-bit	Bit-Parallel	AND/OR	MAC/COMPARISION

Further, ReRAM-based crossbar memory-based PIM architecture for high performance and massive parallelism in PRIME [60], NNPIM [61] but it suffers from very high computation delay time for multiple sense amplifier (SA) operations. Further, in some of the recent work (i.e., ISAAC [10], DaDiano [62] [63]) MAC operation will be processed in a dedicated subsidiary crossbar-based processing chip. Even though the operating speed is high the hardware cost of fabrication and energy dissipation due to the dedicated crossbar-based memristive circuit is almost 85%. Next, there exists ReRAM-based crossbar memory-based PIM architecture for high performance and massive parallelism, e.g., PRIME [60], NNPIM [61]. Still, it suffers from very high computation delay time for multiple sense amplifier operations. Further MAC operation based on dedicated chip in ISAAC [11], DaDiano [63]. These systems are called near-data computing. Even though the operating speed is high, the hardware cost of fabrication and energy dissipation due to the dedicated crossbar-based memristive circuit is almost 85%. Also, the dedicated chip will not incorporate the enormous data set of the current big CNN applications.

Researchers have tried to use the approximate computation technique to reduce the

hardware footprint with tradeoffs in accuracy, some of these works are Pj-AxMTJ [89]. LAcc [90]. Further, XNOR-POP [91] is used to reduce the arithmetic operation in the convolution neural network to implement BNN. This results in a reduction of complex arithmetic operations into simple logic operations. In this article, the author used the wide-IO DRAM to show the BNN implementation. Also, recent work by DIMA [87] converted all the computationally expensive components into simple logic circuits with better approximation. They have shown the feasibility by mapping the AddNet CNN application to SOT-MRAM-based memory architecture. Likewise, modification in sense amplifier performs the logical operation as shown in Pinatubo [92] to perform MAC operation used in CNN applications such as CiM-SCAM using FeFET and SRAM technology. SCOPE [93] uses SC to perform bulk bit-wise logic operations (Limited logic function), but the study is limited only to multiplication. This document can be used as a manual for future scope and improvement needed for PIM architecture with the class of device technology.

Chapter 3 Scalable Stochastic Number Generator for Phase Change Memory (PCM) based In-memory Stochastic Processing

3.1 Chapter Overview

In this chapter, a novel approach to designing a Stochastic Number Generator (SNG) was proposed, which leverages the inherent stochastic nature of Phase Change Memory (PCM). The PCM-based SNG was implemented and tested for different binary input lengths, and its performance was compared to the state-of-the-art Linear Feedback Shift Register (LFSR)-based SNG. The proposed SNG was found to be highly scalable with a minimal hardware footprint, while also overcoming the challenges and constraints associated with the workflow of PCM-based SNG.

Overall, the proposed SNG was shown to offer significant improvements over conventional LFSR-based SNGs in terms of energy and area savings, with up to 250× and 300× improvements, respectively, for a 14-bit binary input length. These results demonstrate the potential of PCM-based SNGs as a promising solution for stochastic arithmetic-based CNN accelerators.

3.2 Background: Phase change memory

A PCM cell embeds a small volume of chalcogenide material $Ge_2Sb_2Te_5$ (GST) [94], which can be programmed into two different states (i.e., crystalline/SET state and amorphous/RESET state) with dramatically different electrical resistance [17]. The amorphous (RESET) state represents a binary "0", while the crystalline (SET) state represents a "1". The resistance of PCM cells in the SET state (RSET) follows a normal distribution in the $k\Omega$ range [30], whereas the resistance of PCM cells in the RESET state (RRESET) follows a normal distribution in the $M\Omega$ range [9], as shown in Figure 3.1-(B).

Typically, for a PCM array, I_{READ} and V_{REF} are judiciously designed such that V_{REF}/I_{READ} falls in between the resistance distributions of SET and RESET cells (Figure 3.1-(B)). As a result, for a PCM row being read, V_{SET} for all SET cells is less than V_{REF}, whereas V_{RESET} for all RESET cells is greater than V_{REF}, enabling the distinction of the SET cells (logic '1's) from the RESET cells (logic '0's) with 100% probability.

Now consider that a SET PCM row with all its cells pre-programmed in the SET state (storing '1's) is being read. For this read operation, if we can control V_{REF} such that V_{REF}/IREAD falls somewhere on the resistance distribution of the SET cells, we can control the number of cells that would be read as '1's. For example, if we can control V_{REF} to be V_{REF}', as shown in Figure 3.1-(B), V'_{REF}/I_{READ} would fall at the center of the resistance distribution for SET cells. As a result, only 50% of the cells in the SET PCM row would be read as '1's (as these cells fall on the left of the V_{REF}'/I_{READ} reference), and the remaining 50% cells would be read as '0's. Learning from this observation, we leverage a judicious control of the PCM read operation to design an efficient SNG, as described next.

3.3 Stochastic Number Generator

Figure 3.1 presents a functional block diagram of our proposed Stochastic number generator (SNG). In the field of Stochastic computing, we need to have, complete control on a total number of 1's and 0's and which is controlled by the voltage reference value (V_{REF}). Apart, from this Stochastic computing, we don't need a true random number generator (RNG) with uniform distribution. We only need to have an RNG that we can control the number of generated 1's in its output. To convert an N-bit input number (Figure 3.1-1) into a 2N-bit Stochastic bit-vector (Figure 3.1-4), our SNG employs a PCM-based Gaussian digital-to-analog converter (GDAC) (Figure 3.1-2) that provides V_{REF} for reading a row of 2N PCM cells (Figure 3.1-3) pre-programmed in the SET state. Figure 3.1-2 illustrates the GDAC operation for a 3-bit binary input (B2B1B0), which can be generalized to any N-bit binary input (BN-1..B1B0). In general, an N-bit GDAC contains N switches (S0 to SN-1), each of which corresponds to a bit (B0 to BN-1) in the N-bit input number and depending on the bit's value ('0' or '1'), connects its respective voltage source (V0 to VN-1 – implemented using charge pumps [93]) to the voltage summer circuit. We assume that the cumulative distribution function (CDF) for the resistance of PCM SET cells (Figure 3.1-3) is available at the design time. Therefore, the voltage level of the voltage source, corresponding to a bit BX from the N-bit input, can be determined at the design time as: VX = I_{READ} × CDF-1(2X/2N), where CDF-1 is the inverse CDF that gives the resistance value RX (Figure 3.1-3 and 3.1-4) for the given probability number. Accordingly, our proposed N-bit GDAC produces V_{REF} using the embedded voltage summer circuit (Figure 3.1-2). When this V_{REF} is used to read the PCM row with 2N SET cells (Figure 3.1), only the number of cells out of total 2N cells are read as '1's, and the remaining cells are read as '0's, thereby converting an N-bit input into a 2N-bit Stochastic output.

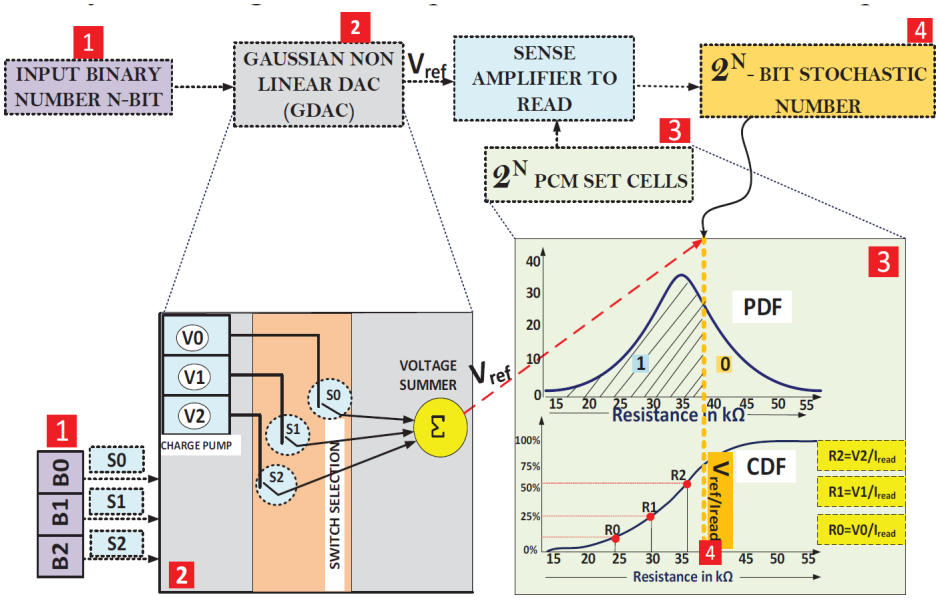

Figure 3.1: Functional block-diagram of our proposed Phase Change Memory (PCM) based Stochastic Number Generator (SNG).

For example, let us understand how this works for a 3-bit input number 101 (i.e., B2=1, B1=0, B0=1). To convert this number into a Stochastic number, our SNG should be able to produce a vector of total 8-bits with five '1's and three '0's. In our SNG (Figure 3.2), the voltage level V2 of the voltage source, corresponding to B2=1, can be evaluated as V2 = IREAD× CDF-122/23 = IREAD× CDF-14/8. Similarly, V1 and V0 can also be evaluated. Consequently, our proposed GDAC produces V_{REF} = B0V0+B1V1+B2V2 = V0+V1, as B1 is zero for our example number 101. This V_{REF} is used to read a total of $2^3 = 8$ PCM SET cells and doing so results in only $B_0 2_0 + B_1 2_1 + B_2 2_2 = 2^0 + 2^2 = 5$ cells to be read as '1's, and the remaining 3 cells to be read as '0's, hence, correctly converting the binary input 101 into an 8-bit Stochastic number.

3.4 Results

We evaluated the area and energy consumption of our proposed SNG and compared the results with the LFSR-based conventional SNG from [25], for different bit-sizes of the input binary number from 4-bits to 14-bits. We used Cadence's Spectre for SPICE-level simulations of the LFSR-based conventional SNG and our GDAC-based SNG with the op-amp-based voltage summer. We take the area, energy, and delay values for PCM from [30] and for charge-pump based voltage-sources from [95]. The $\mu and \sigma$ for the SET resistance distribution for PCM cells are evaluated from [96] to be 34.15kΩ and $6.54k\omega$, respectively. Figure 3.1 and Figure 3.2, respectively, show the area consumption and energy values, for the two SNG designs, evaluated for 14nm technology by scaling the SPICE-based results for 45nm technology. From the figures, as the bit-size of the input binary increases from 4-bits to 14-bits, the energy and area values for our GDAC-based SNG hardly increase, compared to the exponential increase in the energy and area values for the conventional LFSR-based SNG. As a result, for a 14-bit input number, our GDAC-based SNG consumes 300× less area and 250× less energy, compared to the LFSR-based SNG.

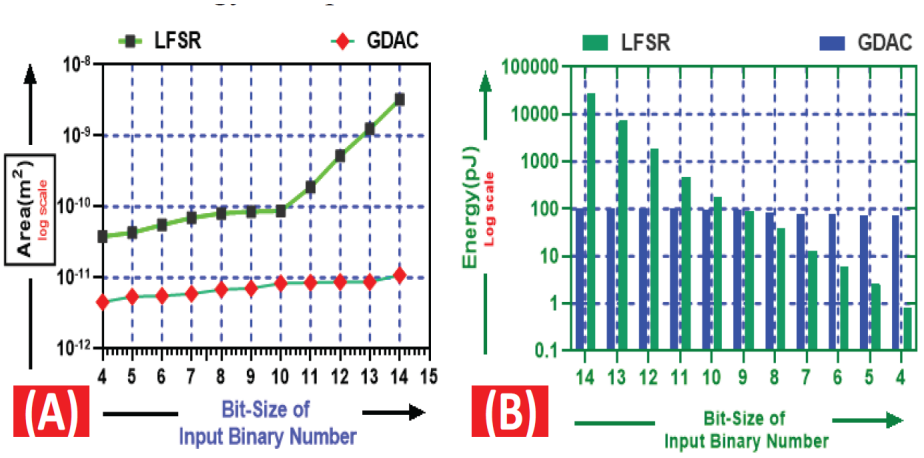

Figure 3.2: (A) Area and (B) energy consumption values of our GDACbased SNG and conventional LFSR-based SNG for various bit-sizes of input binary number.

3.5 Summary

In summary, the use of our PCM-based SNG for Stochastic PIM architectures provides the following benefits: (i) Our SNG can convert any N-bit binary number into a 2Y–bit Stochastic number, by simply using the V_{REF} generated by our GDAC to read total 2Y PCM SET cells. Here, Y can be any number $\geq$ N, as an abundant number of PCM cells would already be available in PCM based PIM architectures, which provides an unprecedented opportunity to improve the precision of the generated Stochastic numbers without adding any significant conversion logic/circuit and related overheads in our designed SNG. (ii) As PCM SET cells are already available in a PCM-based PIM architecture, GDAC is the only block of our SNG that incurs area overhead. Moreover, unlike the LFSR-based SNGs from [25], our GDAC-based SNG can generate all 2Y bits of the output Stochastic number in parallel, which results in huge latency and energy savings for our GDAC-based SNG. In the future, we plan to evaluate the conversion-based and precision-based error efficiencies for our proposed SNG. Moreover, we intend to do a comprehensive comparative analysis of our proposed SNG with other SNG designs from prior work. Further, we will do detailed case studies to explore the utilization of our designed SNG for various Stochastic Processing-In-Memory (PIM) applications, including deep neural network inference and training applications.

Chapter 4 An Accelerator Based on Parallel SC for In-PCRAM Deep Learning Applications

4.1 Chapter Overview

This chapter proposes a new SC-based Processing-In Memory (PIM) engine called ODIN. ODIN is evaluated by mapping four CNN benchmark applications on it and is shown to be significantly faster and more energy-efficient than conventional designs and prior crossbar-based in-situ CNN accelerators. The proposed solution offers a promising approach to address the challenges associated with processing CNNs, and the ODIN accelerator has the potential to significantly improve the energy and time efficiency of CNN-based applications.

4.2 Introduction

CNNs have achieved remarkable progress in recent years, and they are being aggressively utilized in real-world applications related to artificial intelligence (AI) and machine learning [47]. In general, CNNs mimic biological neural networks and utilize compute-heavy arithmetic functions such as multiply-accumulate (MAC), nonlinear activation, and pooling. Although these CNN functions are amenable to acceleration because of a high degree of compute parallelism, their acceleration is challenging because of the need to avoid the memory wall while accessing their large number of operands [9]. To address this problem, several prior works have explored crossbar memory-based processing-in-memory (PIM) accelerators (e.g., [9] [24]) that leverage the Kirchhoff's Law to perform MAC operations in the analog domain. However, such analog computing-based accelerators require power-hungry and sluggish digital-to-analog converters and analog-to-digital converters (DACs and ADCs), which diminishes the performance and energy-efficiency benefits of such accelerators. Moreover, these accelerators do not fully capitalize on the PIM paradigm, as they still have to heavily rely on conventional processor-centric computing for implementing essential CNN functions such as nonlinear activation and pooling. This motivates the need for simple, low-overhead, and energy-efficient in-situ accelerators that can realize the full potential of PIM-based CNN processing. In this chapter, we present a phase change RAM (PCRAM) based in-memory CNN accelerator called ODIN. ODIN uses Stochastic arithmetic to convert complex MAC operations into a series of simple and low-overhead in-situ logical operations that are implemented using the analog computing capabilities of PCRAM [94]. Stochastic arithmetic typically requires additional circuits to enable conversion of the operands between the Stochastic and binary number formats [23]. ODIN employs lightweight CMOS add-on logic inside PCRAM banks to implement such number conversion circuits with minimal area and power consumption overheads. ODIN also employs custom CMOS logic blocks to realize the binary arithmetic-based implementation of nonlinear activation and pooling functions of CNNs. Stochastic arithmetic for CNN processing has been utilized before in the DRAM-based in-situ accelerator described in [23]. The accelerator from [23] employs heavy add-on logic inside DRAM banks, which can

increase the total area of DRAM chips by up to 4× [19]. In contrast, ODIN presents the first-of-a-kind Stochastic arithmetic-based accelerator that includes extremely low-overhead add-on logic and very lightweight modifications in PCRAM banks and PCRAM controller for efficient processing of CNNs. Our key contributions to this work are summarized below.

- We present a novel processing-in-memory (PIM) accelerator called ODIN that leverages hybrid binary-Stochastic arithmetic to accelerate CNN processing in PCRAM;
- ODIN employs low-overhead add-on logic and lightweight PCRAM modifications to implement Stochastic arithmetic based MAC operations and binary arithmetic-based activation and pooling operations directly inside PCRAM banks;
- We evaluate our ODIN accelerator for four CNN benchmark topologies (i.e., CNN1, CNN2, VGG1, and VGG2) from MLBench library [2], with two datasets MNIST and ImageNet;
- We compare the performance of our ODIN accelerator for the considered CNN benchmarks with the following architectures: baseline CPU-only with 32-bit floating precision, CPU-only with 8-bit fixed precision, as well as the pipelined and unpipelined variants of the ISAAC accelerator from [97].

4.3 Related work and motivation

Several processing-in-memory (PIM) based CNN accelerators are proposed in prior work (e.g., [97] [23] [10]- [2]). Some of these accelerators use memory in the conventional volatile RAM configuration (e.g., SRAM [14], DRAM [23] [10]- [13] based CNN accelerators), whereas some others use the emerging non-volatile memory in the crossbar configuration (e.g., STT-MRAM [17] [18], ReRAM [97] [15] [16] [2]). The RAM-configured CNN accelerators use bit-line computation to perform basic bit-parallel logical operations in a single read of a traditional RAM. Each memory column (bit-line) is further transformed into a bit-serial ALU by adding extra logic, multiplexing, and state elements in the peripheral circuitry, to perform complex arithmetic operations such as multiplication and addition, which are very essential for CNN processing [24] [1]. However, these accelerators incur high area-power overheads in their augmented peripherals and suffer from low throughput of bit-serial execution. Along the same line, a recently proposed DRAM-based in-situ accelerator [23] uses Stochastic arithmetic to perform CNN operations. However, this accelerator heavily modifies peripherals to facilitate logic circuitry, registers, and shifters to enable carry-save addition required for MAC processing. These modifications can result in up to 4× area increase per DRAM bank [19]. On the other hand, NVM crossbar-based accelerators also suffer from high overheads of ADC and DAC circuits. In contrast, our proposed design ODIN stands first of its kind to leverage the hybrid binary-Stochastic arithmetic inside PCRAM banks for implementing a very low-overhead acceleration of CNNs.

To explain the structure and operations involved in a CNN, we explain a highly popular CNN model called Convolutional Neural Network (CNN). Generally, CNNs consist of majorly four types of layers (Figure 16), namely convolution (CONV), local response

normalization (LRN), pooling (POOL), and fully connected (FC) layers [1]. Among all these layers, the major portion of information of the input data gathering is taken place in the CONV and FC layers are computationally heavy, wherein a very large number of multiply-accumulate (MAC) operations occur between the synaptic weights and input activation values. In addition to MAC operations, other important operations in a CNN (ANN) are nonlinear activation and pooling. These CNN operations typically involve many steps for intermediate value preparation and storage, and therefore, they can be highly time and energy-consuming. For better insight, consider Eq. (4), related to an FC layer, which consists of MAC operations between the weights ($w_{i,j's}$) and input activation values (ai's). In Eq. (4), $w_{i,j's}$ the synaptic weight connecting i^{th} node in layer 'a' to j^{th} node in layer 'b', and 'f( )' is a non-linear activation function (e.g., ReLU [1]).

Moreover, popular pooling functions for CNNs are maxed and average pooling functions. Also, note that, since ODIN uses Stochastic arithmetic, the results are always between '0' and '1'. Therefore, for processing CNNs on ODIN accelerator, the LRN layer can be safely ignored with minimal loss of accuracy [97].

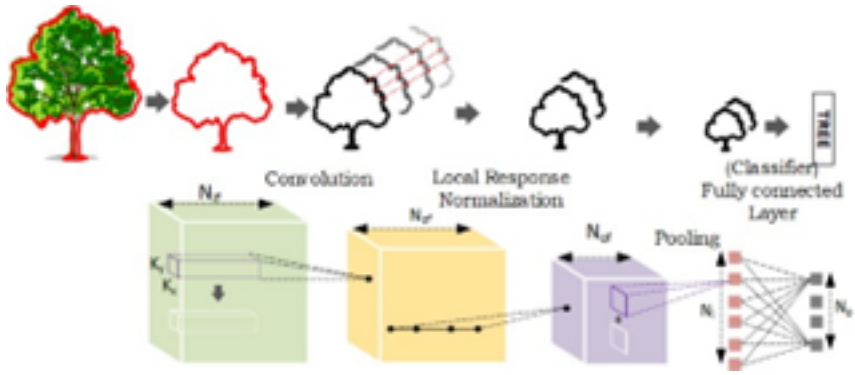

Figure 4.1: Four major types of layers found in Convolutional Neural Networks (CNNs) [1].

4.4 Phase change Ram (PCRAM) architecture

This Section provides a brief background on PCRAM organization required to understand ODIN. The operation of PCRAM is not discussed here, for which the reader is encouraged to refer to prior work [98] and [20]. PCRAM, like DRAM, is organized hierarchically [20]. An example PCRAM memory of 16GB capacity has 2 channels, with 8 ranks per channel, and 16 banks per rank. A PCRAM bank has 16 partitions, each of which is an array of 4096 wordlines and 8k bitlines. A PCRAM bank has 256 peripheral structures, which include the sense amplifiers (S/As) to read and the write drivers (W/Ds) to write. The peripheral structures in PCRAM banks allow us to read and program 256 PCRAM cells in parallel. Therefore, the read and write granularity is 256 bits (the size of a memory line).

