UNIVERSITY OF CALIFORNIA,
IRVINE

Efficient Acceleration of Computation Using Associative In-memory Processing

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Hasan Erdem Yantır

Dissertation Committee:
Professor Fadi J. Kurdahi, Chair
Professor Ahmed M. Eltawil
Professor Rainer Dömer

2018

DEDICATION

I would like to dedicate this dissertation to my family

for their invaluable support and unconditional love

throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

All praise be to Allah, Lord of the worlds.

This dissertation would not have been possible without the help of so many people in so many ways. It is the product of my educational, professional, and personal attainments through my discussions with people.

I would like to express my deepest gratitude to my advisor, Professor Fadi J. Kurdahi, who provided me with the guidance and support in all aspects of my research at UC Irvine and even after UC Irvine. I learned a lot from his guidance and valuable suggestions. His support and courage in the worth of this research especially during the many difficult times were an invaluable motivation source for me.

I would also like to thank my Co-advisor Prof. Ahmed M. Eltawil. The insightful discussions with him opened several prospects to me and enriched my research contributions from different fields. He always promoted my work and encouraged me in going further cordially.

I would like to thank Prof. Rainer Doemer for being in my committee. Even though I could not find an opportunity to conduct research with him, my observations on his professional as well as friendly academic life will be a good guider on my overall academic life.

I would like to thank Prof. Smail Niar for directing me in the hard paths of the academic life as like an academic coach.

I would also acknowledge the support from my research colleagues, especially Mohammed Fouda, Dr. Wael Mahmoud Elsharkasy, Ayoub Neggaz, Dr. Ahmed Nassar, and Dr. Ihsen Alouani.

I would also like to thank all my other colleagues and friends, especially Rasul Torun, Dr. Enver Adas, Dr. Volkan Gunes, Atila Ucar, Dr. Ahmet Tekin, Yasir Ak, and Mert Bayer for making all those years at Irvine a great and enjoyable experience while away from the family.

I would like to thank IEEE and ACM for granting me the permission to use my own publications as part of this dissertation.

Finally, and most importantly, I would like to express my deepest gratitude to my parents and my wife, who gave me encouragement and supported me cordially with their best wishes. Their guidance and constant support have always helped me to overcome the challenges that I have faced throughout my life. Therefore, I can never thank them enough for all they have done for me.

CURRICULUM VITAE

Hasan Erdem Yantır

EDUCATION

Ph.D. in Electrical and Computer Engineering **2018**
University of California, Irvine *Irvine, CA*

M.Sc. in Computer Engineering **2014**
Boğaziçi University *İstanbul, Turkey*

B.Sc. Minor in Electrical and Electronics Engineering **2011**
Yeditepe University *İstanbul, Turkey*

B.Sc. in Computer Engineering **2011**
Yeditepe University *İstanbul, Turkey*

RESEARCH EXPERIENCE

Graduate Research Assistant **2014–2018**
University of California, Irvine *Irvine, CA*

Graduate Research Assistant **2011–2014**
Boğaziçi University *İstanbul, Turkey*

TEACHING EXPERIENCE

Teaching Assistant **2015–2018**
University of California, Irvine *Irvine, California*

EECS 113 - Processor Hardware/Software Interfaces
EECS 152B - Digital Signal Processing Design and Laboratory
EECS 22 - Advanced C Programming
EECS 20 - Assembly Language and C Programming
EECS 12 - Introduction to Programming

REFEREED JOURNAL PUBLICATIONS

Hasan Erdem Yantır, Ahmed M Eltawil, and Fadi J Kurdahi. "A two-dimensional associative processor." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2018

Hasan Erdem Yantır, Ahmed M Eltawil, and Fadi J Kurdahi. "Approximate memristive in-memory computing." *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):129, 2017

Hasan Erdem Yantır, Ahmed M Eltawil, and Fadi J Kurdahi. "A hybrid approximate computing approach for associative in-memory processors." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2018 (Under Review)