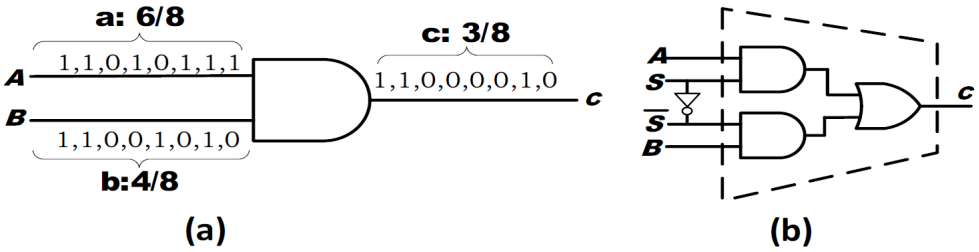

Figure 4.2: Stochastic arithmetic circuits. (a) Multiplier (AND gate), (b) Adder (MUX circuit).

4.5 Stochastic arithmetic

In Stochastic arithmetic, data is processed in the Stochastic number (SN) format instead of the conventional binary number (BN) format. In SN format, information data is represented as pseudorandom bit streams. Using Stochastic arithmetic, the complexity of performing MAC operations can be reduced substantially. For example, multiplication can be implemented by a simple bit-parallel AND operation, and scaled addition (accumulate) can be realized by a bit-parallel multiplexing (MUX) operation [21]. Figure 17(a) depicts an AND gate implementing c = a×b in the SN format and Figure 4.2(b) shows a MUX implementing $c = s \times a + s' \times b$ in the SN format. In the ODIN engine, we always use s=0.5 for MUX-based Stochastic addition.

4.6 ODIN Framework: overview

The design of ODIN accelerator includes the following three hardware architecture attributes: (i) integration with heterogeneous computing system, (ii) lightweight modification of peripherals and inclusion of add-on logic in PCRAM banks to enable in-situ processing of CNNs, and (iii) introduction of new memory controller commands to support the processing-in-memory (PIM) functionality in PCRAM. Each of these attributes is described below.

4.7 Integration with heterogeneous computing system

Figure 4.3 shows the heterogeneous computing system architecture that employs our ODIN accelerator. The system consists of a heterogeneous processing chip (HPC) that has heterogeneous cores (e.g., CPU, GPU cores) integrated with on-chip memory. The HPC connects, through a bus, with a phase change RAM (PCRAM) based main memory subsystem and a storage subsystem (SSD based). The PCRAM main memory subsystem has two channels. One of the PCRAM channels functions as the main memory channel by default. In contrast, the PCRAM modules on the other channel are lightly modified with add-on logic to constitute our ODIN accelerator, which has PIM capabilities for an efficient in-situ acceleration of CNNs. These modified PCRAM modules, referred to as ODIN accelerator modules, also serve as PCRAM-based main memory modules when their PIM

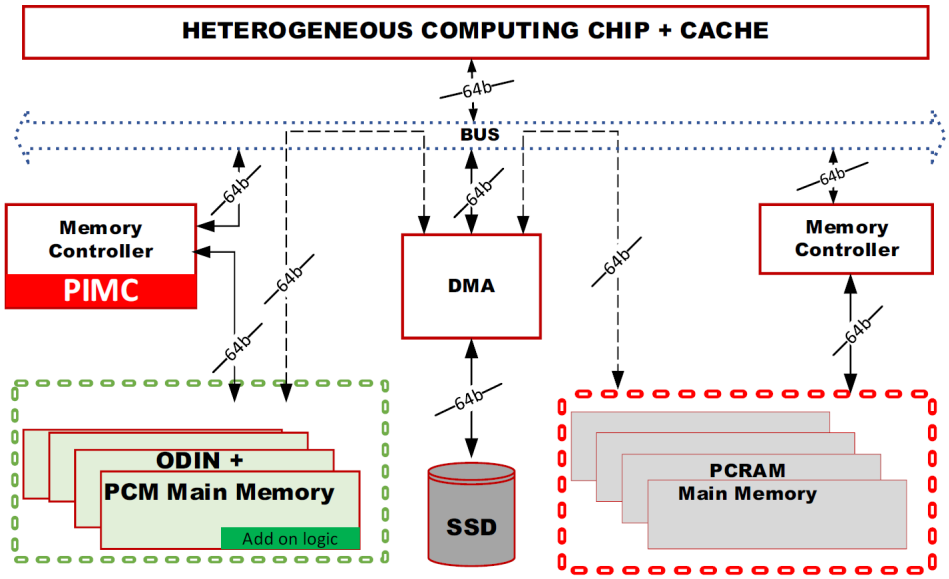

Figure 4.3: Integration of ODIN in a heterogeneous system.

functionality is not in use. Both the ODIN accelerator and PCRAM main memory channels are connected to the storage channel through direct memory access (DMA) controllers, in addition to being connected to conventional memory controllers that manage regular memory operations. It is assumed that the DMA controller, in association with the software stack, can prefetch all the operands related to a CNN under execution from the SSD channel and load them in the ODIN accelerator modules, to subsequently trigger their processing directly inside the ODIN accelerator modules. The regular memory controller associated with the ODIN accelerator channel is also lightly modified with PIM controller (PIMC) capabilities, to support additional commands for efficient management of ODIN's operation.

4.8 Hardware modifications in PCRAM banks

Our ODIN accelerator can fully process CNNs directly inside PCRAM. For that, ODIN employs lightweight modifications in PCRAM banks that enable in-situ execution of essential arithmetic functions of CNNs, such as multiply-accumulate (MAC), activation, and pooling. To reduce the overheads of affected modifications, ODIN employs hybrid binary-Stochastic arithmetic for implementing these functions. To this effect, ODIN implements MAC operations using Stochastic arithmetic, whereas for activation and pooling functions, ODIN uses traditional binary arithmetic. The specific modifications in the PCRAM bank structure, effected to implement various CNN functions, are described in the SubSections below. In addition, ODIN also dedicates an entire partition per PCRAM bank as a scrap memory space that is utilized to help perform various CNN functions in situ. This partition is identified as Compute Partition in Figure 19(a), which shows a schematic of ODIN's PCRAM bank with effected modifications.

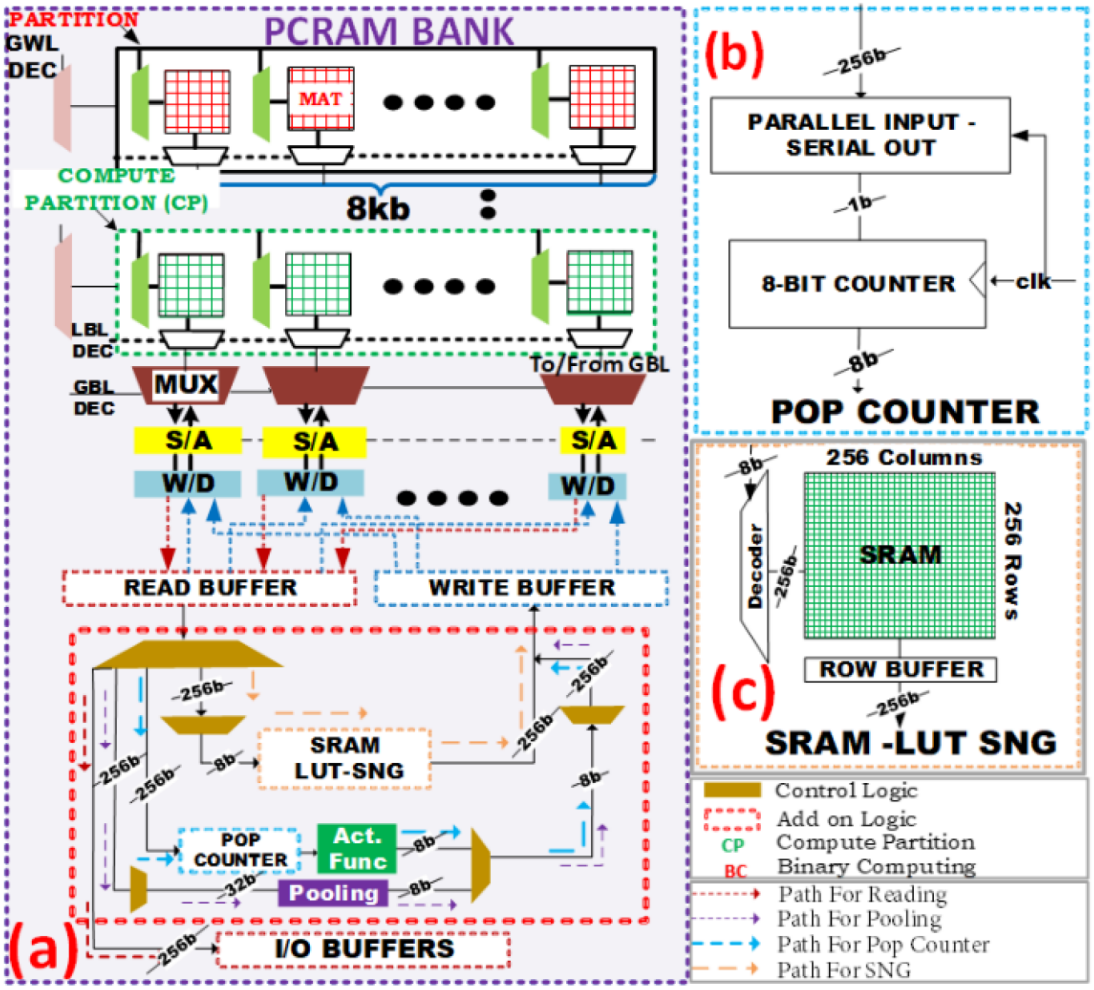

Figure 4.4: Schematic of a PCRAM bank modifications in ODIN.

4.9 Hardware for multiply-accumulate (MAC) operations

To implement in-situ MAC operations inside PCRAM, ODIN uses Stochastic arithmetic, which transforms complex multiplication and addition operations, respectively, into simple bit-parallel AND and multiplexing operations. Multiplexing operations can be further divided into bit-parallel AND and OR operations (Figure 4.2(b)) using Stochastic arithmetic. Thus, ODIN employs Stochastic arithmetic to convert complex MAC operations into a series of simple bit-parallel AND and OR operations. However, for MAC operations to take place inside PCRAM in the Stochastic format, all MAC operands need to be present in PCRAM in the Stochastic format. Storing all MAC operands of a CNN inside PCRAM ahead of time in the Stochastic format may require impractically high storage capacity. This is because an N-bit binary operand typically occupies 2N- bits when it is converted into the Stochastic format with reasonable precision. Therefore, to reduce the storage footprint of CNN MAC operands, ODIN makes two adjustments. First, it makes CNN MAC operands available in PCRAM banks in binary format. Second, it fixes the size of the operand to 8-bits - this adjustment resonates with other prior works on PIM-based CNN accelerators (e.g., [1], [2])

wherein the size of CNN operands is fixed without significant loss of CNN inference and training accuracy. In the wake of these adjustments, to perform the CNN MAC operations in the Stochastic format, ODIN employs the following hardware modifications:

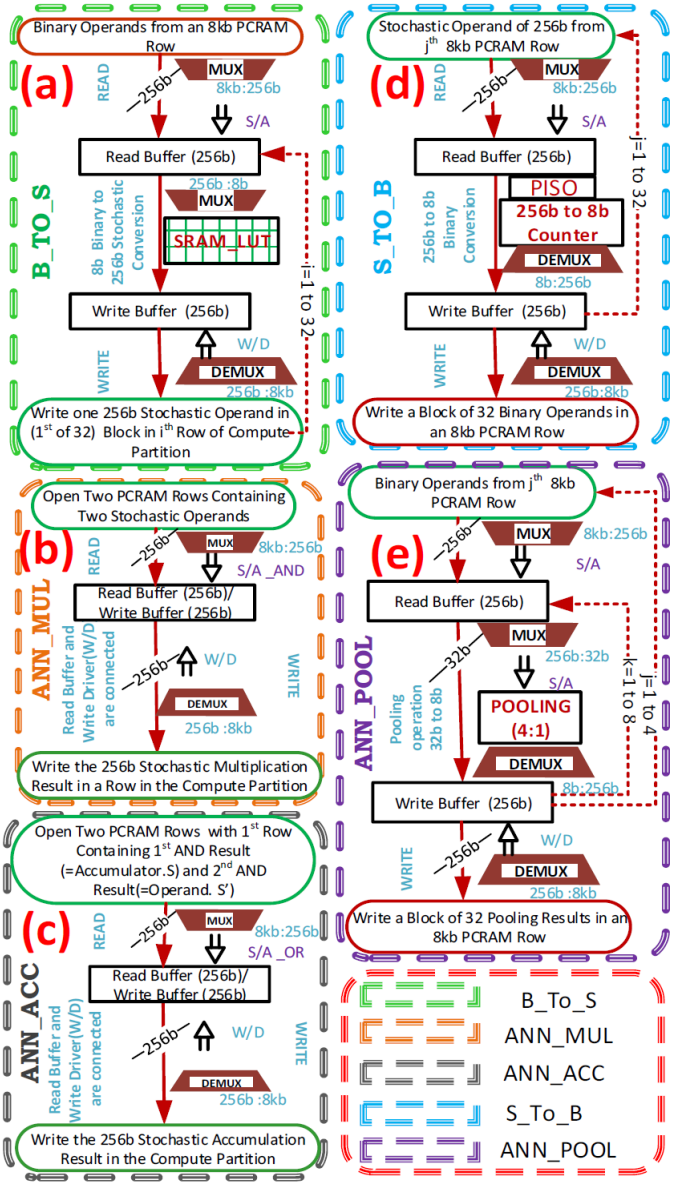

Figure 4.5: PCRAM activity flows for ODIN PIMC commands (a)B_TO_S, (b) CNN_MUL, (c) CNN_MUL, (d) S_TO_B, and (e)CNN_POOL.

Lightweight add-on logic circuits

Lightweight add-on logic circuits are included in every PCRAM bank to enable the conversion of operands between the binary and Stochastic formats. The add-on logic circuits include an SRAM based lookup table for binary to Stochastic conversion, and a pop counter for Stochastic to binary conversion, of CNN MAC operands. Figure 4.4(b)-

4.4(c) shows these add-on logic circuits as part of the schematic of a PCRAM bank. The lookup table consists of an SRAM block of 256x256 cells (Figure 4.4(c)), with peripherals to access the block. An 8-bit operand in the binary format is used to index into the SRAM block of 256 rows via a decoder. The indexed SRAM row provides the Stochastic version of the operand, which is read in the SRAM block's local row buffer. Along the same lines, the pop counter logic includes a clocked 256-bit parallel-in-serial-out (PISO) register connected with an 8-bit digital level counter (Figure 4.4(b)). The PISO stores a 256-bit Stochastic operand and then serially outputs it bit-by-bit, which is fed in the counter to count the number of 1s in the Stochastic operand, to consequently convert the operand in the binary format. From Figure 4.4(a), ODIN also employs lightweight control logic to facilitate movement and processing of the operands through these add-on logic circuits. The overheads of these add-on logic circuit, and their control logic are discussed in Section 5.8 (Table 6).

A low-overhead modification in PCRAM bank peripherals

A low-overhead modification in PCRAM bank peripherals is included to enable in-situ MAC operations (i.e., series of bit-parallel AND and OR operations) among the operands available in PCRAM in the Stochastic format. These modifications are made in the sense-amplifiers, row-address decoders, and wordline drivers, as adopted from PINATUBO framework [99]. Like PINATUBO framework [50], these modifications also enable ODIN to perform bit-parallel logical AND/OR/NOT operations between two operands, by simply storing the two operands in two separate PCRAM rows and then activating and reading both the PCRAM rows simultaneously using appropriate sense-amplifier reference voltages. We extract the overheads of these modifications from [100], and account for them in our system-level analysis in Section 4.10.

4.10 Hardware for activation and pooling functions

To implement binary arithmetic-based CNN activation and pooling functions, ODIN employs two different add-on logic blocks per physical PCRAM bank (Figure 4.6(a)). The logic block for activation function consists of a CMOS implementation of an 8-bit ReLU [27] and it directly follows the pop counter circuit (Figure 4.4(a)). On the other hand, the logic block for pooling function implements CMOS circuit for 4:1 8-bit max pooling [25]. The additional control logic included in the PCRAM bank facilitates the movement and processing of operands through these activation and pooling logic blocks. The overheads of activation and pooling logic blocks are given in (Section 5.5) and are accounted for in for our calculations. our system level analysis (Section VI). We choose 8-bit ReLU and 4:1 max pooling functions for ODIN in this chapter, because the CNN benchmark models (Section VI) we have considered for our analysis use these functions. Nevertheless, we envision that ODIN can be easily extended to use any other activation (e.g., tanh [26], softmax [27]) and pooling (e.g., average pooling [27]) functions with any other precision as well. C. New PIM Controller (PIMC) to Support CNN Processing To orchestrate the processing of CNNs by employing the modified hardware discussed in the previous subsection, ODIN introduces the following five new PCRAM controller com-

mands: (1) B_TO_S, (2) CNN_MUL, (3) CNN_ACC, (4) S_TO_B, and (5) CNN_POOL. Each of these commands con-sists of multiple basic PCRAM READ and WRITE operations (Table 1). Fig. 5(a) -5(e) shows PCRAM activity flows under-taken during these commands. Moreover, the latency/timing parameters associated with these commands are given in Table 1. The PIM controller (PIMC) of ODIN (Fig. 3) breaks down these PCRAM activity flows into a series of PCRAM operation commands (e.g., READ, WRITE) and control signals. Then, the PCRAM controller schedules these commands in appropriate order while abiding by various timing constraints (e.g., timing constraints of the activity flows in Fig. 5), to orchestrate efficient CNN processing, as discussed below.

(1) B_TO_S: this command orchestrates conversion of CNN operands from the binary format to the stochastic format. The activity flow of this command involves reading operands from PCRAM in the binary format, converting them into the stochastic format, and then writing them back into the PCRAM rows of the Compute Partition (Fig. 4.5(a)). As each 256-bit PCRAM block read from an 8kb PCRAM row can fetch 32 8-bit operands, the B_TO_S activity flow includes conversion of 32 8-bit binary operands into 32 stochastic operands (256-bit each), and writing them back in 32 separate PCRAM rows of the Compute Partition. (2) CNN_MUL: this command directs a multiplication operation (i.e., bit-parallel AND) between two CNN operands that are stored in the stochastic format (256-bit each) in two separate PCRAM rows of the Compute Partition. As discussed earlier, ODIN adopts PINATUBO framework [3] for implementing bit-parallel AND operations. As a result, the CNN_MUL activity flow involves simultaneously activating the two PCRAM rows that contain the two 256-bit stochastic operands in the Compute Partition, and then reading both operands simultaneously using an appropriate reference voltage in the sense amplifiers that facilitate reading in the PCRAM bank, as doing so performs bit-parallel AND between the two 256-bit stochastic operands [94]. The reader is encouraged to review [3] for more details on how such bit-parallel AND is performed in the PCRAM sense-amplifiers. (3) CNN_ACC: this command directs an accumulation (addition) operation (i.e., a multiplexing operation converted into a series of bit-parallel AND and OR, Fig. 4.5(c)) between a stochastic operand stored in a PCRAM row of the Compute Partition and an accumulator block that is part of the Accumulator Row in the Compute Partition. From Fig. 2(b), in addition to the stochastic operands, a multiplexing operation also requires S and S' operands. ODIN preprocesses S and S' operands offline and makes them available in two separate PCRAM rows of the Compute Partition. The CNN_ACC activity flow (Fig. 4.5(c)) uses these S and S' operands to perform two bit-parallel AND operations and one following bit-parallel OR operation, to consequently perform a multiplexing (accumulate or addition) operation. Like CNN_MUL, CNN_ACC also adopts PINATUBO to undertake its activity flow

We choose 8-bit ReLU and 4:1 max pooling function for ODIN in this chapter, because the CNN benchmark models (Section 5.6) we have considered for our analysis use these functions. Nevertheless, we envision that ODIN can be easily extended to use any other activation (e.g., tanh [76], softmax [77]) and pooling (e.g., average pooling [25]) functions with any other precision as well.

(4) S_TO_B: After the results of all MAC operations related to at least 32 neurons of an CNN layer are available in the stochastic format, ODIN invokes this command to convert the results into the binary format and then apply the activation function on them. The S_TO_B

Table 4.1: : Required number of PCRAM reads, writes, and resultant total latency values for various ODIN PIMC commands.

ODIN PIMC Command	#Reads	#Writes	Latency(ns)
B_TO_S	33	32	3504
S_TO_B	32	32	3456
CNN_POOL	32	32	3456
CNN_MUL	1	1	108
CNN_ACC	1	1	108

Table 4.2: Requirement of memory capacity, number of PCRAM reads and writes for implementing various CNN topologies on ODIN accel-erator.

	Fully Connected Layers			**Convolution Layers**			**Accuracy**
	Memory (Gb)	**Read (x106)**	**Write (x106)**	**Memory (Gb)**	**Read (x106)**	**Write (x106)**	**(%)**
VGG1	1.93	247	248	0.229	58.8	30.3	89.5
VGG2	1.96	251	252	0.234	60.01	30.9	88.51
CNN1	0.00095	1.22	1.226	0.0002	0.62	0.32	97.45
CNN2	0.00098	1.254	1.257	0.00026	0.67	0.34	97.21

activity flow (Fig. 4.5(d)) involves reading 32 stochastic MAC results from 32 different PCRAM rows of the Compute Partition, converting them one after another into the binary format using the pop count circuit block, applying the activation function on them one after another using the CMOS logic block for ReLU, and then assembling 32 resultant 8-bit binary activation values in the 256-bit write buffer through demultiplexing, before writing them back (from write buffer) into a PCRAM row in a partition other than the Compute Partition. (5) CNN_POOL: In addition to MAC and activation functions, CNNs may also require (e.g., CNNs) pooling functions. CNNs typically employs one pooling layer (either max or aver-age pooling) after every convolution layer [7]. ODIN invokes this command to process a CNN pooling layer. The CNN_POOL activity flow (Fig. 4.5(e)) changes depending on the size of the pooling filter (e.g., 4:1 vs 9:1). Consequently, the activity flow includes reading multiple (either 4 (Fig. 4.5(e)) or 9) 256-bit PCRAM blocks containing 32 8-bit binary operands, reducing the number of operands by applying the pooling function using the pooling logic block, and then assembling 32 pooling function outputs in a single 256-bit PCRAM block to be written back. The multiple 256-bit PCRAM blocks that are read at the beginning of this activity flow can come from the same or different PCRAM rows.

4.11 Implementation and hardware overheads

A. Implementation of CNN Processing on ODIN Although ODIN can be used for processing CNNs during forward propagation (inference) as well as back propagation (training) phases, in this chapter we only discuss and analyze how ODIN's performance during the inference phase. To begin with CNN processing, ODIN first uploads trained and quantized weights of all layers of the CNN in the PCRAM banks in the 8-bit binary format via the DMA controller (Fig. 4.3). It also uploads the input activation values to the CNN in the 8-bit binary format. After the uploading of CNN weights and inputs via

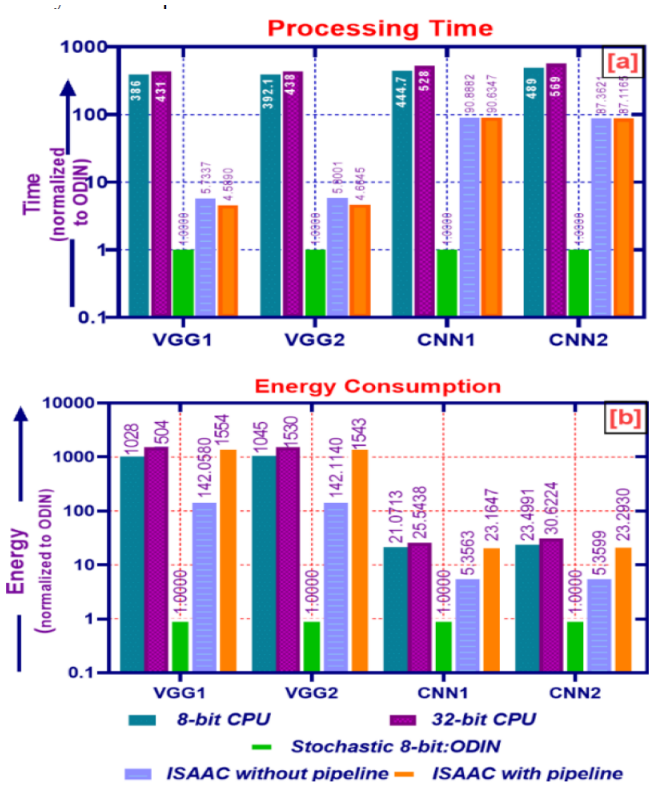

Figure 4.6: (a) Execution time and (b) energy consumption for ODIN and other considered CNN processing systems, across VGG-1/2 and CNN-1/2 topologies.

the DMA controller is over, ODIN employs the PIM controller (PIMC) to begin processing the CNN layer by layer, starting from the first layer. To process each layer, ODIN's PIMC breaks down the inference task into a sequence of PCRAM commands and control signals (following the activity flows from Section IV), which are then scheduled by the PCRAM controller to orchestrate the CNN layer processing

The processing time taken by ODIN differs for each layer, depending on the type (e.g., fully-connected, convolution, pooling) and size (i.e., #nodes, #connections) of the layer. This is because the type and size of the layer dictates (i) how many PCRAM reads and writes would be required to process the layer, and (ii) what the required storage capacity would be to store the operands. The type, size, and number of layers in an CNN depends on the architecture of the CNN model in use. Therefore, the storage requirement and the processing performance (i.e., delay, energy) of ODIN would differ between differ-ent CNN architectures. To provide a peek into these dependencies, we have evaluated the required number of PCRAM reads and writes and storage requirement for some benchmark CNN topologies, as shown in Table 4.3. Our used methodology for deriving these results is discussed in Section 5.10.

4.12 Overheads of Hardware Modifications

The overheads (delay, area, and timing overheads) of different hardware modifications made in PCRAM banks as part of our ODIN framework are listed in Table 3. In addition,

Table 4.3: ANN benchmark topologies [2].

CNN1	conv5x5-pool-784-70-10 (MNIST Dataset)
CNN2	conv7x10-pool-1210-120-10 (MNIST Dataset)
VGG1	conv3x64-conv3x64-pool-conv3x128 conv3x128-pool-conv3x256-conv3x256-conv3 x 256-pool-conv3x512- conv3x512-conv3x512-pool-conv3 x512-conv3x512 conv3x512-pool-25088-4096-4096-1000 (ImageNet Dataset)
VGG2	conv3x64-conv3x64-pool-conv3x128 conv3x128-pool-conv3x256-conv3x256-conv3x256-conv1 x 512-pool-conv3x512-conv3x512-conv3x512-con1 x512-pool-conv3x512-conv3x512-conv3x512-con1x 512- pool-25088-4096-4096-1000 (ImageNet Dataset)

Table 4.4: Area, energy and delay values for various add-on logic circuits for ODIN (scaled for 14nm CMOS).

Hardware Component	Energy (pJ)	Delay (ns)	Area (mm2)
SRAM-LUT [28]	0.297	0.316	0.402
16:8 Mux [28]	4.662	0.007	0.159
256:8 Mux [28]	4.72	0.0077	0.639
256:32 Mux [28]	18.6	0.0303	0.688
8:32 Demux [28]	18.64	0.0305	0.158
8:256 Demux [28]	149.19	0.242	0.493
256:1024 Demux [28]	902.8	1.465	1.266
ReLU Logic [25]	185	4.3	0.02
Pooling Logic [25]	2140	39.3	3.06

Table 4.5: Requirement of memory capacity, number of PCRAM reads and writes for implementing various CNN topologies on ODIN accel-erator.

	Fully Connected Layers			**Convolution Layers**			**Accuracy**
	Memory (Gb)	**Read (x106)**	**Write (x106)**	**Memory (Gb)**	**Read (x106)**	**Write (x106)**	**(%)**
VGG1	1.93	247	248	0.229	58.8	30.3	89.5
VGG2	1.96	251	252	0.234	60.01	30.9	88.51
CNN1	0.00095	1.22	1.226	0.0002	0.62	0.32	97.45
CNN2	0.00098	1.254	1.257	0.00026	0.67	0.34	97.21

ODIN also needs to add PIMC inside the PCRAM controller module. To support the PIMC, ODIN requires five additional control signals corresponding to the controller commands discussed in Section IV. Overall, the overheads of ODIN are significantly less than the overheads of other processing in crossbar memory-based accelerators from prior work. This is because, like the crossbar accelerators from prior work, ODIN does not need high-area consuming, power-hungry, and throughput-limiting analog-to-digital converters (ADCs) and digital-to-analog converters (DACs). We account for the overheads from Table 3 in our system-level evaluation presented in the next section.

4.13 Conclusions

In this work, we presented an energy-efficient and high throughput ANN accelerator called ODIN, which is a PCRAM based processing-in-memory (PIM) engine. ODIN leverages the hybrid binary-stochastic arithmetic to accelerate the essential operations of CNNs, such as multiply-accumulate (MAC), activation, and pooling, directly inside the banks of PCRAM main memory. We analyzed the performance of ODIN in terms of execution time and energy consumption, and compared it with two baseline CPUonly systems and two ISAAC-based variants, for four different benchmark CNN topologies from MLBench. The results of our analysis for the considered CNN topologies indicate that our ODIN accelerator can be at least 5.8× faster and 23.2× more energy-efficient, and up to 90.8× faster and 1554× more energy-efficient, compared to the crossbar-based in-situ CNN accelerator from prior work. These results corroborate the excellent capabilities of ODIN for accelerating CNNs. The detailed explanation related to the future direction in the chapter9

Chapter 5 A Parallel SC Based In-DRAM CNN Accelerator

5.1 Chapter Overview

This chapter provides an overview of the novel in-DRAM-based CNN accelerator (ATRIA). ATRIA employs lightweight modifications in DRAM cell arrays to implement bit-parallel rate-coded unary computing, also known as stochastic computing, based on the acceleration of multiply-accumulate (MAC) operations inside DRAM. This enables ATRIA to significantly improve the latency, throughput, and efficiency of processing CNN inferences by performing 16 MAC operations in only two consecutive memory operation cycles.

To evaluate the performance of ATRIA, four benchmark CNNs were mapped on the accelerator and compared with five state-of-the-art in-DRAM accelerators from prior work. The results of this analysis showed that while ATRIA exhibited only a 3.5% drop in CNN inference accuracy, it still achieved improvements of up to 3.2× in frames-per-second (FPS) and up to 10× in efficiency ($FPS/W/mm^2$) compared to the best-performing in-DRAM accelerator from prior work.

5.2 Introduction

Convolutional Neural Networks (CNNs) have achieved remarkable progress in recent years, and they are being aggressively utilized in real-world applications related to Artificial Intelligence (AI) and machine learning [12] [41]. In general, CNNs mimic biological neural networks and utilize compute-heavy arithmetic functions such as multiply-accumulate (MAC), nonlinear activation, and pooling. Although these CNN functions are amenable to acceleration because of a high degree of compute parallelism, their acceleration using traditional ASIC platforms (e.g., Dadiannao [12], EIE [22] [65]) is challenging because of the need to avoid the memory wall while accessing their large number of operands [10]. To address this problem, several prior works have explored processing-in-memory (PIM) designs based on the emerging non-volatile memory (NVM) crossbar technologies (e.g., ISAAC [41], PRIME [20], XNOR-RRAM [21]) as well as the traditional DRAM technology (e.g., DRISA [98] [101], SCOPE [97], DRACC [19], LACC [94]). Such PIM designs strive to avoid data movement to consequently achieve a balance between computational efficiency and memory performance while processing CNNs in situ.

However, it is challenging to support MAC operations in PIM designs. The NVM crossbar-based PIM designs, such as ISAAC [41] and PRIME [20], leverage Kirchoff's Law to perform MAC operations in the analog domain. However, such analog computing-based accelerators require power-hungry and sluggish digital-to-analog converters and analog-to-digital converters (DACs and ADCs), which diminishes the performance and energy-efficiency benefits of such accelerators. Alternatively, the DRAM-based PIM designs implement in-situ MAC operations digitally, for which they break a single MAC operation into multiple functionally complete memory operation cycles (MOCs) that are serially run on a single subarray (the smallest logical cell array in a DRAM module). Multiple such

subarrays typically work in parallel to achieve high processing throughput. Such designs require a very larger number of MOCs per MAC operation. For instance, DRISA [98] requires up to 222 MOCs per MAC. To reduce the required number of MOCs, SCOPE [97], DRACC [19], and LACC [94] employ lightweight optimizations that simplify the implementation of MAC operations. SCOPE adopts rate-coded unary (stochastic) computing to implement approximate multiplication, requiring a reduced number of up to 25 MOCs per MAC [97]. On the other hand, DRACC [19] eliminates most multiply operations by employing quantized CNNs that use ternary weights, whereas LACC [94] employs lookup table based multiply operations. Because of these optimizations, DRACC and LACC require reduced number of MOCs per MAC of up to 13 and 11 respectively. This can still incur very high latency and energy consumption as one MOC can incur up to 49 ns latency and up to 4nJ energy consumption [98] [102] [19], depending on the utilized DRAM technology node, and subarray size (bitline length). The high latency and energy values per MAC operation have prevented the DRAM-based PIM designs from being immediately adopted for CNN inference.

In this chapter, we present a novel CNN accelerator called ATRIA. ATRIA employs bit-parallel rate-coded unary (stochastic) computing, which enables it to perform 16 MAC operations in only 2 consecutive MOCs. ATRIA is most related to SCOPE [97]. It significantly improves SCOPE in two ways. First, SCOPE uses rate-coded unary (stochastic) computing to perform only multiply operations, whereas it uses conventional binary arithmetic to perform accumulated operations. In contrast, ATRIA performs both multiply and accumulate operations using bit-parallel rate-coded unary (stochastic) computing. Second, both SCOPE and ATRIA require expensive binary-to-stochastic (B-to-S) and stochastic-to-binary (S-to-B) conversions of operands, but ATRIA is better able to hide the latency of these conversions by successfully removing them from the critical processing path. Moreover, ATRIA restricts the precision errors induced due to the rate-coded unary (stochastic) computing-based accumulated operations by employing stochastic operands that are 2× larger in size. As a result, ATRIA exhibits only a 3.5% drop in CNN inference accuracy on average compared to SCOPE. Despite this slight drawback, ATRIA substantially outperforms SCOPE as well as other in-DRAM accelerators such as DRISA and LACC in terms of the latency, throughput (frames-per-second (FPS)), and efficiency (FPS/W/mm^2) of processing state-of-the-art CNNs.

5.3 Concept of Bit-Parallel Rate-Coded Unary (stochastic) computing

The use of rate-coded unary (stochastic) computing simplifies the implementation of complex arithmetic functions, such as multiplication and accumulation, by reducing them to simple bit-wise logical operations [9]. To perform a multiplication of 2 N-bit stochastic operands (A and B in Fig 6.1(a)) in the bit-serial manner, the bit-streams of the operands are applied to an AND gate serially, and the bit-wise output of the AND gate is collected for total N clock cycles to generate the multiplication output bit-stream (C in Fig 6.1(a)). Similarly, to perform a scaled accumulation of 4 (or more) N-bit stochastic operands in the bit-serial manner (A, B, C, D in Fig 6.1(b)), the bit-streams of the operands are applied to a MUX, whose bit-wise output is selected by a 2-bit (or larger) random number (RND in Fig 6.1(b)) every clock cycle for total N clock cycles, to generate the output bit-stream

that represents a scaled accumulation (E in Fig 6.1(b)). To reduce the area and static power consumption of computing, such bit-serial implementation of rate-coded unary (stochastic) computing compromises the latency of computing.

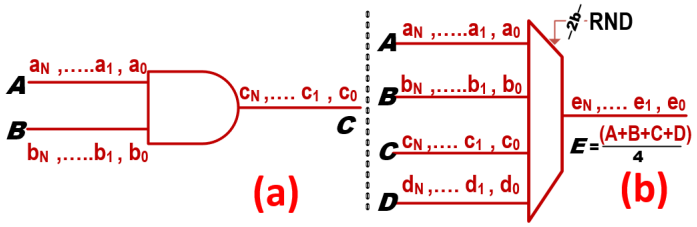

Figure 5.1: Bit-serial rate-coded unary (stochastic) computing circuits for (a) multiplication (AND gate), (b) scaled accumulation (MUX).

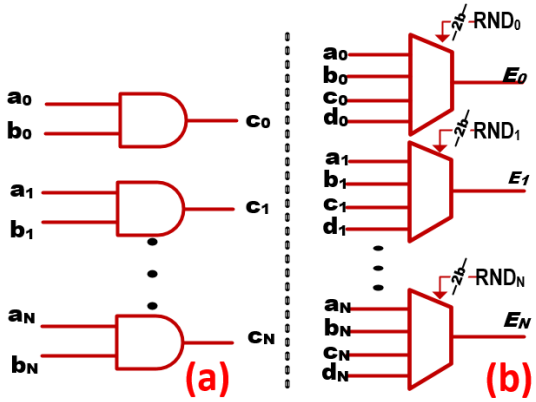

Figure 5.2: Bit-parallel rate-coded unary (stochastic) computing circuits for (a) multiplication (array of AND gates), (b) scaled accumulation (array of MUXs). Here, the individual N bits of operands A, B, C, and D from Fig 6.1 are striped across N copies of AND gates and MUXs.

In contrast, we observe that the latency of computing can be improved by N× if the rate-coded unary (stochastic) computing can be implemented in the bit-parallel manner. For example, if N copies of AND gates and MUX circuits are available (Figs. 6.3(a) and 6.3(b)), the N-bit outputs for the stochastic multiplication and scaled accumulation can be obtained in one clock cycle in the bit-parallel manner. In a nutshell, the idea for such bit-parallel implementation of rate-coded unary (stochastic) computing is to transform the input bit-streams into bit-vectors by striping them across the N copies of the AND gates and MUX circuits, and then perform bit-wise AND and MUX operations to generate output bit-vectors. For instance, the individual N bits a_1 to a_N, b_1 to b_N, c_1 to c_N, and d_1 and d_N of operands A, B, C, and D from Fig 6.1(b) are striped across N copies of MUXs in Fig 6.3(b). As a result, the individual N bits of the scaled accumulation output E can be collected in a bit-parallel manner from N MUXs. For such bit-parallel scaled accumulation (i.e., MUX operation), total N RND signals (RND_1 to RND_N) are needed which can be

generated a priori and made available in a parallel manner (Fig 6.3(b)). Although Fig 6.3(b) illustrates bit-parallel scaled accumulation for only four input stochastic operands (A, B, C, and D), this concept can be extended for more or less than 4 input stochastic operands as well.

Such bit-parallel rate-coded unary (stochastic) computing naturally fits well for in-DRAM processing of applications because the inherent parallelism of DRAM makes it fundamentally easy to provision data in the bit-parallel manner. Our proposed in-DRAM accelerator ATRIA employs such bit-parallel rate-coded unary (stochastic) computing to implement in-DRAM MAC operations for the first time, and exploits the benefits of such implementation to substantially improve the latency and throughput of in-DRAM CNN processing, compared to the in-DRAM CNN processing accelerators from prior work.

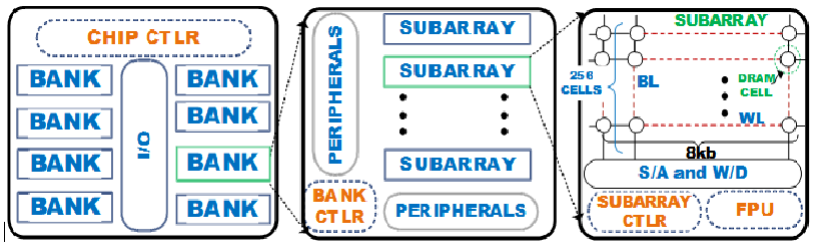

Figure 5.3: The hierarchical structure of our ATRIA accelerator chip.

5.4 ATRIA: Overview

Our ATRIA accelerator architecture employs an 8Gb DRAM module with 8 chips. Fig 5.3 illustrates the schematic of one such chip. Each chip has 8 banks, with 64 subarrays per bank, and 32 mats per subarray of 256×256 bits size each. Each row in a subarray is of 8Kb size, therefore, each subarray contains total 8Kb sense amplifiers (S/As) and write drivers (W/Ds). Each subarray acts as a processing element (PE), which is defined as the smallest independent cell-array structure that can perform computing. Therefore, there are total 4096 PEs in ATRIA. Like the other in-DRAM accelerators from prior work (e.g., DRISA [98], SCOPE [97], LACC [94]), the PEs in ATRIA can also operate in parallel to process CNN inference in situ. To process CNN inference, each PE (i.e., subarray) in ATRIA employs a feature processing unit (FPU), as shown in Fig 5.3. In addition, to orchestrate these in-parallel processing operations inside the PEs, ATRIA employs hierarchical controllers (chip, bank and subarray controllers (CTLRs) in Fig 5.3). The operation of these hierarchical CTLRs is described in Section 5.4. The structure and operation of each FPU in ATRIA support our concept of bit-parallel rate-coded unary (stochastic) computing for in-situ processing of CNNs, as discussed next.

Structure of a PE in ATRIA

A PE of our ATRIA accelerator is basically a DRAM subarray that is integrated with an FPU and a subarray CTLR, as illustrated in Fig 5.4. The subarray part of the PE is structured in the manner the conventional DRAM subarrays are organized [23] [28]. Therefore, in this

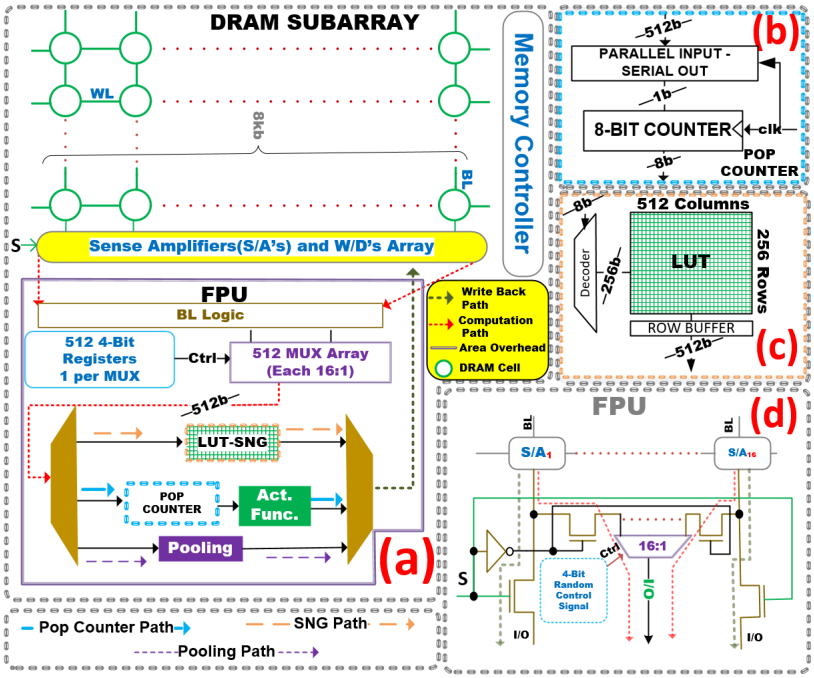

Figure 5.4: Schematic of a processing element (PE) of ATRIA. (a) Schematic of a subarray and feature processing unit (FPU); (b) pop counter for S-to-B conversion [2]; (c) LUT for B-to-S conversion; (d) a 16:1 MUX and its connections with S/As as part of the FPU

section, we only provide details of the structure of the FPU. The role of the subarray CTLR is discussed in Section 5.4. The FPU consists of various hardware components that support the implementation of the following six functions: (i) bit-parallel stochastic multiply operation (MUL), (ii) bit-parallel stochastic accumulate operation (ACC), (iii) binary to stochastic (B-to-S) conversion, (iv) stochastic to binary (S-to-B) conversion through pop counter (PC), (v) nonlinear activation function ReLU, and (vi) max pooling function. To support bit-parallel MUL, three 8Kb rows of the subarray (Row 1, Row 2 and Row 3 in Fig 5.4(a)) are reserved and operated following the triple row activation and charge-sharing protocol of AAP memory operation cycle (MOC) from Ambit [102] (see Section 5.4).