Hasan Erdem Yantır, Ahmed M Eltawil, Smail Niar, and Fadi J Kurdahi. "Power optimization techniques for associative processors." *Journal of Systems Architecture, Elsevier*, 2018 (Under Revision)

Rana A Abdelaal, Hasan Erdem Yantır, Ahmed M Eltawil, and Fadi J Kurdahi. "Optimizing energy through adaptive bit width adjustment on resistive associative processors." *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2018 (Under Review)

Hasan Erdem Yantır, Ahmed M Eltawil, and Fadi J Kurdahi. "A 1K In-memory Fast Fourier Transform Processor." *IEEE Design & Test*, 2018 (Under Preparation)

Bashar Romanous, Hasan Erdem Yantir, Walid Najjar, Ahmed M. Eltawil, ad Fadi J. Kurdahi. "Accelerating Convolutional Neural Networks (CNNs) using Associate In-memory Processor. *ACM TACO*, 2018 (Under Preparation)

REFEREED CONFERENCE PUBLICATIONS

Mohamed Ayoub Neggaz, Hasan Erdem Yantır, Smail Niar, Ahmed M Eltawil, and Fadi J Kurdahi. "Rapid in-memory matrix multiplication using associative processor." *In Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, March 2018

Wael M Elsharkasy, Hasan Erdem Yantır, Amin Khajeh, Ahmed M Eltawil, and Fadi J Kurdahi. "Efficient pulsed-latch implementation for multiport register files: work-in-progress." *In Proceedings of the 2017 International Conference on Compilers, Architectures and Synthesis for Embedded Systems Companion*, page 5. ACM, 2017

Hasan Erdem Yantır, Mohammed E Fouda, Ahmed M Eltawil, and Fadi J Kurdahi. "Process variations-aware resistive associative processor design." *In Computer Design (ICCD), 2016 IEEE 34th International Conference on*, pages 49-55. IEEE, 2016

Hasan Erdem Yantır and Arda Yurdakul. "An efficient heterogeneous register file implementation for fpgas." *In Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 293-298. IEEE, 2014

Gorker Alp Malazgirt, Hasan Erdem Yantır, Arda Yurdakul, and Smail Niar. "Application specific multi-port memory customization in fpgas." *In Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, pages 1-4. IEEE, 2014

Hasan Erdem Yantır, Salih Bayar, and Arda Yurdakul. "Efficient implementations of multi-pumped multi-port register files in fpgas." *In Digital System Design (DSD), 2013 Euromicro Conference on*, pages 185-192. IEEE, 2013

Hasan Erdem Yantır, Ahmed M Eltawil, and Fadi J Kurdahi. "A Systematic Approach for Low-Power & High Endurance Resistive Associative Processor Design." *ASP-DAC*, 2019 (Under Preparation)

Hasan Erdem Yantr, Ahmed M Eltawil, and Fadi J Kurdahi. "APSim: An Open-Source Associative Processor Simulator with Benchmark Suite" *ASP-DAC*, 2019 (Under Preparation)

SOFTWARE

APSim `https://github.com/hasaney/APSim`
A cycle-accurate Associative Processor Simulator

ABSTRACT OF THE DISSERTATION

Efficient Acceleration of Computation Using Associative In-memory Processing

By

Hasan Erdem Yantır

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2018

Professor Fadi J. Kurdahi, Chair

The complexity of the computational problems is rising faster than the computational platforms' capabilities. This forces researchers to find alternative paradigms and methods for efficient computing. One promising paradigm is accelerating compute-intensive kernels using in-memory computing accelerators since memory is the major bottleneck that limits the amount of parallelism and performance of a system and dominates energy consumption in computation. Leveraging the memory intensive nature of big data applications, an in-memory-based computation system can be presented where logic can be replaced by memory structures, virtually eliminating the need for memory load/store operations during computation. The massive parallelism enabled by such a paradigm results in highly scalable structures.