The hardware components that support bit-parallel ACC consist of an array of 512 copies of 16:1 MUXs and their associated 512 copies of 4-bit registers (Fig 5.4(a)). These 4-bit registers store the pre-determined random values that enable the output selection (16:1) for their respective MUXs. Each MUX has 16 inputs, therefore, the total number of inputs for the entire array of 512 MUXs is 8Kb. These 8Kb MUX inputs are connected to 8Kb S/As, with 16 adjacent S/As feeding one MUX and vice versa (Fig 5.4(d)). Note that the S/As in the commodity DRAMs typically connect to I/O logic through signal S and related control transistors (M_1 to M_16) (Fig 5.4(d)). To facilitate connections of S/As to MUXs, ATRIA employs one additional inverter (INV) and 16 transistor switches (T1 to T16) per MUX, which can be controlled by the same signal S (Fig 5.4(d)). An 8Kb row from the subarray can be read into 8Kb S/As (Fig 5.4(a)), which can hold total 16 stochastic bit-vectors of 512-bit size each (16 × 512 = 8Kb). These 16 stochastic bit-vectors can be striped across 512 MUXs, so that each individual bit of a bit-vector is fed into a different MUX with each

MUX having all its 16 inputs from 16 different bit-vectors. This arrangement sets up the array of MUXs to perform a 16-operand scaled ACC in the bit-parallel manner, following our concept of bit-parallel rate-coded unary (stochastic) computing discussed in Section 5.3. The detailed functioning of this array of MUXs for performing scaled ACC is presented in Section 5.4.

In addition, to implement in-memory B-to-S conversion, each FPU in ATRIA employs a lookup table (Fig 5.4(c)). Our idea of using lookup table-based B-to-S conversion is inspired from the design of SCOPE accelerator [97]. This enables ATRIA to employ the deterministic method for B-to-S conversion to eliminate correlation errors [97]. Moreover, each FPU in ATRIA employs an additional lookup table to perform ReLU (Fig 5.4(a)). Further, it also incorporates a pop counter to perform in-memory S-to-B conversion (Fig 5.4(b)), as well as logic to implement max pooling function (Fig 5.4(a)). ATRIA implements the max pooling and ReLU functions in the binary domain. This mandates that the results of processing of every CNN layer's parameters always go through S-to-B, ReLU, and then B-to-S conversions before they can activate processing of the next CNN layer. This in turn eliminates the undesirable propagation of precision errors (which are very common in rate-coded unary (stochastic) computing [9]) between the stochastic operations of two consecutive CNN layers (see more on errors in Section 5.5. The overheads of incorporating FPUs in ATRIA PEs are discussed in Section 5.4. The next section describes the functioning of an FPU-enabled PE of our ATRIA accelerator.

Functioning of a PE in ATRIA

Each PE of our ATRIA accelerator can perform all essential functions required for processing CNNs, such as MAC, max pooling, and ReLU. In addition, since ATRIA employs rate-coded unary (stochastic) computing, each PE can also perform important functions for implementing rate-coded unary (stochastic) computing, such as B-to-S and S-to-B (pop count) conversions. On one hand, each PE performs B-to-S, S-to-B (pop count), ReLU, and max pooling functions by relaying the related operands along the data processing path in the FPU through the corresponding hardware components (Fig 5.4(a)). To orchestrate the relaying of the operands to perform these functions, the PE makes use of the subarray CTLR whose functioning along with the functioning of other hierarchical CTLRs in ATRIA is discussed in Section 5.4. On the other hand, each PE of ATRIA can perform a MAC function (F_{MAC}) of 16 stochastic operands of 512-bit size each, by employing a series of total five memory operation cycles (MOCs) (similar to the AAP/AP MOC from [98] [102]). These MOCs engage the reserved rows Row 1, Row 2 and Row 3 (Fig 5.4(a)) and the MUXs in the FPU, as discussed next.

Fig 5.5 illustrates how ATRIA performs F_{MAC}. ATRIA performs F_{MAC} in two main steps. Step 1 engages the reserved subarray rows Row 1, Row 2, and Row 3 to perform MUL. Step 2 engages the array of MUXs to perform ACC. Before performing F_{MAC}, ATRIA first makes the involved stochastic operands available in the reserved subarray rows Row 1 and Row 2. For that, it performs two MOCs similar to RowClone [25] to copy the contents of two source rows into Row 1 and Row 2 respectively. Consequently, Row 1 contains 16 512-bit operands N_1 to N_{16} (Fig. 5(a)). Similarly, Row 2 contains 16 512-bit operands M_1 to M_{16} (Fig 5.5(a)). In addition, ATRIA initializes Row 3 with '0's at system

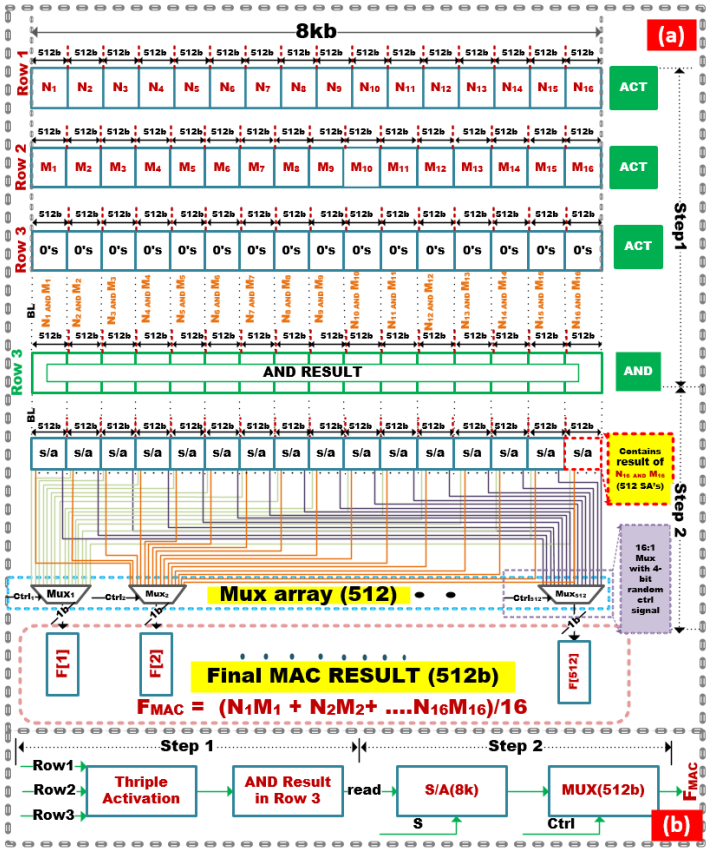

Figure 5.5: A schematic showing the operation of a PE of ATRIA to perform a 16-operand multiply-accumulate (MAC) function (F_{MAC}).

boot. After these initial steps, ATRIA schedules Step 1 of F_{MAC}, which employs the triple row activation and charge-sharing based MOC from Ambit [102] to perform bit-parallel logical AND (i.e., stochastic MUL) of the involved operands N_1 to N_{16} and M_1 to M_{16}. At the end of the MOC for Step 1, Row 3 contains the results of bit-parallel logical AND, i.e., N_1 AND M_1 to N_{16} AND M_{16} (Fig 5.5(a) and Fig 5.5(b)). These results essentially represent the outcome of bit-parallel stochastic MUL, i.e., N_1M_1 to $N_{16}M_{16}$. After this, ATRIA schedules Step 2 of F_{MAC}, where it performs a MOC to read the stochastic MUL results from Row 3 into S/As. These results from S/As are then pushed through the array of 16:1 MUXs, MUX1 to MUX512. The 512-bit output of this array of MUXs is selected using the pre-latched random control signals RND1 to RND512. This 512-bit output is the stochastic scaled ACC of the input operands N_1M_1 to $N_{16}M_{16}$. In other words, this 512-bit output presents $F_{MAC} = (N_1M_1 + N_2M_2 + \ldots + N_{16}M_{16})/16$ (Fig 5.5(a) and Fig 5.5(b)). ATRIA then uses one more MOC to store the result of this F_{MAC} into a row in the subarray through W/Ds. Thus, ATRIA uses only 5 MOCs (2 MOCs for initializing Row 1 and Row 2, 1 MOC for MUL, 1 MOC for ACC, and 1 MOC for write back) to perform a scaled MAC function F_{MAC} (also called dot-product) of 16 stochastic operands. In other words, if a MAC operation is conventionally defined as a MUL of two operands followed by an accumulate operation (i.e., A = A + N_iM_i), then ATRIA uses only 5 MOCs to perform 16 MAC operations in parallel. However, we find from our evaluation results in Section 5.5

that the use of bit-parallel rate-coded unary (stochastic) computing in ATRIA can increase precision errors. Nevertheless, we also find that the increased precision errors are worth tolerating for dues to the substantial performance benefits of ATRIA.

System Integration and Controller Design

In this section, we describe how our ATRIA accelerator integrates with the host system and how the hierarchical controllers of ATRIA orchestrate the processing of CNNs. ATRIA integrates with the host system in the same way the conventional GPU or FPGA based accelerators do through PCIe bus. For a CNN processing using ATRIA, the host system stores the weighting parameters and inputs of the CNN in the individual PEs (subarrays) of ATRIA via direct memory access (DMA). We adopt the strategy from SCOPE [97], wherein the weighting parameters are stored in ATRIA in the stochastic format. This strategy ensures that in-situ B-to-S conversions are required only for activation parameters, which dramatically reduces the number of in-situ B-to-S conversions. As a result, the latency and energy of processing CNNs with ATRIA are dramatically reduced as well.

After storing the inputs and weighting parameters of a CNN in PEs of ATRIA, the host-side ATRIA CTLR (not shown in Fig 5.3) orchestrates the processing of the CNN in conjunction with the hierarchical ATRIA CTLRs shown in Fig 5.3. The host-side ATRIA controller generates a series of μ-operations, which are received by the hierarchical ATRIA CTLRs. We adopt the designs from [98] for these CTLRs. These CTLRs support simultaneous multi-subarray/bank activation for better parallelism. The first chip-level CTLR is essentially a decoder, and it also helps with inter-bank data movement. The bank-level CTLRs decode the μ-operations and convert them into addresses, vector lengths, and control codes, and then send them to subarray CTLRs in the active subarrays. The subarray CTLR consists of address latches, local decoders, and counters. The address latches are essential for multi-subarray activation [98]. The counters are used for continuously updating addresses to local subarray decoders. In addition, the subarray CTLR also contains buffers to support the communication of operands.

Inter-bank and inter-subarray data communications in ATRIA are supported through the interconnects design adopted from LISA [26]. Data communications are carried out in binary format instead of stochastic format, which results in better energy-efficiency [97]. Also, the inclusion of buffers in the subarray CTLRs enables pipelined data communications, which enables better use of resources and efficient hiding of long latencies, reducing the memory bottleneck to improve the throughput of CNN processing with ATRIA.

Overhead Analysis

Table 5.1 lists the latency, energy, and area overheads of various hardware components that are part of the FPUs inside the PEs of our ATRIA accelerator. These results are based on our logic synthesis analysis for 22nm node. We considered standard SRAM for LUT implementation. After accounting for the extra area overhead of these components from Table Table Table 5.1, the total area for 8Gb ATRIA accelerator becomes 77mm^2. For comparison, DRISA-1T1C-NOR [98], DRISA-3T1C [97], SCOPE-Vanilla [97], SCOPE-H2D [97], and LACC [94] consume 55mm^2, 64.6mm^2, 259.4mm^2, 273.4mm^2, and 61mm^2

Table 5.1: Latency, energy, and area overhead values of various hardware components of the FPUs in the PEs of ATRIA.

Component	Total Area (mm^2)	Latency (ns)	Energy per PE (pJ)
16:1 MUXs for ACC	1.3×10-3	2	10
4-bit Registers for RND Storage	1.1×10-5	2	15.6
B-to-S LUT (512×256)	3.4	1	0.3
S-to-B Pop Counter (PC) (2GHz)	2.1×10-5	256	153.6
ReLU LUT	1.2	1	0.3
Max Pooling Logic	4.1	5	940

area respectively. Thus, ATRIA consumes larger area than DRISA-1T1C-NOR, DRISA-3T1C, and LACC. Nevertheless, ATRIA still achieves substantially better area and energy efficiency compared to these accelerators (Section 5.5). Similarly, despite the S-to-B pop counter in ATRIA incurring a long latency of 256ns (Table 5.1), the performance of ATRIA does not get much affected, as ATRIA manages to keep this latency out of the critical processing path (Section 5.5).

5.5 Evaluation

Modeling and Setup for Evaluation

We evaluate ATRIA and compare it with other in-DRAM accelerators from prior work such as SCOPE-Vanilla [97], SCOPE-H2D [97], DRISA-1T1C-NOR [98], DRISA-3T1C [98], and LACC [94]. We first evaluate the per-MAC latency, per-MAC energy, and total area values for our considered accelerators. We divide the evaluation of per-MAC latency/energy into two parts: latency/energy of a multiply operation (MUL) and latency/energy of an accumulate operation (ACC). All our considered accelerators follow the AAP/AP memory operation cycle (MOC) from Ambit [102]. Therefore, the latency and energy values per MOC and total number of MOCs per MAC are evaluated first for all considered accelerators. Different accelerators have different latency and energy per MOC because they employ different lengths of local bitlines in their subarrays. For example, DRISA [98] and SCOPE [97] employ shorter local bitlines with only 64 cells per bitline. In contrast, LACC employs 512 cells per bitline, whereas ATRIA employs 256 cells per bitline. Shorter bitlines typically yield lower latency per MOC [23]. We evaluate latency using SPICE [17] based modeling of local bitlines. To evaluate per-MOC energy as well as total accelerator area, we used CACTI [16]. We developed a custom simulator in Python to model the MOC-accurate transaction-level performance behavior of our considered accelerators, as well as to evaluate system-level performance metrics such as frames-per-second (FPS), latency, efficiency (FPS/W/mm^2), and memory bottleneck ratio. Memory bottleneck ratio is defined as the ratio of total stall time (time for which an accelerator needs to wait for the operands) over total inference processing time. We considered four state-of-the-art CNNs to evaluate these metrics. The quantized versions of these CNN models were trained using PyTorch for ImageNet dataset and 8-bit fixed-precision of activation and weight parameters. These activation and weight parameters were extracted and provided as the input to our Python based performance simulator, which also took our evaluated energy, latency, and

Table 5.2: Average APE (μAPE), standard deviation in APE (σAPE) and CNN testing accuracy (A) for SCOPE-Vanilla, SCOPE-H2D and ATRIA for various CNNs.

CNN	SCOPE-Vanilla			SCOPE-H2D			ATRIA		
Benchmarks	**μAPE**	**σAPE**	**A(%)**	**μAPE**	**σAPE**	**A (%)**	**μAPE**	**σAPE**	**A(%)**
Alexnet	0.23	0.04	93.6	0.09	0.01	96.7	0.33	0.05	92.2
GoogleNet	0.30	0.05	87.7	0.17	0.03	88.5	0.41	0.07	87.7
VGG16	0.35	0.05	91.9	0.21	0.03	95.1	0.53	0.09	90.2
ResNET-50	0.26	0.04	90.1	0.12	0.02	93.6	0.47	0.08	89.8

area values for our considered accelerators as the input. Next, we present and discuss the results of our simulation-based study.

Precision Error and Accuracy Results

ATRIA has one caveat compared to SCOPE. The use of MUX based bit-parallel stochastic accumulation in ATRIA can increase the absolute precision error (APE) of computing, as explained in [9]. An APE for an operation (i.e., MUL or ACC) is defined as the absolute difference between the expected result and the observed result of the operation. From [9] and [27], APE depends on the operand values, input size (i.e., number of operands), and operand size (i.e., bit-stream length). For a MUX based stochastic ACC with an input size of 16 (as is the case for ATRIA), the average APE (μAPE) can be reduced to an acceptable value in the range between 0.2 to 0.54, if the operand size is kept 512 bits or longer [9] [27]. Therefore, we increase the operand size, i.e., bit-vector length, of the bit-parallel stochastic operands in ATRIA to 512 bits from their full-precision length of 256 bits (corresponds to 8-bit binary operands). The resultant μAPE values and corresponding standard deviation in APE (σAPE) for four benchmark CNNs are listed in Table 5.2. The μAPE and σAPE values in Table 5.2 were obtained for the complete set of individual APEs for all MAC results required in respective CNNs when the inferences of these CNNs are implemented on ATRIA, SCOPE-Vanilla and SCOPE-H2D for the ImageNet dataset. Table 5.2 also lists the inference accuracy results. As evident, ATRIA exhibits 2.9× and 1.5× more μAPE, and 3.2× and 1.6× more σAPE than SCOPE-H2D and SCOPE-Vanilla respectively, on average across the CNNs. Nevertheless, compared to SCOPE-H2D and SCOPE-Vanilla, ATRIA exhibits only 3.5% and 0.85% drop in inference accuracy on average across the CNNs, which we reason is acceptable due to the significant performance benefits of ATRIA, as evident from Sections 5.5 and 5.5.

Per-MAC Latency Results

Table 5.3 lists our evaluated latency values and number of PEs (#PEs) for ATRIA and other in-DRAM CNN accelerators. The latency values include values for MUL and ACC in number of MOCs (#MOCs), latency per MOC in ns, as well as the latency values for LUT-based B-to-S conversion and pop-count (PC) operations (required for S-to-B conversion). From Table Table 5.3, ATRIA holds three crucial advantages. First, it exhibits smaller per-MAC latency over SCOPE, DRISA and LACC Table Table 5.3. This is because ATRIA performs 16 MAC operations in parallel. For that, ATRIA uses total 5 MOCs (total 85ns

Table 5.3: Comparison of various accelerators with ATRIA, in terms of number of PEs (#PEs) and latency of MUL, ACC, MAC, binary to stochastic conversion (B-to-S), and pop count (PC) operations.

Various Accelerators	Latency Values						#PEs
	MUL #MOCs	ACC #MOCs	MOC (ns)	MAC (ns)	B-to-S (ns)	PC (ns)	
DRISA-3T1C [98]	200	11	8	1768	-	-	32768
DRISA-1T1C-NOR [98]	200	22	10	2110	-	-	16384
LACC [94]	1	10	21	231	-	-	16384
SCOPE-Vanilla [97]	3	4	8	56	1	176	65536
SCOPE-H2D [97]	21	4	8	200	1	176	65536
ATRIA	3/16	2/16	17	5.25	1	256	4098

latency with each MOC incurring 17ns latency) (Section 5.4); 2 MOCs to copy the operand rows, 1 MOC to perform 16 in-parallel MULs, 1 MOC to perform 16 in-parallel ACCs, and 1 MOC to store the MAC result. In Table 5.3, for ATRIA, 2 MOCs for operand row copy are counted in total MUL MOCs, and 1 MOC for MAC result store is counted in total ACC MOCs. Thus, by performing 16 MAC operations in parallel, ATRIA achieves shorter per-MAC latency.

Second, ATRIA can better hide the latency for PC operations, compared to SCOPE. This is because, SCOPE utilizes full adder-based PC operations that need to be performed inside PEs. Therefore, despite using the as-late-as-possible (ALAP) scheduling algorithm, PC operations in SCOPE inevitably stall the PEs. In contrast, ATRIA offloads PC operations to dedicated serial counters (operating at 2GHz) per PE (Sections 5.4 and 5.4). As a result, ATRIA does not need to stall PEs for PC operations, enabling itself to better hide PC latency. Therefore, although ATRIA yields higher latency per PC operation than SCOPE (Table Table 5.3), ATRIA efficiently hides this higher latency, not letting it affect the performance.

Third, ATRIA exhibits smaller bottleneck ratio compared to SCOPE and DRISA (see Fig 5.6(d) in Section 5.5). Bottleneck ratio is defined in Section IV.A. ATRIA achieves lower bottleneck ratio, because the use of massively large number of PEs in SCOPE and DRISA results in unavoidable inter-PE communication latency, a substantial portion of which remains on the critical processing path because of the inherently limited parallelism available for such inter-PE communications. In contrast, ATRIA is better at hiding the inter-PE communication latency, due to its smaller number of PEs and its LISA [26] substrate-based implementation of intra-bank, inter-bank, and inter-PE data communications (Section 5.4).