The present thesis is studied against this background. The objective is to conduct a broad perspective research on in-memory computing. For this purpose, associative computing architectures (i.e., Associative Processors, or AP) are built by both traditional (SRAM) and emerging (ReRAM) memory technologies together with their corresponding software frameworks. For ReRAM-based APs, the reliability concerns coming with the emerging memories are resolved. Architectural innovations are developed to increase the energy efficiency. Furthermore, approximate computing approach is introduced for APs to perform efficient/low-

power approximate in-memory computing for the tasks which can tolerate some accuracy lost. The works also propose a novel two-dimensional in-memory computing architecture to cope with the existing deficiencies of the traditional one-dimensional AP architectures.

Chapter 1

Introduction

1.1 Motivation

The explosive growth of the Internet coupled with readily available powerful computing platforms has created the perfect conditions for wide-spread adoption. It is widely recognized that the most significant obstacle to the transformative vision of ubiquitous access to computing resources is the power/energy consumption barrier, limiting the extent, scope and longevity of billions of computing devices. Over the last decade, there has seen a sharp increase in the need for ultra-efficient computation platforms that necessitates robust, low-power processing cores. This need becomes more urgent due to the increased need for scaling computation to tackle key computations such as deep learning, artificial intelligence and their tremendous requirement of efficient big data processing. A key enabler to such vision is the availability of computing power that can process vast amounts of information rapidly, reliably, and at a low power budget. On the other hand, the research community speculates that CMOS scaling could end in around 2024 [68], making it unlikely that further area, performance, and energy improvements would be purely based on fabrication technology. There

is a general consensus of the researchers that computing based on traditional architectures is approaching its limits in terms of scalability and power consumption [67, 68]. For these reasons, researchers have begun to develop alternative computing methods. To cope with these standing barriers facing the future of computing, one must look into other means of improving the efficiency of computation by increasing parallelism rather than depending on transistor feature reduction [13]. Heterogeneous computing has been adopted as a mean to cope with the increased need for performance and power efficiency, whereby specialized engines are entrusted with domain-specific (e.g., video) or function specific tasks (e.g., vector/matrix processing) relying on GPUs, FPGAs, etc. However, this approach becomes insufficient if processing elements cannot be fed by the memory at the desired processing rate, leading to a significantly degraded overall performance despite the advancement in the process technology and parallelism. The requirements for these devices, which are necessary to support the future computing needs, such as small area, low power and high performance are in conflict with the realities of high performance computing systems using traditional architectures. Unfortunately, current processor solutions have not been adopted because they are significantly less efficient than ASICs in terms of power and area (or GOPS/W/mm^2). Perhaps the two most formidable barriers are (1) the gap between processing and memory speed, and (2) the power consumption barrier for a given performance target, limiting the extent, scope, size, weight, and longevity of the computing systems.

Clearly a radical shift from current approaches is needed to meet the demands of future computing systems at both the architecture and the device levels. In fact, it is well recognized that the lower limits of system power consumption are orders of magnitude below state-of-the-art low power realizations. Consider a case-in-point where a study to simulate one second of the human brain activity required 82,944 processors running for 40 minutes [96]. This is especially intriguing when one realizes that the brain consumes approximately 10 watts while performing an estimated 10^{12} - 10^{14} operations per second. The problem is exacerbated when considering mobile devices where power and area are crucial enablers. Modern multicore

processor chips rely extensively on very large cache hierarchies (L1, L2 and L3). In fact, these account for over 80% of the chip area and an even larger percentage of the energy budget [72]. To address these processing requirements of the ever-increasing amount of information, new computing paradigms started to emerge that focus more on the memory bottleneck problem together with the emerging semiconductor technologies. Broadly, these architectures aim to perform operations directly in the memory or near the memory to eliminate the data movement costs. This allows the creation of the new architectures on which von Neuman bottleneck has a minimal effect.