CNN Inference Performance Results

We evaluate the preformance of ATRIA and compare it with the following inDRAM CNN accelerators from prior work: DRISA3T1C [98], DRISA1T1CNOR [98], SCOPE-Vanilla [97], SCOPEH2D [97], and LACC [94]. We consider four CNNs: VGG16 [13], Alexnet [11], ResNET_50 [15], GoogleNET [14], with the ImageNet dataset. Using the setup described in Section 5.5, we evaluated latency, FPS, FPS/W/mm^2, and bottleneck ratio, for batch size of 1 and 64. Fig 5.6(a) shows efficiency (FPS/W/mm^2) results. For

batch size 1, ATRIA is 18×, 64×, 98× and 50× more efficient than DRISA-1T1C-NOR, DRISA-3T1C, SCOPE-Vanilla, and SCOPE-H2D, respectively, on average across CNNs. However, ATRIA is 15% less efficient than LACC, due to the LACC's lower area (Section 5.4). Nevertheless, for batch size 64, ATRIA is more efficient than LACC as well. ATRIA is 136×, 522×, 3.4×, 71×, and 95× more efficient than DRISA-1T1C-NOR, DRISA-3T1C, LACC, SCOPE-Vanilla, and SCOPE-H2D, respectively, on average across CNNs. In general, ATRIA is more efficient due to the following two reasons: (i) better FPS due to lower per-MAC latency (Table Table 5.3), (ii) reasonable average power consumption of 23.4W.

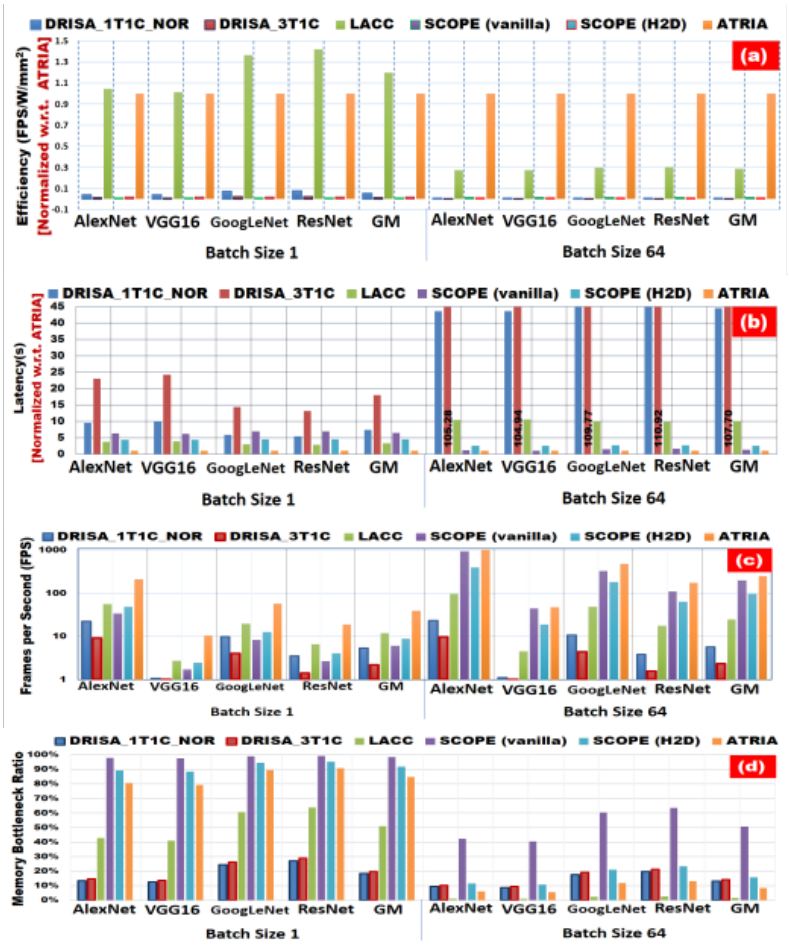

Figure 5.6: (a) Efficiency (FPS/W/mm^2), (b) latency, (c) throughput (FPS), and (d) memory bottleneck ratio (MBR) results for various in-DRAM accelerators across CNNs. GM means geometric mean.

Fig 5.6(b) shows CNN processing latency results normalized w.r.t. ATRIA. For batch size 1, ATRIA achieves 7.4×, 18×, 3.3×, 6.5×, and 4.4× lower latency than DRISA-1T1C-NOR, DRISA-3T1C, LACC, SCOPE-Vanilla, and SCOPE-H2D, respectively, on average across CNNs. Similarly, for batch size 64, ATRIA achieves 44×, 107×, 10×, 1.2×, and 2.6× lower latency than DRISA-1T1C-NOR, DRISA-3T1C, LACC, SCOPE-Vanilla, and SCOPE-H2D, respectively, on average across CNNs. ATRIA achives lower CNN process-

ing latency because of its lower per-MAC latency and its ability of efficienty hiding its higher S-to-B conversion latency. Moreover, DRISA-1T1C-NOR, DRISA-3T1C, LACC, SCOPE-Vanilla, SCOPE-H2D, and ATRIA achieve 60×, 59×, 30×, 2×, 6×, and 10× higher latency for batch size 64 than batch size 1. This is because the higher parallelism of SCOPE variants (more #PEs in Table Table 5.3) allow them to process larger batch size without saturating the latency benefits, by distributing the batch processing across multiple PEs.

Fig 5.6(c) shows FPS results. For batch size 1, ATRIA has on average 7.4×, 18×, 3.3×, 6.5×, and 4.4× higher FPS than DRISA-1T1C-NOR, DRISA-3T1C, LACC, SCOPE-Vanilla, and SCOPE-H2D, respectively. For batch size 64, ATRIA has on average 44×, 107×, 10×, 1.2×, and 2.6× higher FPS than DRISA-1T1C-NOR, DRISA-3T1C, LACC, SCOPE-Vanilla, and SCOPE-H2D, respectively. ATRIA has higher FPS due to the combined effects of lower per-MAC latency and lower memory bottleneck ratio (Section 5.5), as discussed next.

Finally, Fig 5.6(d) gives memory bottleneck ratio (MBR) results. MBR for all accelerators reduces for batch size 64 than batch size 1 because increasing batch size to 64 does not substantially increase the stall time for weighting parameter accesses, but doing so increases CNN processing time due to the required time-sharing of resources across multiple batch inputs, resulting in lower MBR. For batch size 64, ATRIA has lower MBR than all other accelerators, except for LACC. LACC has only 1% MBR for batch size 64, which corroborates the results from [94]. This is because the kernel mapping algorithm used in LACC enables better resource utilization. SCOPE variants have the highest MBR for both batch sizes because in SCOPE the latency for S-to-B conversions come in the critical path (Section 5.5). In contrast, ATRIA is able to better hide this latency to achieve lower MBR.

5.6 Conclusions

In this chapter, we presented an energy-efficient and high-throughput CNN accelerator called ATRIA, which utilizes the novel concept of bit-parallel rate-coded unary (stochastic) computing to achieve ultra-low latency for multiply-accumulate (MAC) operations. We mapped four benchmark CNNs on ATRIA to compare its performance with five state-of-the-art in-DRAM accelerators from prior work. The results of our analysis show that ATRIA exhibits only a 3.5% drop in CNN inference accuracy and still achieves improvements of up to 3.2× in frames-per-second (FPS) and up to 10× in efficiency (FPS/W/mm^2), compared to the best-performing in-DRAM accelerator from prior work. These results corroborate the excellent capabilities of ATRIA for accelerating the inference tasks of deep CNNs.

Chapter 6 A Substrate for In-DRAM StoB for CNN Applications

6.1 Abstract of the chapter

Stochastic Computing (SC)-based in-situ Convolutional Neural Network (CNN) accelerators are a popular approach used to reduce the hardware complexity required for arithmetic computation, such as Multiply-Accumulate (MAC) and General Matrix Multiplication (GEMM) in the CNN accelerator. However, for stochastic in-memory-based CNN accelerators, the intermediate-layer results of these CNN benchmark applications must be stored in the binary domain to reduce memory overhead, and Stochastic to Binary (StoB) conversion becomes a critical component of SC CNN accelerators. The generic counter-based StoB time consumption increases significantly with the input stochastic bitstream length.

To address this issue, we propose AGNI, a novel high-performance and energy-efficient in-memory DRAM-based bit-parallel StoB conversion method with minimal hardware modification. In this chapter, we introduce a novel Analog-to-Digital Converter (ADC) technique to extract the analog equivalent voltage of the input stochastic bitstream data, which is converted to unary data (temporal data) by leveraging the DRAM circuitry. Finally, the unary data is converted to binary using the priority encoder.

To evaluate the performance of AGNI, we simulate the proposed circuits in LTSPICE with a $45\eta m$ gpdk technology node. We also perform post-processing using the Python simulator for the StoB's output binary bitstream operand lengths ranging from 4-bits to 8-bits. We compare AGNI with existing pop counters (PC), such as parallel PC and serial PC, regarding the area, energy, and latency. The simulation results show that AGNI has tremendous improvement in computation speed over similar PCs. Specifically, the simulation results show that AGNI has a significant Energy-Delay Product (EDP) saving of 350× over similar PC.

We believe that this study opens up new horizons for StoB computing, enabling the implementation of smaller yet more accurate arithmetic circuits for in-memory accelerators than conventional StoB conversion.

6.2 Introduction

Convolutional Neural Networks (CNNs) gained immense popularity in recent times and have been extensively used in many real-world applications related to machine learning (ML) and Artificial Intelligence (AI) [62] [103]. These CNNs mimic the structure of the human biological brain's neural network. CNNs use computationally complex arithmetic operations like multiply-accumulate (MAC), nonlinear activation, and pooling. Nonetheless, these CNN functions can be accelerated due to their high degree of compute parallelism. Their acceleration using conventional ASIC platforms (e.g., Dadiannao [62], EIE [104]) is challenging for the reason of avoiding the memory wall while accessing their large number of operands [103]. In CNNs, the significant compute requirement and data-intensive operations are MAC(multiply and accumulate) operations. Further, the input

computation data on the current cutting-edge CNN benchmark applications is massive, such as RESNET-50 [105], GoogLeNet [5] with tensor size as large as 15GB. Moreover, using the von-Neumann architecture on this huge amount of data is not energy-efficient and incurs significant latency for the data movement from the CPU and memory [103].

Several prior works have explored processing-in-memory (PIM) designs based on the emerging non-volatile memory (NVM) crossbar technologies (e.g., ISAAC [103], PRIME [106], XNOR-RRAM [107]) as well as the traditional DRAM technology (e.g., DRISA [108], SCOPE [3], DRACC [109], LACC [58]) to address this data migration length issue. Such PIM solutions work to prevent data migration in order to balance memory performance and computational efficiency while processing CNNs locally.

But, computing MAC operation using the PIM is challenging. Analog NVM crossbar-based PIM CNN MAC accelerators (e.g., ISSAC [103] and PRIME [106]) require power-inefficient analog-to-digital (ADC) and digital-to-analog (DAC) converters. This additional circuitry indulges in performance degradation and incurs a huge area overhead. Alternatively, digital in-DRAM-based MAC accelerators breaks a MAC operation into multiple functionally complete operation cycles (MOCs) [109], such as DRISA (222 MOCs) [108], DRACC (13 MOCs) [109], and LACC (11 MOCs) [58]. However, these accelerators require huge MOCs and each MOC can incur up to 49ns and 4nJ of latency and energy, respectively.

Subsequently, to mitigate large MOCs requirements and area overhead, SCOPE [3] and ATRIA [4] use a Stochastic computing-based MAC PIM accelerator to reduce the hardware footprint and leverage computing parallelism. However, in these accelerators, Stochastic to Binary (StoB) conversion consumes maximum time duration. Even though ATRIA StoB operations are hidden from the critical path to some extent, they are not always. SCOPE and ATRIA use a parallel pop counter (PC) and Serial pop counter-based StoB conversion, respectively. Moreover, this counter circuitry's overhead and energy consumption depend on the bit-precision. These DRAM PIM accelerators require high-speed counters circuitry (GHz range). However, the generic DRAM cells are low-frequency operating and low-power grids. Thus, this lead to a lot of EM/IR challenges along with fabrication in designing the high-speed counter circuitry. Thus, there is a need for a StoB converter with the least area overhead and high energy efficiency in the DRAM PIM accelerator.

This chapter presents a novel in-situ DRAM-based StoB converter called AGNI. AGNI employs bit parallelism, which helps to perform even 256-bit SN conversion to 8-bit BN within $55ns$ and EDP of just $21.2ns.pJ$. The area of the overhead of the proposed accelerator for the BN length of 8-bit is just 4%, with minimal hardware modifications. AGNI performs the StoB in 4 steps. *READ*:- The first step is reading the stochastic number. *S_to_A*:- Second step is to convert race-logic/SC-based unary number into an equivalent analog voltage (V_{ANG})(explained in Section V-B). *A_to_U*:-Third analog voltage (V_{ANG}) is converted into the temporal logic-based unary representation using the in-memory-based ADC network(explained in Section 6.6). *U_to_B*:- Finally, the converted temporal unary number is sent to the priority encoder (PE) to calculate the total number of the ones in the unary bit stream data. Also, AGNI hides the latency of the PE from the critical path and thus helps in parallelism.

The organization of this chapter is as follows: Starts with the introduction to StoB conversion in the CNN accelerators and the need for an efficient StoB converter; Next, we

provide the background on the drawbacks of a prior StoB converter for in-memory accelerators; Later, architectural overview and hardware modifications are explained; Further, we analyzed the proposed design and provided a comparison result with previous works; Finally, we conclude the chapter with future directions.

6.3 Background and related work

The widely used StoB converter is the counter-based technique, i.e., serial pop counter [110]. This converter works on the principle of cycle-based serial calculation to count the number of 1s in the bit stream. The latency of the operation increases exponentially with the Binary number (BN) length. So, to overcome this drawback, researchers use the parallel pop counter for StoB conversion. Due to the parallel operation of the partial sum creation using the full adders (FAs) latency of the computation is reduced enormously. In parallel PC, the latency has reduced dramatically compared to serial PC. However, the hardware requirement for internal partial sum calculation skyrocketed. Also, these increases in circuitry result in higher energy consumption and area overhead. Further, to overcome the overhead area issue, much research is moving toward the approximate parallel PC. This counter is a hybrid version of parallel PC, but it induces inaccuracy due to the approximation in the calculation. Also, the area overhead is high compared to serial PC [110]. There is a need for a novel StoB converter that should be area, and energy-efficient with reduced latency for in-memory computing-based deep learning applications.

Rate coded unary (Stochastic Number) and Transition coded unary

To clarify the concept of AGNI, we need to give a brief overview of the different unary representations. Due to the brevity, we are restricting the explanation to a unipolar format. Unary computing is broadly classified into rate-coded unary (see Fig. 6.1(a)) and transition-coded unary(see Fig. 6.1(b)). Here in rate coding representation, the information is contained in the frequency of an event. Rate coding is adopted in stochastic computing. The positioning of the 1's in the bit stream is random. Here in Fig. 6.1(a), the value is 4/8, i.e., 0.5. This probability means seeing one at any bit position is half (0.5).

Similarly, in a transition coded scheme where the timing relation of events contains information. Here the 1's and 0's are found in groups as shown in Fig. 6.1(b).M

Figure 6.1: (a) rate coded unary representation, (b) transition coded unary representation

Flash ADC with intermediate transition coded conversion

Fig. 6.2 shows a schematic of flash ADC with eight inputs. Thus the circuit has eight voltages comparators (i.e., $C_1, C_2, C_3, C_4, C_5, C_6, C_7, and\ C_8$) as shown in Fig. 6.2. The

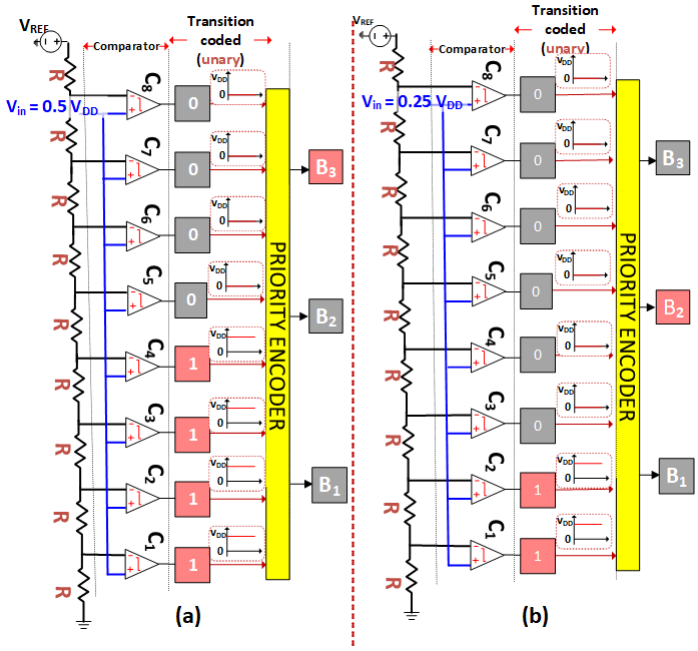

Figure 6.2: Flash ADC with a) Input $V_{in} = 0.5V_{DD}$, and b) Input $V_{in} = 0.25V_{DD}$.

positive terminals of all comparators are connected to the analog input V_{in}. The negative terminal of the comparator is connected to the V_{REF}. Now consider the scenario with inputs equal to 0.5 V_{DD}, i.e., the output detected by the priority encoder is binary digit four, as shown in Fig. 6.2(a). The above ADC highlights that intermediate data conversion is in rate coding format, as shown in Fig. 6.2.

Future, if we change the input analog value, the output binary value detected changes. For example, from the initial input (V_{in}) value of 0.5 V_{DD} , as in the scenario mentioned above, is changed to 0.25 V_{DD} , then the binary value detected is two as shown in Fig. 6.2(b).

6.4 Proposed technique to convert the rate coded unary to temporal coded unary

This section is useful in better understanding of AGNI working principle of StoB conversion. Here the input stochastic number is stored and then converted into analog equivalent voltage via ADC as shown in Fig. 6.3 using S_to_A peripheral unit (will be explained in Section 6.6). This equivalent analog voltage is converted to transition coded unary by utilsing the A_to_U peripheral unit (Section 6.6). Next step is the data extraction of the unary value to compute the number of 1's in bitstream by using priority encoder of U_to_B peripheral unit (Section 6.6). *AGNI's novel technique of converting the stochastic to transition coded unary number intermediate step helped by not using the performance in-efficient counter method.*

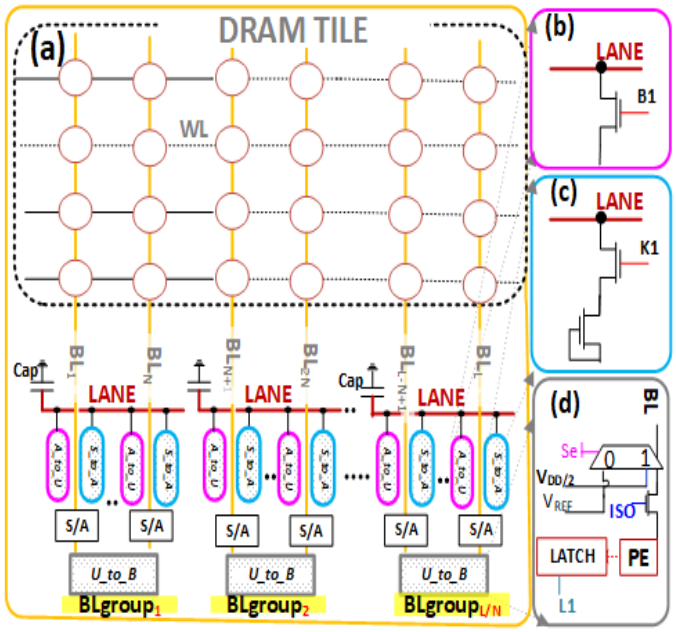

Figure 6.3: Schematic layout of AGNI substrate and employed peripherals. Illustration of (a) an AGNI-modified DRAM tile, (b) an A_to_U peripheral unit, (c) an S_to_A peripheral unit, and (d) a U_to_B peripheral unit.

6.5 Overview of Our AGNI Substrate

The purpose of our AGNI substrate is to enable in-situ conversion of input stochastic operands (bit-vectors) into binary numbers. To fulfill this purpose, the AGNI substrate employs a few modifications in the structure of each tile of a commodity DRAM module. These modifications are highlighted in Fig. 6.3(a). Evidently, our AGNI substrate logically groups the bitlines of each DRAM tile into multiple bitline groups (BLgroups). Each BLgroup corresponds to an input stochastic operand. Therefore, the number of bitlines in a BLgroup equals the number of bits in an input stochastic operand (i.e., the size of the input stochastic operand's bit-vector). Consequently, if the size of each input stochastic operand is N bits, and if each DRAM tile has a total of L bitlines (L is 256 or 512 typically), then each DRAM tile contains a total of L/N logical BLgroups, with each BLgroup having a total of N bitlines.

Further, to enable stochastic-to-binary number conversion of input operands, AGNI employs additional peripherals in each DRAM tile atop the already existing sense amplifiers (SAs). As shown in Fig. 6.3(a), these peripherals include per-bitline S_to_A units, per-bitline A_to_U units, per-BLgroup U_to_B units, and per-BLgroup analog lanes (see LANEs in the figure). Each LANE is horizontally laid out across a BLgroup and has a capacitor at the end. From Fig. 6.3(c), Each S_to_A unit contains two transistors (Fig. 6.3(c)), whereas each A_to_U unit contains one transistor (Fig. 6.3(b)). Each U_to_B unit contains one isolation transistor (ISO) per bitline (i.e., N ISOs per BLgroup) along with one priority encoder (PE) and a latch (Fig. 6.3(d)). All S_to_A and A_to_U units belonging to a BLgroup connect their corresponding bitlines and SAs to the corresponding LANE and analog lane

capacitor. The N S_to_A units of a BLgroup enable stochastic-to-analog number conversion; the N A_to_U units of the same BLgroup enable analog-to-unary (transition coded unary) number conversion; and the U_to_B unit of the same BLgroup enables unary-to-binary number conversion. Thus, the additional peripherals of AGNI enable one stochastic-to-analog-to-unary-to-binary number conversion per BLgroup, thereby enabling a total of L/N such conversions in-parallel per DRAM tile. The operation of our AGNI substrate that enables such conversions is discussed next.