Theoretically and intuitively, the most memory efficient computing paradigm is in-memory computation where all computations are performed inside the memory without moving the data. *Associative processors* (AP) are excellent in-memory computational platforms for massively parallel (Single Instruction Multiple Data) computing that combine the memory and the processor in the same location [32]. In associative computing, the operations are carried out on the rows of a memory simultaneously. This feature inherently solves the memory-wall problem of traditional processor architectures since the memory and the processor are integrated. Even though numerous associative processors (AP) architectures were proposed in 1970's and 1980's [32, 142], their adoption was limited due to the unmanageable area and power requirements [32]. This reality has been changing with the availability of new semiconductor technologies (such as ReRAM [154], STT-RAM [4], and MRAM [140]), materials, and ultra scaling in transistors that allow for extremely dense memory structures. As described later in this section, Table 1.1 shows a summary of the emerging nonvolatile memory technologies [66] to point out this reality. As shown in the table, these devices provide ultra-high density when compared with traditional memory technologies such as SRAM together with non-volatility.

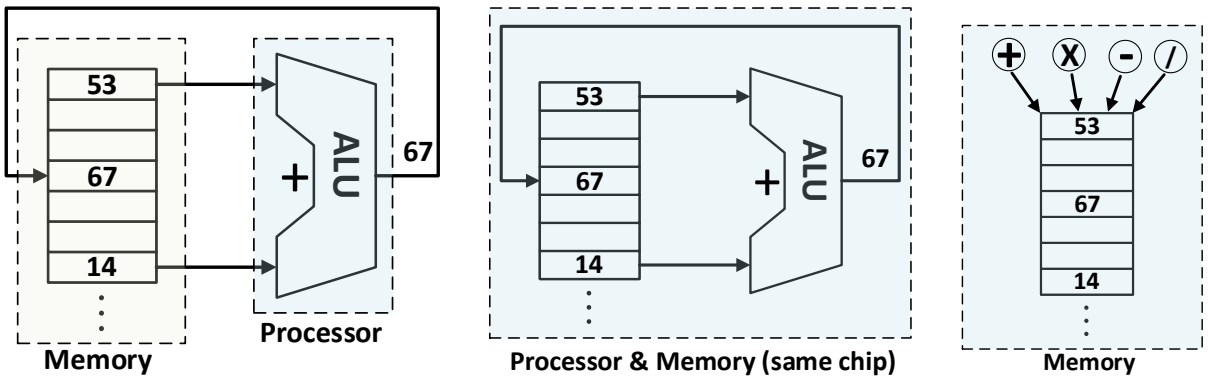
As a consequence, the improvements in the semiconductor industry lead to a resurrection of the AP approach in the research community [176, 42], and even in the commercial semicon-

ductor industry as an end product [38, 33, 61, 123]. In-memory based accelerators where logic can be replaced by memory structures, virtually eliminating the processor-memory traffic can achieve at least an order of magnitude more energy-efficiencies per area (GOPS/W/mm^2), when compared to existing systems. In this dissertation, the main focus is on the architectures of associative processors ranging from low-power computing to reliable in-memory processing and approximate in-memory computing. The approaches presented in this dissertation contribute to the design of in-memory processors/accelerators based on associative processing.

1.2 Background

1.2.1 Computation Types

There are many categorization of computation types such as with respect to the number of concurrent instruction and the data streams processed on the architecture (i.e., Flynn's taxonomy [31]), the degree of the parallelism (i.e., Feng's classification [11]), or pipelining and parallelism (i.e., Handlers Classification [7]). Irregardless of these classifications and processing features of the architecture, another categorization type is the organization of the processor with respect to the memory. From this perspective, there are three computation types which are defined by the relative placement of the memory and the processor. Referencing this classification, any computing system can be fall into one of these three classes; traditional(out-memory), near-memory, or in-memory. The following three subsections details this classification.