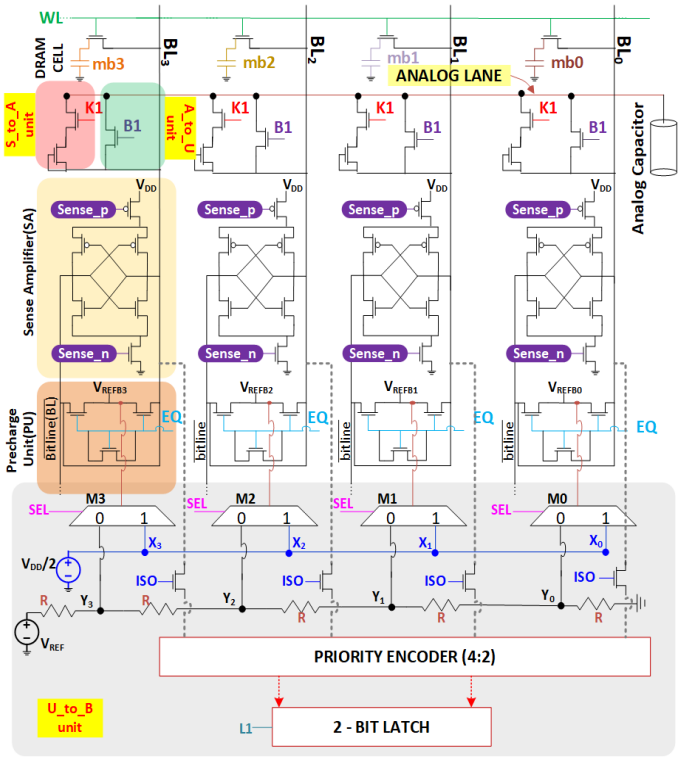

Figure 6.4: Schematic of AGNI StoB substrate with N = 4 and BN = 2. Consisting of peripheral structures S_to_A units, A_to_U units and U_to_B unit

6.6 Operation of Our AGNI Substrate

As implied from the previous section, AGNI substrate undertakes stochastic-to-binary conversion of input operands in the following three sequential steps: (i) stochastic to analog (S_to_A) conversion, (ii) analog to transition-coded unary (A_to_U) conversion, and (iii) transition-coded unary to binary (U_to_B) conversion. For these steps to work for an input stochastic operand, the operand needs to be read into the SAs of its corresponding BLgroup, which can be achieved by activating the DRAM row that contains the stochastic operand. Thus, a DRAM row activation must precede the above three steps, to constitute a sequence of a total of four steps for the operation of AGNI substrate for achieving stochastic-to-binary number conversion.

To realize these four steps, our AGNI substrate utilizes several timing signals. The timing signals required for the first step (i.e., DRAM row activation) include the standard

DRAM operation signals [111] [112]. The remaining three steps require additional new timing signals to control the added peripherals of the AGNI substrate. The definitions and exact uses of these signals are summarized in Table 6.1. These signals affect various hardware units of the AGNI substrate. This is illustrated in Fig. 6.4 for an example BLgroup of AGNI substrate with $N = 4$.

Table 6.1: Definitions and uses of various timing signals employed by AGNI substrate.

Standard DRAM Operation Signals	
WL	Signal to turn on a DRAM wordline to enable charge sharing between a row of DRAM cells and corresponding bitlines
sense_p **sense_n**	Complementary signals that are used with each SA to enable the sensing and amplification of the bitline voltage perturbation
EQ	Signal for the precharge unit to equalize the BL voltages
Newly Added Timing Signals	
K1	Signal to turn on S_to_A units to enable charge flow from the SAs of a BLgroup to the analog LANE capacitor
B1	Signal to turn on A_to_U units to enable charge flow from the analog LANE capacitor of a BLgroup to the bitlines
ISO	Signal to turn on/off the isolation transistors, to connect/disconnect the priority encoder from a BLgroup
SEL	Signal to the MUXEs that enables the selection of a SA reference voltage (V_{REF})
L1	Signal to enable the latch for the binary result

The BLgroup illustrated in Fig. 6.4 has 4 bitlines, i.e., $BL_0, BL_1, BL_2, and\ BL_3$. These bitlines correspond to four DRAM bit-cells, i.e., mb0, mb1, mb2, and mb3, respectively. From Fig. 6.4, each bitline is connected to a SA (highlighted in light yellow) and a pre-

Table 6.2: Toggle time stamps (↑ or ↓) for various timing signals to realize the four operational steps of our AGNI substrate.

Activate	0ns ($EQ\uparrow$) 5ns ($EQ\downarrow$) 7ns ($WL\uparrow$) 9ns ($sense\_n\uparrow$) 12ns ($WL\downarrow$)
S_to_A	13ns ($K1\uparrow$) 37ns ($K1\downarrow\ sense\_n\downarrow$)
A_to_U	38ns ($EQ\uparrow$)($SEL\downarrow$) 42ns ($EQ\downarrow$) 43ns ($B1\uparrow$) 45ns ($sense\_n\uparrow$)
U_to_B	45ns ($ISO\uparrow$) 51ns ($L1\uparrow$) 52ns ($L1\downarrow$) 55ns ($B1\downarrow\ ISO\downarrow$)

charge unit (highlighted in light orange). Additionally, each bitline connects to one S_to_A unit (highlighted in light red) and a A_to_U unit (highlighted in light green). Moreover, all four bitlines of the BLgroup (i.e., $BL_0, BL_1, BL_2, and\ BL_3$) connect to one U_to_B unit (highlighted in grey), which consists of one N:$\log_2 N$ priority encoder, one $\log_2 N$-bit latch, N isolation transistors (ISOs), one resistor ladder based voltage divider, and N multiplexers that select the V_{REF} values for corresponding SAs. A V_{REF} value is either $V_{DD}/2$ or an appropriate level from the voltage divider. The selection of V_{REF} values from the voltage divider enables the SAs to operate as voltage comparators that can provide analog-to-unary conversion (just like flash ADC; Fig. 6.2). In the following subsections, we explain how the toggling of various timing signals listed in Table 6.1 has to be AGNI signals orchestrated to realize the four operational steps of AGNI substrate.

The exact time stamps for the toggling of these signals are summarized in Table 6.2. The time evolution of these signals are also depicted in Fig. 6.5. Note that at the initialization, these signals are in the OFF state. The time evolution of these signals triggers the voltage levels corresponding to various DRAM structures (e.g., bitlines, analog capacitor, bit-cells) to evolve, which is also illustrated in Fig. 6.5. We have evaluated various timing and voltage signals depicted in Fig. 6.5 by modeling and simulating the circuit shown in Fig. 6.4 using LTSpice.

DRAM Row Activation (Step 1)

The DRAM row activation step employs EQ, WL, and sense_n (sense_p) signals to read the input stochastic operands into the SAs of their corresponding BLgroups. For this step, EQ is first toggled ON (to a higher voltage level in Fig. 6.5) at 0 ns. At 0 ns, we consider that SEL has been ON; therefore, at 0 ns, V_{REF} for the SAs has already been selected to be $V_{DD}/2$. As a result, the voltage on the bitlines swiftly settles to $V_{DD}/2$ (see the evolution of voltage on the bitlines in Fig. 6.5(d)). This step is conventionally known is bitline pre-charging. We are able to achieve swift bitline pre-charging in AGNI because we consider short bitline DRAM architecture with only 8 cells per bitline [111]. Then, EQ is toggled OFF at 5 ns. Then, at 7 ns, WL is toggled ON. As a result, the DRAM cells (see mb0, mb1, mb2, and mb3 in Fig. 6.4) connect to their respective bitlines (see BL_0, BL_1, BL_2, and BL_3 in Fig. 6.4) to start sharing their charge with the bitlines. Due to this charge sharing, the voltage on the DRAM cells dips (see Fig. 6.5(e) at 7 ns) and the voltage on the bitlines is perturbed (see Fig. 6.5(d) at 7 ns).

Then, at 9ns, sense_n (sense_p) is toggled ON (see Fig. 6.5(f)), which enables the SAs to sense the perturbed bitline voltage and amplify it to the full swing. In Fig. 6.5(d), since the bitline voltage perturbation is in the positive direction (corresponding to logic '1' stored in the DRAM cells), the bitline voltage is swung to V_{DD} by the SAs. After this full-swing amplification of bitline voltage perturbation, the SAs complete replenishing the DRAM cell voltage at 11 ns. The perturbation amplification and cell replenishing both occur swiftly because we consider the short bitline DRAM architecture for AGNI. Then, at 12 ns, WL is toggled oFF, to disconnect the DRAM cells from the bitlines, to mark the end of the DRAM row activation step.

Note, during this step, the bitline voltage evolution encounters transient noise at two events, due to parasitic effects. First, at 5 ns when EQ is toggled OFF. This event is labeled

as *glitch 1* in Fig. 6.5(d). Second, at 12 ns when WL is toggled OFF (labeled as *glitch 2* in Fig. 6.5(d)).

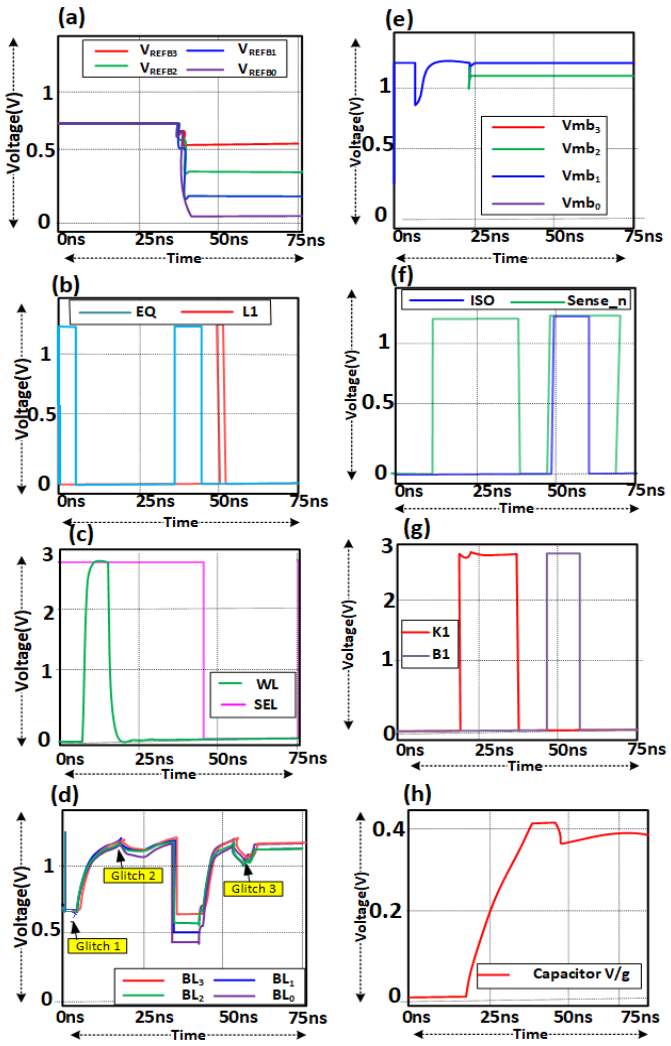

Figure 6.5: SPICE simulation for AGNI StoB substrate with N = 4. a)voltages of the precharge units (V_{REF}) b)Equalizer (EQ) and Latching signals(L1), c) Wordline (WL) and SEL signals, d)Bitline (BL) voltages e)DRAM cell capacitor voltage, f) sense_n and isolation signal, g) charging (K1) and discharging (B1), and h) capacitor voltage

S_to_A Conversion (Step 2)

At the end of the DRAM row activation *Step 1*, the values are latched into the SA. Now, S_to_A Conversion step employs sense_n (sense_p), and $K1$ signals to conduct the conversion of the read stochastic operands to analog voltage. In S_to_A peripheral unit consist of a pass transistor and diode i.e., back biased nmos (see Fig. 6.3(c). Due to this circuitry construction of the S_to_A unit, when K1 is toggled ON (to higher voltage in Fig. 6.5), SAs that stored with logic 1 are only connected to analog lane, and excluding the SAs with logic 0. Here $K1$ lasts for a brief time of 25ns,i.e. toggle ON at 13ns ,and toggle OFF

at 37ns (see Fig. 6.5(g). At the end of 37ns, the stored value at the analog capacitor is the equivalent to the total number of ones in the rate coded unary bit stream. To better explain this concept, consider Fig. 6.6, which shows the analog capacitor voltage for $N = 4$. Here if all the input bit streams are logic 1 state (i.e., DRAM cells mb0=mb1=mb2=mb3= logic 1), then the capacitor voltage captured at 52ns is 514mV. This voltage is also called the V_{MAX}. Table 6.3 shows the V_{MAX} for different BN lengths from 4 bits to 8 bits.

Further, consider in AGNI StoB substrate with BLgroup size four (i.e., $N = 4$), if all the DRAM cells are logic '0' expect one DRAM bitline cell is at logic '1' (i.e., mb3 = mb2 = mb1 = logic '0', mb0 = logic '1'). Then the equivalent voltage captured in the analog capacitor is 281mV. This Fig. 6.6 also shows that the charging sequence of the analog capacitor, which starts at 13ns ($K1$ = toggle ON) and ends at 37ns ($K1$ = toggle OFF). The timing sequence is Tabulated in Table 6.3 and the corresponding DRAM signals.

From Fig. 6.6 example with $N = 4$, we clearly see the four different levels of analog voltages for four combinations of DRAM cell logic pattern. Similarly, for any BLgroup of size N, AGNI's S_to_A produce N different analog voltage. Hence the stored value at the analog capacitor is the equivalent to the total number of ones in the rate coded unary bit stream. This is the main principle of our novel AGNI's S_to_A operation.

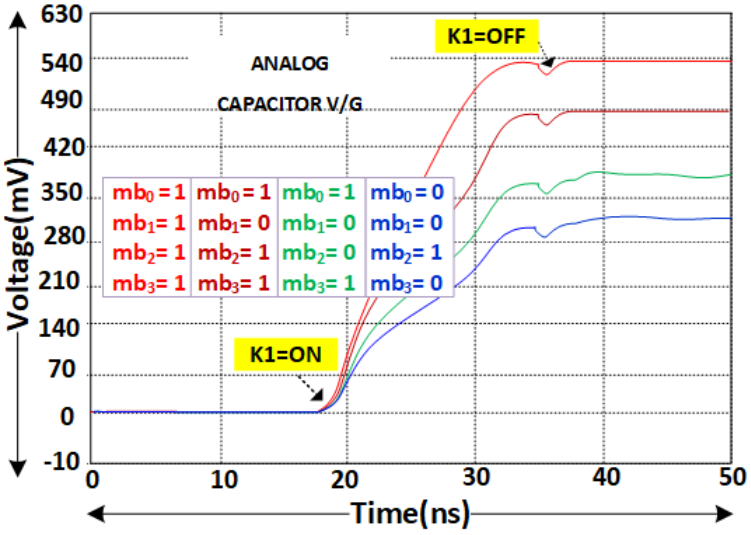

Figure 6.6: Analog capacitor voltage at different number of 1's for 4 bit SN

A_to_U Conversion (Step 3)

As implied in the above Section 6.6 S_to_A stage, value of the stochastic number (SN) is stored in the analog capacitor as equivalent analog voltage. Now, in A_to_U conversion step employs $EQ, SEL, B1$, and $sense\_n$ ($sense\_p$) to convert the analog voltage to transition unary for successfully reading the value by priority encounter (This is similar to as we explained in Section 6.3). Here AGNI initiate a new read operation sequence at 37ns with $K1$ being disabled (i.e., toggled OFF). Now the selection of the V_{REF} is from the resistance ladder (i.e., SEL toggle OFF, as shown in Fig. 6.5(c)). After the BL is equalized to the

Table 6.3: MAE, MAPE, RMSE, and V_{MAX} of AGNI at different BLgroups lengths (N)

BN length	N	**MAE**	**MAPE%**	**RMSE**	$V_{MAX}(mV)$
4 bits	16	0.28	3.58	0.41	630
5 bits	32	0.41	3.93	0.50	715
6 bits	64	0.37	1.58	1.03	735
7 bits	128	0.29	0.97	0.43	755
8 bits	256	0.20	0.59	0.35	785

corresponding V_{REF} (as shown in Fig. 6.5(a)) at 42ns. We initiate toggle ON of $sense\_n$ and $B1$ signals for discharging to the BL. This charge sharing between analog capacitor and BL is shown in Fig. 6.5(d). With the toggle ON of $B1$ signal, charge sharing initiation at 45ns a dip in analog capacitor voltage is observed (see Fig. 6.5(h)). Here SA perform the function of the comparator similar to as explained in the Section 6.3. Now the SAs will compare the voltages of analog capacitor (V_{ANG}) with reference voltages(V_{REF}) generated from the resistance ladder (i.e., V_1, V_2, V_3, and V_3 if N = 4) . In general, SA compares the analog voltage to N node voltages from the resistance ladder. Here, if the analog voltage is greater than the V_{REF}, the perturbation is +δV change in the BL voltage and treated as logic 1. Similarly, if the analog voltage V_{ANG} is less than the V_{REF}, the perturbation is -δV change in the BL voltage and treated as logic 0. The value perceived by the SA will be in the transition coded unary (see Section 6.3).

During this step, the BL also experiences a third voltage spike, due to switching $B1$ signal that leads to charge sharing between the BL and analog capacitor at 45ns. (labeled as *glitch 3* in Fig. 6.5(d)).

U_to_B Conversion (Step 4)

Here in this step the U_to_B employs $ISO, L1$, and $B1$ signals to perform the unary coded to binary conversion. The unary transition coded number in previous stage (i.e., A_to_U Section 6.6) at 47ns is converted to binary value by using the priority encoder (similar to the Section 6.3). Here we employs an isolation signal (ISO) to connect the priority encoder to the BL as shown in Fig. 6.5(f). The sequence of this U_to_B starts with ISO signal toggle ON (logic 1) at 45ns for a brief of 10ns and ends with toggle OFF at 55ns (see in Fig. 6.5(f)). AGNI wait for 7ns to avoid noise in priority encoder and avoid the inaccuracy. Now the latching sequence signal L1 is initiated for 1ns, i.e, toggle ON at 51ns and ends with toggle OFF at 52ns (see Fig. 6.5(b)). The binary info is latched into the $\log_2 N$ bit latch. At the end of 55ns with toggle OFF of signals (i.e., $B1, ISO$) is enforce.

Thus the full cycle of StoB is completed with four distinct steps (i.e, *Step 1 -Step 4*) within 55ns. Finally, at the end of 55ns, a new stochastic operand is ready to perform the StoB conversion.

The proposed AGNI design is re-configurable for any length of StoB based on the DRAM BLgroups (see Fig. 6.3).

6.7 Evaluation

Overhead estimation for the peripheral units

We modeled the proposed design AGNIs on 2D DDR4_512 DRAM organizations for 45nm technology node using CACTI [113]. Also, each DRAM cell consumes $3F^2$ area, while the horizontal pitch is 3F. Further, the height of SA , precharge unit, and write drivers consume *117F, 90F, and 27F* respectively [114]. Additionally, the height of the peripheral units of AGNI such as S_to_A, A_to_U and U_to_B consume *27F*, *27F*, and *110F* respectively. Therefore, the effective height of the proposed design AGNI design in 2D DDR4 DRAM tile comes out to be *1934F*.

In this Section, we also going to explain the area overhead estimate of the peripheral units such as charge pump circuits required for the node voltages to perform the AGNI StoB operations (i.e., A_to_U step, S_to_U step). For the analysis purpose, we can compute the node voltage generated from the resistance ladder via charge pump (CP) [115] [116]. Table 6.4 provides the area overhead involved in CP circuits, dynamic power dissipation, area of per charge pump (Acp), capacitance per CP, optimal stages in the CP (Nopt) referring to [115], and total wasted power per CP for different BLgroups (N). From the calculation, for N = 256 the total wasted power per CP is 6.85E-08mW. Similary, we have provided the values for rest of the BLgroups N = 16, 32, 64, and 128.

Table 6.4: Charge pump area and power dissipation

N	Nopt	Ctotal per CP(fF)	Acp (um2)	Dynamic power per CP (W)	total wasted power per CP (mW)
16	2	0.017	0.0087	1.30E-09	3.91E-09
32	2	0.037	0.0186	2.74E-09	8.22E-09
64	2	0.076	0.038	5.55E-09	1.67E-08
128	2	0.154	0.077	1.12E-08	3.37E-08
256	2	0.316	0.158	2.28E-08	6.85E-08

Evaluation Setup

AGNI design is implemented at $45\eta m$ gpdk technology node and LTSPICE as the SPICE simulator to perform the transient AC analysis of our proposed StoB conversion. We implemented the AGNI substrate for five BLgroups, i.e., with N = 16, 32, 64, 128, and 256 bits corresponding to binary operand lengths of 4, 5, 6, 7, and 8 bits respectively. Our analysis is considered with DRAM bitlines (L) of size 512-bitlines. We employed the brute-force method to consider all the combinations of the stochastic operands. Then we compute the error introduced in the Stochastic to binary conversion with AGNI substrate. Also, Table 6.5 provides the mean absolute error (MAE) (Eqn. 6.1), Mean absolute percentage error (MAPE) (Eqn. 6.2), and Root mean square error (RMSE) (see Eqn. 6.3). Here, in the below Eqn. (6.1 - 6.3), y_i is the predicted value, x_i is the actual value, and n is the total number of data points. Along with this above mentioned error parameters, we also

tabulated the V_{MAX} for five different BLgroups, (i.e., for N = 16, 32, 64, 128, and 256 bits) from SPICE simulation.