(a) Traditional (b) Near-memory (c) In-memory

Figure 1.1: Computation types with respect to the memory organization

Traditional (Out-memory)

As the traditional and most commonly used computational method, out-memory architectures place the memory and processor in separate chips. During the computation, the data stored in the memory is fetched to the processor. The processors performs the operations by the arithmetic logic unit (ALU) and the results are sent back to the memory. The communication between processor and memory is handled through high-speed buses. The von Neuman (Princeton) [162] and Harward architectures are the examples of out-memory computation and they differ in terms of where the instructions and the data are stored [47].

Near-memory

As the von Neuman architecture faces the memory bottleneck problems, the researchers have come up with the idea of placing both memory and processor inside the same chip in order to enable higher bandwidth between them, thus to obtain the faster communication and computation. In this organization, the data read from the memory can reach to the processor much faster (and vice versa) than a traditional architecture since they are in the same chip. The study of intelligent RAM (IRAM) [93] is one example for near-memory computation where DRAM is integrated with the processor by eliminating the memory cache to optimize

the cost and performance trade-off [92].

In-memory

In-memory computation aims to perform the operation directly inside the memory (i.e., not inside the same memory chip as in near-memory computation). There are different realizations of in-memory computing architectures. The most basic one is placing a small arithmetic logic units within the memory to perform some operations on a group of data [43]. In some cases, the operations are performed by using the analog or digital processing capability of emerging technologies such as memristor [46, 104]. As another method, associative processing described in the next chapter (Chapter 2) performs the in-memory computation by using look-up tables of the arithmetic and logical operations. Unlike the von Neuman or near-memory computation in which the data sent to processor for computation, associative processors sent the functionality or operation over data without moving it.

In here, it is worth to mention that associative processing term is commonly used for the architectures employing associative memories (i.e., CAM) for computing. For example, even though associative computing architectures proposed in [69, 43, 52, 54, 51] uses a CAM structure, the operation is not done inside the CAM or memory. The CAM is only used for associative search and the search results are processed either using a small, basic processor near the memory (inside the same chip) or the main processor. On the other hand, the associative processor referenced in this thesis perform the processing within the memory directly. In other worlds, the computation is done inside the CAM without using any additional processor. Therefore, there are differences between the two processor types. Unfortunately, associative computing terms are used for both architectures since they are based on the associativity of the memory. In reality, their computation methods are totally different.

1.2.2 Non-volatile memories

There are many alternatives in the memory design and each alternative has its own ff. For in-memory computing, the memory structure in which computation is done also plays an important role. Traditionally, designers depended on a hierarchical memory structure to balance the requirements of speed, power and area across the various layers. However, emerging memory technologies are providing alternative means to flatten the hierarchy, while maintaining or surpassing the best features of current state of the art architectures. Table 1.1 shows a detailed comparison between traditional (current) and emerging memory technologies [72]. In the case of traditional memories, there exists a clear trade-off between speed and density, where SRAM provides the highest speed with low density. On the other hand, Flash memories are very dense but much slower. Emerging technologies provide means to avoid this tradeoff. Three of the four main emerging technologies are resistive based memories (ReRAMs), namely Phase Change Memory (PCM), Spin-Transfer Torque (STT), and memristors based on the REDOX phenomenon. From an architecture point of view, the three technologies are quite similar; nevertheless, memristor seem to be the most promising, since it provide the highest density among all the emerging technologies and the lowest access latency [72]. Moreover,memristors can scale much easier than STT-RAMs, offer better read times and generally better or comparable write times and energy per bit. When compared to PCM-RAMs, memristors use much less energy per bit, and are better on almost all other metrics such as density, read and write times [49]. However, there are many challenges that need to be addressed before Resistive memories can genuinely replace current memory technologies. Among these challenges are manufacturability, variability, and as will become clear later, power density. Some of these challenges are being addressed in industry and academia. Recently, Intel and Micro Technologies announced their 3D Xpoint Memory technology bearing many of the characteristics of memristors such as density, speed, and crossbar architecture, and claiming to be 10x denser than DRAM and 1000x faster than

NAND flash. These memories have started to be packaged into Non Volatile-DIMMs and Solid State Drives [165, 64].