To analyze the performance of AGNI StoB in a DRAM converter, we are comparing it with two in-situ popcounters (PCs), i.e., parallel pop counter and serial pop counter. Further, the values tabulated in Table 6.5 are considered for in-memory computing accelerators with a parallel pop counter for StoB conversion, used similar to SCOPE [3]. Likewise, the serial counter's latency value is considered for in-memory computing such as ATRIA [4]. Table III shows the area, Energy delay product (EDP), and $area \times latency$ for the considered prior works and AGNI.

$$MAE = (\sum_{i=1}^{n} |y_i - x_i)/n) \tag{6.1}$$

$$MAPE = \left(\frac{1}{n}\right) \sum_{i=1}^{n} \left|\left(\frac{x_i - y_i}{x_i}\right)\right| \tag{6.2}$$

$$RMSE = \sqrt{\left(\frac{\sum_{i=1}^{n}(y_i - x_i)^2}{n}\right)} \tag{6.3}$$

Results and Discussions

The results show that the mean absolute error of AGNI substrate at higher binary operand (BN) lengths are better than lower bits. The reason for this degradation in the lower bit is due to the effects of parasitic capacitance. Hence, the MAE of the AGNI substrate with a binary operand length of 4 bits is 0.28 higher than the MAE of 0.2 at 8-bit BN length, as shown in Table 6.3.

Similarly, the mean absolute percentage error (MAPE) of AGNI substrate is 3.58% and 0.59% at 4-bit binary operand length and 8 bits, respectively. Also, as shown in Table 6.3, the RMSE values at 4 bits and 8 bits BN lengths are 0.41 and 0.35.

From simulation results, our proposed design AGNI outperformed parallel PC in terms of area and energy savings. Our design demonstrated an area and EDP saving of 390× and 28×, respectively, at 4 bits binary length over parallel PC, as shown in Table 6.5. Further, AGNI showed significant EDP and area savings with higher BN lengths. This improvement in our proposed design comparison to parallel PC is the exponential requirement of full adders components for partial summing [117]. For example, area saving and EDP saving of 923× and 350×, respectively, are recorded for 8-bit binary operand length. Parallel PC has a slight edge on the latency over AGNI, but our proposed design saves significant EDP and area saving justifies this increase in latency.

Similarly, we compared our novel design with a serial PC. The simulation result shows enormous savings in EDP. Due to the factor of the serial counter being cycle based, we observed our designs gained EDP savings of 59× and 930× at 4 bits and 8 bits BN lengths respectively. Further, AGNI also has better area saving over serial PC. From the simulation, we observe 8× and 96× at 4 bits and 8 bits BN lengths, respectively.

Further, from Table 6.5, AGNI has an $area \times latency$ saving of 21× and 23× w.r.t. Parallel PC and serial PC, respectively, at 4 bits binary operand length. Similarly, at higher

Table 6.5: Comparison of EDP, and AREA of prior StoB designs (Parallel PC [3], Serial PC [4]), and AGNI

BN operand Length	**Designs**	**Area** (mm^2)	**EDP** ($ns.pJ$)	**Area.latency** ($mm^2.ns$)
8	**Parallel PC**	24.180	7440.00	483.60
	Serial PC	2.560	19660.80	655.36
	AGNI	0.026	21.23	1.95
7	**Parallel PC**	11.700	2880.00	187.20
	Serial PC	1.280	4915.20	163.84
	AGNI	0.013	10.25	0.95
6	**Parallel PC**	5.460	1008.00	65.52
	Serial PC	0.640	1228.80	40.96
	AGNI	0.007	5.03	0.47
5	**Parallel PC**	2.340	288.00	18.72
	Serial PC	0.320	307.20	10.24
	AGNI	0.003	2.54	0.23
4	**Parallel PC**	0.780	36.00	2.34
	Serial PC	0.160	76.80	2.56
	AGNI	0.002	1.28	0.11

BN length, the AGNI's $area \times latency$ significantly increases. i.e., 247× and 333× over parallel and serial PC, respectively. Overall, our design has outperformed parallel and serial PC in terms of area and energy.

For further reference, the simulation files of the AGNI substrate are provided in the following link AGNI simulations link.

6.8 Conclusion and future scope

In this chapter, we have proposed a novel in-situ StoB converter AGNI for Deep learning applications by leveraging the DRAM circuitry. The design uses a novel technique of converting the input stochastic bit stream into an equivalent analog voltage. This analog voltage is fed into the flash ADC to convert to a unary number. Next, a unary value is sensed by the priority encoder to extract the binary value. Converting stochastic to unary mitigate in AGNI, the use of area in-efficient counter-based technique as, in prior works [4] is mitigated. The results illustrate that the area overhead for peripheral units such as S_to_A, A_to_U, and U_to_B is minimal i.e., 4% for 8 bits BN length. AGNI has a significant EDP saving of 350× over parallel PC. These results corroborate the excellent capabilities of AGNI for accelerating the stochastic in-memory accelerator for deep CNNs.

This chapter can be used as a pathfinder to further enhance the StoB conversion, with even better latency and EDP saving. There is a lot of scope for improvising the timing and unary conversion method. This research can even be pushed to a deeper sub-nano meter technology node to analyze the performance.

Chapter 7 Conclusions

In summary, the report highlights the significant computational and memory requirements of Convolutional Neural Networks (CNNs) and the challenges they pose for traditional von-Neumann computing architectures. To address these challenges, the report proposes the Processing-In Memory (PIM) technique, which utilizes different memory technologies such as DRAM and PCM, with Stochastic arithmetic and minimal add-on logic to reduce the hardware requirements for CNN's arithmetic operations.

The report presents designs for scalable Stochastic Number Generator (SNG), DRAM CNN accelerator, non-volatile memory (NVM) class PCRAM-based CNN accelerator, and DRAM-based stochastic to binary conversion (StoB) for in-situ deep learning. These designs utilize stochastic computing to reduce the hardware requirements for CNN's arithmetic operations and enable energy and time-efficient processing of CNNs.

The report also identifies future research directions for the proposed designs, including in-situ PCRAM-based SNG, ODIN, ATRIA, and AGNI, and presents initial findings for these ideas. The proposed solution has the potential to significantly improve the energy and time efficiency of CNNs, providing a promising path for the future of CNN-based applications.

In conclusion, the report offers a comprehensive approach to address the challenges of processing CNNs, and the proposed designs have opened up new research directions and possibilities for the development of efficient and scalable CNN processing systems. Overall, the proposed solution has the potential to enable the development of reliable PIM-based systems with minimal hardware modifications and design complexity, providing a promising path for the future of CNN-based applications.

Chapter 8 Future Direction 1: Optimized GDAC-based SNG, Analysis and Mitigation of the Impacts of PCRAM Reliability Issues

8.1 Introduction

Our objective is to enhance the performance of the GDAC-SNG [118] and improve its area and energy efficiency. Furthermore, we analyze the impact of reliability issues on PCRAM-based SNGs and plan to propose mitigation techniques for these problems in future work.

Generating a random bitstream to represent an N-bit binary number in the SC domain using an LFSR-based SNG is not suitable for stochastic PIM-based CNN accelerators. Thus, we are exploring PCRAM-based SNGs that leverage the inherent stochastic property of phase change memory cells. However, resistance drift is a reliability issue that affects PCRAM cells' performance, as shown in Figure 8.1. We are conducting a thorough study to understand the impact of these issues on our proposed PCRAM-based SNG (CAPSTONE).

To address these problems, we are designing a new SNG based on the concept of the GDAC-based SNG [119] that offers better area optimization and lower latencies, as shown in Figure 8.2.

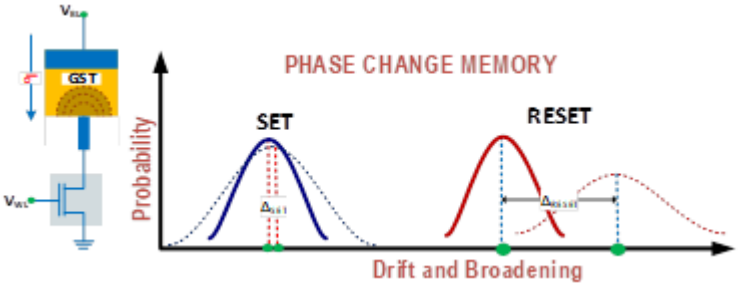

Figure 8.1: Resistance Drift and broadening in the PCRAM row for SET and RESET state.

In the Stochastic computing (SC) domain, the circuit's correlation and precision error are critical metrics to measure Stochastic circuits' performance as discussed in Chapter 6. Thus, in this work, we aim to make a detailed analysis of the correlation and precision error calculation, with and without the impact of physical variations, i.e., PCM drift resistance in the GST material, which depends on the drift coefficient relevant to time and temperature. Later, in this chapter, we also intended to introduce a Section on the mitigation technique to neutralize the performance degradation impact on proposed SNGs due to PCM drift resistance. Below are the key problems in the GDAC-SNG that we would like to solve.

8.2 Problems in the existing OPAMP-based SNG

The development of Phase Change Random Access Memory (PCRAM) technology presents certain challenges that need to be addressed. Firstly, there is a need to reduce the area and latency of operational amplifier (OPAMP) based SNGs. Secondly, correlation

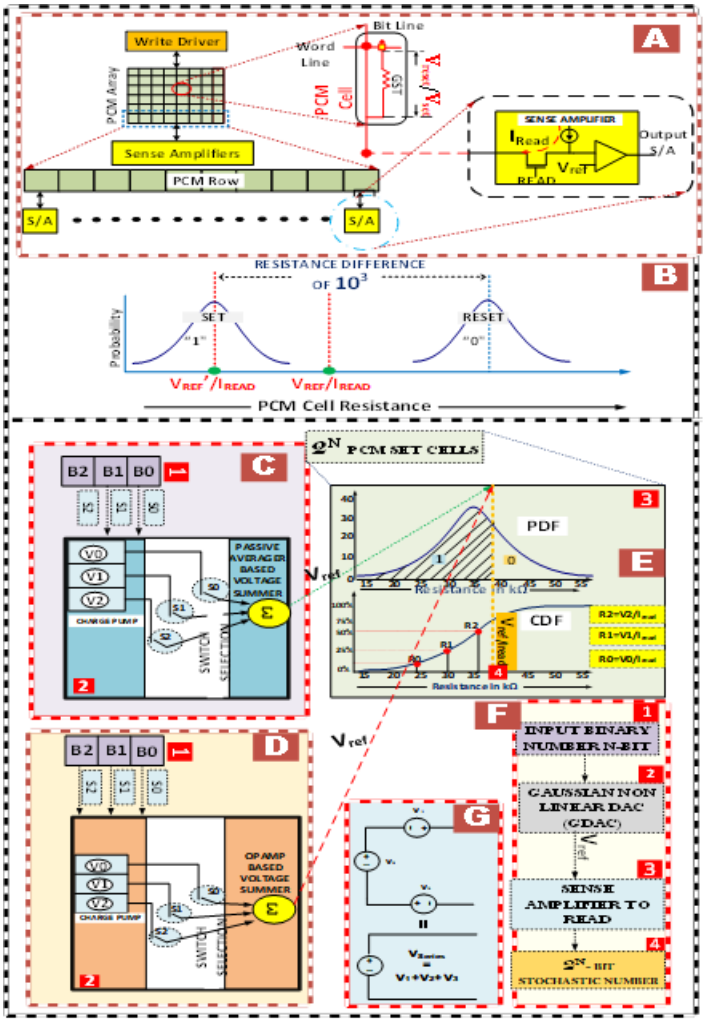

Figure 8.2: Illustration of (A) PCRAM cell array, and (B) PCRAM resistance distributions for SET and RESET cells. (C) block diagram of CAPSTONE PCRAM-based GDAC SNG without opamp-summer circuit. (D) block diagram of PCRAM-based GDAC SNG [11] (E) PDF and CDF voltage mapping of 2^N PCRAM cells. (F) flow chart of the CAPSTONE (G) passive averager.

errors need to be addressed to improve the accuracy of PCRAM. Lastly, precision errors due to PCM resistance drift need to be mitigated to ensure the stable and reliable performance of the technology. These challenges require further research and development in the field of PCRAM to overcome and improve the functionality of the technology.

8.3 Key ideas/Approaches

To address the correlation issue, we plan to exploit the properties of PCRAM cells to manipulate the correlation of the generated stochastic numbers(as shown in Figure 8.2). Additionally, we aim to use the voltage drift method to solve the correlation error issue of the PCRAM-based SNG (as shown in Figure 8.3). These proposed solutions require further research and development to improve the functionality and reliability of the PCRAM-based

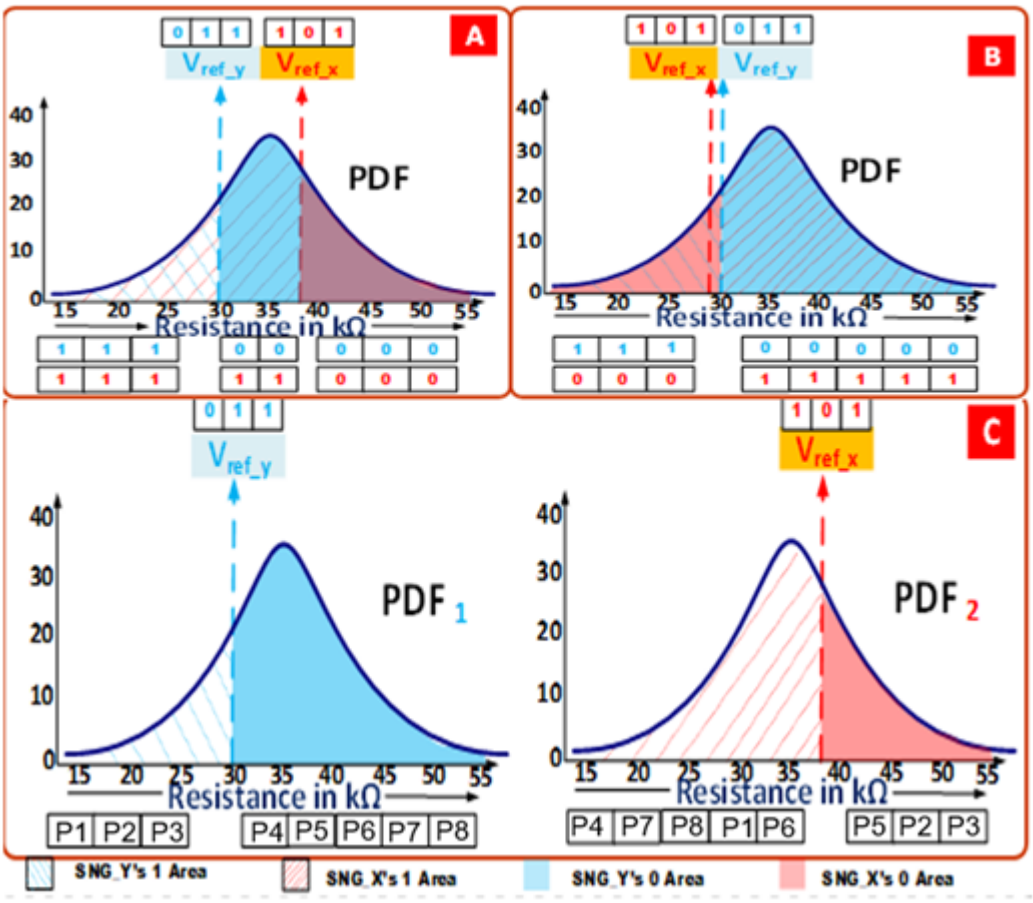

Figure 8.3: Correlation Manipulation: (A) positive correlation manipulation, (B)Negative correlation, and (C) uncorrelated.

SNG.

8.4 Goals of this work

The objectives of this work are multi-fold. Firstly, we aim to design a PCRAM-based SNG called CAPSTONE that is energy-efficient and has a small area footprint. CAPSTONE will incorporate a passive averager-based summer circuit. Secondly, we plan to analyze the impact of physical variations in PCRAM cells on the performance of CAPSTONE, particularly resistance drift under different temperature and time conditions. We will also study the correlation and precision error of CAPSTONE and explore mitigation techniques for corresponding machine-learning applications. Finally, we will compare CAPSTONE's performance with that of other state-of-the-art SNGs such as Halton, VDC (Sobol), and LFSR [8] [120] [121].

Chapter 9 Future direction 2: PSCA – Stochastic Computing based In-PCRAM Accelerator for CNN Processing

In this work, we are reusing the framework of our previous work ODIN [118] for CNN acceleration with minimal reads and write operations. In our previous work ODIN [118], the proposed system has increased the speed of operation by 90.8× over ISAAC crossbar-based state-of-the-art accelerator. However, there are many reads and writes for a single MAC operation in the ODIN framework, which will impact the endurance of the PCRAM cells.

9.1 Motivation

PCRAM has the potential to enable high-performance, in-situ CNN acceleration using stochastic computing. However, the existing ODIN-based PCRAM CNN accelerator suffers from the drawback of requiring multiple read and write operations for a single MAC operation, leading to increased latency. To overcome this limitation, we propose PSCA - a novel Stochastic Computing-Based In-PCRAM Computing Accelerator for CNN Processing. PSCA leverages the bit-parallel accumulation technique introduced in our previous work, ATRIA [4], to significantly reduce latency for MAC operations. Specifically, PSCA uses the MUX available at the PCRAM memory bank level to perform multiplication and accumulation within just two memory operation cycles (MOCs) for 32 stochastic numbers, each with a 256-bit length.

9.2 Challenges

PCRAM, as a memory device, requires larger sense amplifiers (SAs) when compared to charge-based memory devices such as DRAM. Because of this requirement for larger SAs, a single SA is shared by multiple PCRAM bit lines, typically 32-bit lines, which results in a shortage of resources in the PCRAM subarray for computing. This is a significant issue that must be addressed when designing PCRAM-based systems. Additionally, there are latency issues associated with PCRAM when compared to DRAM. The read and write memory operation times are larger in PCRAM cells than in DRAM cells, which can negatively impact system performance. It is essential to consider these factors when designing and optimizing systems that rely on PCRAM as their primary memory device.

9.3 Idea

Below are the steps to follow for the implementation of our proposed design in the PCRAM-based memory chip:

- The memory chip capacity we considered is 8 Gb (1 GB) capacity, 1 rank, 8 chips per rank, 8 banks per chip, and 64 subarrays per bank.

- Thus, the total number of processing engines will be 4096 PEs, which is equal to 66 mm^2 with add-on logic overheads.
- We are considering a small tile size with only 256 cells per Bitline because T_{RCD} will be reduced.
- MUX is already a part of PCRAM, but it is shared among all partitions, so no PALP is available for computing, only for data movement.
- The overhead for the proposed framework would be to add 512 4-bit random select codes for 512 MUXs and 512 4-bit latches.
- The memory controller changes are as follows:
  - Hierarchical controllers constitute subarray-level controllers, bank-level controllers, and rank-level controllers.
  - Subarray-level controllers govern intra-subarray data movement and in-RAM and in-peripherals feature extraction computing.
  - Bank-level controllers govern inter-subarray data movement.
  - Each subarray level controller contains a feature extraction block (FEB) that implements S-to-B, pooling, activation, and B-to-S, as well as input and output queues and FSMs for time control.
  - Bank-level controllers implement subarray-wise input-output queues and FSMs for control.
- The Stochastic to binary conversion (S-to-B) and binary to Stochastic conversion (B-to-S) of all parameters are summarized below:
  - BtoS: All parameters (weights and activation values) can be stored in memory in the Stochastic domain to begin with, so at the beginning, store all weights and input activation values in the Stochastic domain.
  - StoB: For each output activation value, the corresponding Stochastic inner product result gets converted to binary, then pooling and activation are carried out. It is then converted back to Stochastic and passed on to the bank-level controller.

9.4 Aim of this work

The aim of this work is summarized as follows: -

- We are going to design the MUX-based bit-parallel Stochastic accumulation approach to perform 32 MAC operations within 5 MOCs (memory operation cycles).
- Compare PSCA with the state-of-the-art Stochastic computing-based accelerator SCOPE [3], Binary variant of DRAM-based PIM accelerator DRISA [108], and NVM based accelerator PRIME [106] on selected CNN applications.

- Later, we are planning to include the endurance calculation and mitigation technique of PSCA for different memory configurations.
- Finally, we intend to analyze the accuracy of PSCA with respect to existing CNN accelerators such as SCOPE_H2D/SCOPE_VANILLA [3].

Chapter 10 Future Direction 3: In-DRAM BtoS Conversion with Variable Parametric Precision for Improving the Performance of CNN Applications

10.1 Motivation

Not all CNN applications require the same parametric precision, and different features of a CNN model may have different precision requirements. As a first step towards in-DRAM B-to-S conversion for CNN applications with parametric precision requirements, we propose future work. Due to the progression of precision, even using only the 4 to 6 most significant bits (MSBs) of an input binary number to determine the target T_{RCD} for B-to-S conversion is feasible. From the initial observation, accuracy degradation of the proposed design by considering the significant 4 to 6 MSBs is tolerable, especially for inherently error-tolerant applications such as CNNs, achieving up to 93.75% to 98.43% accuracy in the converted stochastic numbers (SNs).

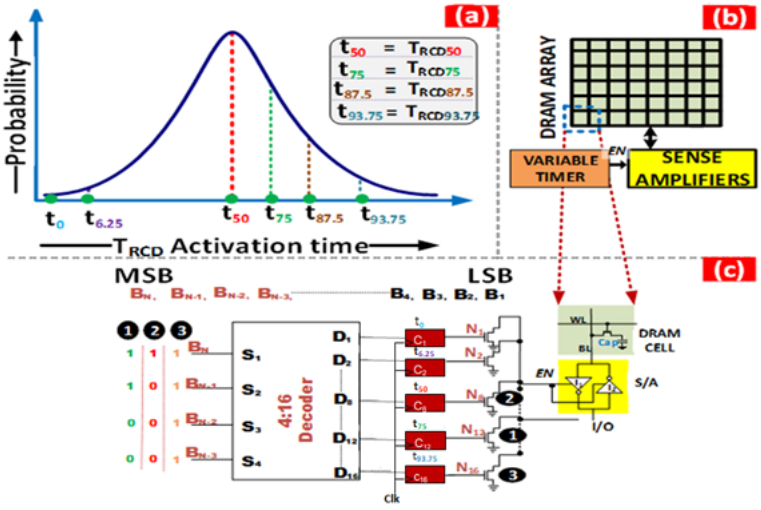

Figure 10.1: (a) Gaussian distribution of DRAM T_{RCD} activation time, b) DRAM peripheral connection with addon logic variable timer, (c) enlarged portion of the variable timer with enabled connection to the sense amplifier.

10.2 Idea

Below are the key highlights of the proposed ideas.

- The working principle is the same as GDAC-SNG but, here the normal distribution of T_{RCD} timing parameter of DRAM cells is considered instead of the PCRAM cell resistance. Figure 10.1(a) shows the T_{RCD} probabilistic distribution (PDF) of this DRAM cells row.
- To explain, consider binary number (B) from $B_N to B_1$ as shown in Figure 10.1(c), where we considered only the starting 4 most-significant bits for the GDAC compu-

tation (i.e., BN to BN-2), and rest of bits are ignored from the T_{RCD} delay calculation (i.e., BN-3 to B1).

- To give a better understanding, consider a 9-bit binary input (B), i.e., N=9, and with the proposed design idea, it considers only B9 to B6 for B-to-S computation. For this 9-bit binary number to be represented in the stochastic domain successfully without precision error, we need 2^9 DRAM cells (i.e., 512 cells). Consider Figure 10.1(a), which represents the Gaussian distribution of 512 cells in a DRAM row with respect to the T_{RCD} activation parameter.

- If the B_9 binary bit value is 1, then 50% of the DRAM cells are read as logic '1' (i.e., 256 out of 512 DRAM cells). Similarly, if B9 and B8 are 1's, then 75% of the DRAM cells are read as logic '1' (i.e., 384 cells out of 512 cells).

- Likewise, if $B_9, B_8, B_7, and B_6$ values are 1's, then 93.75% of the DRAM cells are read as logic '1' (i.e., 256 out of 512 DRAM cells).

- Thus, we can have 93.75% precision accuracy for the proposed SNG with just 4 bits into consideration for B-to-S computation.

- The only add-on logic of the proposed design is the variable timer circuit in the DRAM cell array, as shown in Figure 10.1(b).

- The arrangement of the variable timer circuitry is as shown in Figure 10.1(c), with 16 saturation counters (i.e., $C_1 to C_1 6$) and pull-down transistors (i.e., N1 to N16). These pulldown transistors' output corresponds to 16 T_{RCD} timing values from t0 to $t_{93.75}$, as shown in Figure 10.1(a).

- Here in Figure 10.1(c), the output enables signal (EN) from the 4:16 decoder is connected to the enable signal of the sense amplifier, which corresponds to the T_{RCD} timing, causing the sense amplifier (SA) to be active.

- Upon initial observation, it appears that the proposed design's accuracy degradation when considering only the significant 4 to 6 MSBs is acceptable, particularly for applications that are inherently error-tolerant, such as CNNs.

Bibliography

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *IEEE JPROC*, pp. 2295–2329, 2017.

[2] R. Zunino and P. Gastaldo, "Analog implementation of the softmax function," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2, 2002, pp. II–II.

[3] S. Li *et al.*, "Scope: A stochastic computing engine for dram-based in-situ accelerator," in *MICRO*, 2018, pp. 696–709.

[4] S. M. Shivanandamurthy *et al.*, "Atria: A bit-parallel stochastic arithmetic based accelerator for in-dram cnn processing," in *IEEE ISVLSI*, 2021, pp. 200–205.

[5] C. Szegedy *et al.*, "Going deeper with convolutions," *in CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[6] Chandra *et al.*, "Low power high performance priority encoder using 2d-array to 3d-array conversion," *Procedia Computer Science*, vol. 171, pp. 1037–1045, 2020.

[7] M. K. Qureshi *et al.*, "Scalable high-performance main memory system using phase-change memory technology," in *SIGARCH*. ACM, 2009.

[8] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, p. 92, 2013.

[9] 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," in *ACM ASPLOS*, 2017.

[10] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," *ACM ISCA*, pp. 383–396, 2017.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 1097–1105.

[12] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, and N. Sun, "Dadiannao: A machine-learning supercomputer," *IEEE MICRO*, pp. 609–622, 2014.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE CVPR*, 2015, pp. 1–9.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016, pp. 770–778.

[16] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization*, vol. 14, no. 2, pp. 1–25, 2017.

[17] Q. Lou, C. Pan, J. McGuinness, A. Horvath, A. Naeemi, M. Niemier, and X. S. Hu, "A mixed signal architecture for convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 15, no. 2, pp. 1–26, 2019.

[18] B. Zamanlooy and M. Mirhassani, "Efficient VLSI implementation of neural networks with hyperbolic tangent activation function," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 1, pp. 39–48, 2013.

[19] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "Dracc: A DRAM based accelerator for accurate CNN inference," in *IEEE/ACM Design Automation Conference (DAC)*, 2018, pp. 1–6.

[20] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, vol. 44, no. 3, 2016, pp. 27–39.

[21] X. Sun, S. Yin, X. Peng, R. Liu, J.-s. Seo, and S. Yu, "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," in *IEEE/ACM Design Automation and Test in Europe (DATE)*, 2018, pp. 1423–1428.

[22] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, vol. 44, no. 3, 2016, pp. 243–254.

[23] B. Jacob, D. Wang, and S. Ng, *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2010.

[24] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *IEEE TECS*, pp. 1–19, 2013.

[25] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, and M. A. Kozuch, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," *IEEE MICRO*, pp. 185–197, 2013.

[26] 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 *IEEE HPCA*, 2016, pp. 568–580.

[27] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang, and B. Yuan, "Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *IEEE ICCD*, 2016, pp. 678–681.

[28] I. G. Thakkar and S. Pasricha, "3d-prowiz: An energy-efficient and optically-interfaced 3d dram architecture with reduced data access overhead," *IEEE TMSCS*, vol. 1, no. 3, pp. 168–184, 2015.

[29] S. Li *et al.*, "Scope: A stochastic computing engine for dram-based in-situ accelerator," in *Proc. MICRO*, 2018.

[30] K. Kim *et al.*, "An energy-efficient random number generator for stochastic circuits," in *Proc. ASPDAC*, 2016.

[31] I. G. Thakkar *et al.*, "Dyphase: A dynamic phase change memory architecture with symmetric write latency and restorable endurance," *IEEE TCAD*, 2017.

[32] G. W. Burr *et al.*, "The inner workings of phase change memory: Lessons from prototype pcm devices," in *IEEE Globecom*, 2010.

[33] L. Jiang *et al.*, "A low power and reliable charge pump design for phase change memories," in *Proc. ISCA*, 2014.

[34] H. F. Chen *et al.*, "Emat: An efficient multi-task architecture for transfer learning using reram," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, p. 1.

[35] L. Chang *et al.*, "Corn: In-buffer computing for binary neural network," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.

[36] H. Mao, "Lergan: A zero-free low data movement and pim-based gan architecture," in *Microarchitecture (MICRO) 2018 51st Annual IEEE/ACM International Symposium on*. IEEE, 2018, pp. 669–681.

[37] H. Zhang, "Stateful reconfigurable logic via a single-voltage-gated spin hall-effect driven magnetic tunnel junction in a spintronic memory," *IEEE Transactions on Electron Devices*, vol. 64, no. 10, pp. 4295–4301, 2017.

[38] L. Wang, "Voltage-controlled magnetic tunnel junctions for processing-in-memory implementation," *IEEE Electron Device Letters*, vol. 39, no. 3, pp. 440–443, 2018.

[39] J. Yu, J. Wu, J. Guo, G. Li, J. Li, L. Li, and Q. Li, "Memristive devices for computation-in-memory," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, 2018, pp. 1646–1651.

[40] S. Angizi, M. Daneshtalab, and P. Liljeberg, "Pima-logic: A novel processing-in-memory architecture for highly flexible and energy-efficient logic computation," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[41] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM ISCA*, vol. 44, no. 3, pp. 14–26, 2017.

[42] B. Zamanlooy, M. J. Abdollahi Azgomi, and S. Sasanian, "Efficient vlsi implementation of neural networks with hyperbolic tangent activation function," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 1, pp. 39–48, 2013.

[43] J. Yue, "Aeris: area/energy-efficient 1t2r reram based processing-in-memory neural network system-on-a-chip," p. 146, 2019.

[44] D. Fan and S. Angizi, "Energy efficient in-memory binary deep neural network accelerator with dual-mode sot-mram," in *IEEE International Conference on Computer Design (ICCD)*, 2017.

[45] T. Sun, "Fbl (flexible block-wise loading) algorithm for effective resource allocation in ofdma systems with reduced uplink feedback information," in *IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications*, 2008.

[46] A. Agrawal *et al.*, "X-sram: Enabling in-memory boolean computations in cmos static random-access memories," *Circuits and Systems I: Regular Papers IEEE Transactions on*, vol. 65, no. 12, pp. 4219–4232, 2018.

[47] D. Nguyen *et al.*, "Memristive devices for computing: Beyond cmos and beyond von neumann," in *Very Large-Scale Integration (VLSI-SoC) 2017 IFIP/IEEE International Conference on*, 2017, pp. 1–10.

[48] S. Li *et al.*, "Pinatubo: A processing in-memory architecture for bulk bitwise operations in emerging nonvolatile memories," in *DAC*, 2016.

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

[50] S. Li *et al.*, "Pinatubo: A processing in-memory architecture for bulk bitwise operations in emerging nonvolatile memories," in *DAC*, 2016.

[51] M. Nourazar *et al.*, "Code acceleration using memristor-based approximate matrix multiplier: Application to convolutional neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2684–2695, 2018.

[52] L. Chang, "Dasm: Data-streaming-based computing in nonvolatile memory architecture for embedded system," *Very Large-Scale Integration (VLSI) Systems IEEE Transactions on*, vol. 27, no. 9, pp. 2684–2695, 2018.

[53] W. Kang and et al., "In-memory processing paradigm for bitwise logic operations in stt–mram," *IEEE Transactions on Magnetics*, vol. 53, no. 11, pp. 1–4, 2017, art no. 6202404.

[54] K. Shin, J.-Y. Park, M. Kim, S. Lee, and T. Kim, "Mcdram: Low latency and energy-efficient matrix computations in dram," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2613–2622, 2018.

[55] Q. Deng, W. Jiang, J. Cong, X. Qian, and B. Liu, "Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator," in *Design Automation Conference (DAC) 2019 56th ACM/IEEE*, 2019, pp. 1–6.

[56] C. Eckert, P. Chi, R. Diamant, A. Gao, and D. Yang, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *ISCA*, 2018.

[57] S. Gupta, S. Venkataramani, and K. Roy, "Nnpim: A processing in-memory architecture for neural network acceleration," *Computers IEEE Transactions on*, vol. 68, no. 9, pp. 1325–1337, 2019.

[58] Q. Den *et al.*, "Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator," in *IEEE DAC*, 2019, pp. 1–6.

[59] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," *IEEE MICRO*, pp. 609–622, 2014.

[60] L. N. Smith and N. Topin, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[61] S. Gupta, S. Venkataramani, and K. Roy, "Rapid: A reram processing in-memory architecture for dna sequence alignment," in *Low Power Electronics and Design (ISLPED) 2019 IEEE/ACM International Symposium on*, 2019, pp. 1–6.

[62] Chen, "Dadiannao: A machine-learning supercomputer," in *IEEE MICRO*, 2014, pp. 609–622.

[63] Liu *et al.*, "Scaling analysis of nanowire phase change memory," *IEEE Electron Device Letters*, vol. 32, no. 9, pp. 1174–1189, 2011.

[64] S. Angizi and et al., "Rimpa: A new reconfigurable dual-mode in-memory processing architecture with spin hall effect-driven domain wall motion device," in *IEEE Symposium on VLSI*, 2017.

[65] W. Shen and et al., "Stateful logic operations in one-transistor-one-resistor resistive random access memory array," *IEEE Electron Device Letters*, vol. 40, no. 9, pp. 1538–1541, 2019.

[66] L. Xie, "A mapping methodology of boolean logic circuits on memristor crossbar," *Computer-Aided Design of Integrated Circuits and Systems IEEE Transactions on*, vol. 37, no. 2, pp. 311–323, 2018.

[67] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-tile 2.4-mb in-memory-computing cnn accelerator employing charge-domain compute," *IEEE Journal of Solid-State Circuits*, 2019.

[68] L. Jiang, "Xnor-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 drams," in *Low Power Electronics and Design (ISLPED 2017 IEEE/ACM International Symposium on*, 2017, pp. 1–6.

[69] S. Angizi, "Dima: A depthwise cnn in-memory accelerator," in *Computer-Aided Design (ICCAD) 2018 IEEE/ACM International Conference on*, 2018, pp. 1–8.

[70] J. Sim *et al.*, "Lupis: Latch-up based ultra-efficient processing in-memory system," in *Quality Electronic Design (ISQED) 2018 19th International Symposium on*, 2018, pp. 55–60.

[71] D. Reis *et al.*, "A computing-in-memory engine for searching on homomorphically encrypted data," *Exploratory Solid-State Computational Devices and Circuits IEEE Journal on*, vol. 5, no. 2, pp. 123–131, 2019.

[72] Madhavan *et al.*, "High-throughput pattern matching with cmol fpga circuits: Case for logic-in-memory computing." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 26, pp. 2759–2752, 2018.

[73] H. Mao *et al.*, "Lergan: A zero-free low data movement and pim-based gan architecture," in *Microarchitecture (MICRO) 2018 51st Annual IEEE/ACM International Symposium on*, 2018, pp. 669–681.

[74] S. Angizi, Z. Shi, M. Gao, M. Xie, and Y. Zhang, "Graphs: A graph processing accelerator leveraging sot-mram," in *Design Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 378–383.

[75] S. Jain, A. Sengupta, and K. Roy, "Computing-in-memory with spintronics," in *Design Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1640–1645.

[76] M. Xie, Y. Zhang, S. Angizi, M. Gao, and Z. Shi, "Aim: Fast and energy-efficient aes in-memory implementation for emerging non-volatile main memory," in *Design Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 625–628.

[77] X. Williams and N. Mahapatra, "Analysis of recent trends in automatic object identification," in *International Conference on Computational Science and Computational Intelligence (CSCI)*, 2019.

[78] W. Burr, "Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses) using phase-change memory as the synaptic weight element," in *IEEE TED*, 2015.

[79] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Computer Architecture (ISCA) ACM/IEEE 43rd Annual International Symposium on*, 2016, pp. 27–39.

[80] M. Gao, P. Li, and H. Yang, "Tangram: Optimized coarse-grained dataflow for scalable nn accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2019, p. 807.

[81] S. Angizi *et al.*, "Imce: Energy-efficient bit-wise in-memory convolution engine for deep neural network," in *Design Automation Conference (ASP-DAC) 2018 23rd Asia and South Pacific*, 2018, pp. 111–116.

[82] K. Zou, X. Gao, X. Wu, X. Duan, W. Liu, and H. Wang, "Xorim: A case of in-memory bit-comparator implementation and its performance implications," in *Design Automation Conference (ASP-DAC) 2018 23rd Asia and South Pacific*. IEEE, 2018, pp. 349–354.

[83] S. Hamdioui *et al.*, "Guest editorial memristive-device-based computing," *Very Large-Scale Integration (VLSI) Systems IEEE Transactions on*, vol. 26, no. 12, pp. 2581–2583, 2018.

[84] L. Yang *et al.*, "Optimal application mapping and scheduling for network-on-chips with computation in stt-ram based router," *Computers IEEE Transactions on*, vol. 68, no. 8, pp. 1174–1189, 2019.

[85] V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," *IEEE MICRO*, vol. 37, no. 4, pp. 273–287, 2017.

[86] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *IEEE JPROC*, vol. 105, no. 12, pp. 2295–2329, 2017.

[87] L. Jiang *et al.*, "A low power and reliable charge pump design for phase change memories," in *Proc. ISCA*, 2014.

[88] S. M. Shivanandamurthy *et al.*, "A scalable stochastic number generator for phase change memory based in-memory stochastic processing: work-in-progress," in *CODES/ISSS*, 2019.

[89] Z. Li *et al.*, "Dscnn: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks," in *IEEE ICCD*, 2016, pp. 678–681.

[90] C. Chandra *et al.*, "Low-cost inter-linked subarrays (lisa): Enabling fast inter-subarray data movement in dram," in *IEEE HPCA*, 2013, pp. 568–580.

[91] S. Song *et al.*, "Palp: Enabling and exploiting partition-level parallelism in phase change memories," *ACM TECS*, vol. 18, no. 3, p. 26, 2019.

[92] X. Chen and et al., "A high-throughput and energy-efficient rram-based convolutional neural network using data encoding and dynamic quantization," in *Design Automation Conference (ASP-DAC) 2018 23rd Asia and South Pacific*, 2018, pp. 123–128.

[93] M. K. Qureshi *et al.*, "Scalable high-performance main memory system using phase-change memory technology," in *ISCA*. ACM, 2009.

[94] Q. Deng, Y. Zhang, M. Zhang, and J. Yang, "Lacc: Exploiting lookup table-based fast and accurate vector multiplication in dram-based cnn accelerator," in *DAC*, 2019.

[95] S. Han *et al.*, "Eie: Efficient inference engine on compressed deep neural network," in *IEEE ISCA*, vol. 44, no. 3. ACM, 2009, pp. 243–254.

[96] S. Xiaoyu, Y. Zhang, H. Li, Q. Hu, and L. Shi, "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1423–1428.

[97] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Scope: A stochastic computing engine for dram based in-situ accelerator," *IEEE MICRO*, pp. 696–709, 2018.

[98] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," *IEEE MICRO*, pp. 288–301, 2017.

[99] A. Biswas *et al.*, "Conv-sram: An energy-efficient sram with in-memory dot-product computation for low-power convolutional neural networks," *in IEEE JSSC*, vol. 54, no. 1, pp. 217–230, 2018.

[100] A. D. Patil, Z. Miao, A. Shafiee, S. Venkataramani, and K. Sengupta, "An mram-based deep in-memory architecture for deep neural networks," in *IEEE (ISCAS)*, 2019.

[101] B. L. Le and et al., "An efficient racetrack memory-based processing-in-memory architecture for convolutional neural networks," in *IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, 2017, pp. 383–390.

[102] 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," *IEEE MICRO*, pp. 273–287, 2017.

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

[104] Han *et al.*, "Eie: Efficient inference engine on compressed deep neural network," *in IEEE ISCA*, vol. 44, no. 3, pp. 243–254, 2016.

[105] v. Krizhe *et al.*, "Imagenet classification with deep convolutional networks," vol. 1097, 2010.

[106] Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *in IEEE ISCA*, vol. 44, no. 3, pp. 27–39, 2016.

[107] X. Sun *et al.*, "Xnor-rram: A scalable and parallel resistive synaptic architecture for binary neural networks," in *IEEE DATE*, 2018, pp. 1423–1428.

[108] S. Li *et al.*, "Drisa: A dram-based reconfigurable in-situ accelerator," in *IEEE MICRO*, 2017, pp. 288–301.

[109] Q. Deng *et al.*, "Dracc: A dram based accelerator for accurate cnn inference," in *IEEE DAC*, 2018, pp. 1–6.

[110] Balobas *et al.*, "High-performance and energy-efficient 256-bit cmos priority encoder," in *IEEE ISVLSI*, 2017, pp. 122–127.

[111] D. Lee *et al.*, "Tiered-latency dram: A low latency and low cost dram architecture," in *IEEE HPCA*, 2013, pp. 615–626.

[112] L. Orosa *et al.*, "Codic: A low-cost substrate for enabling custom in-dram functionalities and optimizations," in *IEEE ISCA*, 2021, pp. 484–497.

[113] Muralimanohar *et al.*, "Cacti 6.0: A tool to model large caches," vol. 27, p. 28, 2009.

[114] Chang *et al.*, "Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms," *in IEEE ACM*, vol. 1, no. 1, pp. 1–42, 2017.

[115] L. Jiang *et al.*, "A low power and reliable charge pump design for phase change memories," *in IEEE ISCA*, pp. 397–408, 2014.

[116] I. G. Thakkar *et al.*, "Dyphase: A dynamic phase change memory architecture with symmetric write latency and restorable endurance," *in IEEE TCAD*, vol. 37, pp. 1760–1773, 2017.

[117] K. Kim *et al.*, "Approximate de-randomizer for stochastic circuits," in *IEEE ISOCC*, 2015, pp. 123–124.

[118] S. M. Shivanandamurthy, I. G. Thakkar, and S. A. Saleh, "Odin: A bit-parallel stochastic arithmetic based accelerator for in-situ neural network processing in phase change ram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2021.

[119] D. Bhattacharjee, R. Devadoss, and A. Chattopadhyay, "Revamp: Reram based vliw architecture for in-memory computing," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2017.

[120] S. H. H. Angizi *et al.*, "Graphs: A graph processing accelerator leveraging sot-mram," in *DATE*. IEEE, 2019, pp. 378–383.

[121] S. Bavikadi, H. Hosseini, and A. Mahdi, "A review of in-memory computing architectures for machine learning application," in *Proc. GLSVLSI*, 2020.

Supreeth Mysore Shivanandamurthy

supreethms@uky.edu

Education

- **Masters of Technology in VLSI Designs**, Vellore Institute of Technology, Vellore, India, July 2015
- **Bachelors of Engineering in Electronics and Communication**, VTU University, India, July 2013