Table 1.1: ITRS report for emerging memory technologies [66] and their comparison with traditional memories based on recent literature [45, 171, 70, 106].

	Traditional Memories				**Emerging Memories**			
	SRAM	**DRAM**	**NAND FLash**	**NOR Flash**	**FeRAM**	**STT-RAM**	**PCM**	**ReRAM**
Cell Element	6T	1T1C	1T	1T	1T1C	1(2)T1R	1D1R	**1R**
Feature Size (nm)	45	36-65	45	16	180	65	45	**5**
Density (Gbit/cm^2)	0.4	0.8-13	4.9	97.6	0.14	1.2	12	**1000**
Read time (ns)	**0.2**	2-10	15	10^5	45	35	12	-
Write time (ns)	**0.2**	2-10	10^3	10^5	65	35	100	< 1
Nonvolatile	No	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Endurance [45]	**>1e16**	**>1e16**	1e15	1e15	1e14	>1e12	1e9	1e15
Retention Time	-	<<seconds	**Years**	**Years**	**Years**	**Years**	**Years**	**Years**

* The abbreviations used are: T-transistor, C-capacitor, R-resistor, and D-diode. The bold font indicates the best value per row.

The Memristor (memory resistor) is a nonlinear passive device that changes its state according to the net charge passing through its two terminals, and maintains its state after the electrical bias is removed. The Memristor is widely considered as the fourth basic two terminal passive element, alongside with resistor, capacitor, and inductor. The existence of such a device was postulated since 1970s [19, 20], however it was not until 2008 when a fabricated device was related to the theory [154]. While the memristive phenomenon has been observed for quite some time [131], recent fabrication advances [167, 172, 82, 37, 89, 75] make it very appealing where the resistance of the [8] device represents a memory state [17, 26, 87, 184, 48, 163]. In addition to memory systems, memristors find many applications such as programmable analog circuits [148, 127, 21], logic and arithmetic circuits [34, 136, 113, 35, 101, 166], neural networks [1, 74, 85, 126, 62, 81, 99], electronic oscillators [24, 185, 186, 156, 157, 22], filters [23], and cryptography [108]. Several behavioral, circuit, and physical models have been introduced in order to facilitate the emerging technology [5, 132, 12, 164, 160, 71].

1.3 Contributions

The objective of this thesis is to explore the design space, system architecture, software and micro-architecture, enabling low-power in-memory accelerators for large-scale data intensive applications. The primary contributions of this study can be summarized as follows:

- **Architecture and Trade-offs of Associative Processors:** The understanding of the associative processing and processors are described in detail together with the newly defined operations and system-wide configuration schemes. The proposed system architectures allow reprogrammability in memory-based computation and they are uniquely suited for vector based operations, while fully benefiting from the extreme parallelism. It is shown that emerging memory technologies make it possible to build CAMs that deliver one to three orders of higher GOPS/W/mm^2 than current parallel architectures. The architectural innovations are proposed that reduce or eliminate the need for any supporting logic, thus addressing the two main barriers to adoption and making AP based on CAMs an excellent candidate for the development of in-memory accelerators. The statement are supported by the trade-off analysis in terms of energy, performance and reliability.

- **Approximate In-memory Computing:** For the first time, approximate in-memory computing is introduced for the APs. For ReRAM-based APs, two approximate computing methodologies are proposed; bit trimming and memristance scaling. It is emphasized that the two methods are naturally supported by the APs as dynamic and tunable. Later, these concepts are extended for the SRAM-based APs where bit trimming and cell voltage scaling is applied. Furthermore, the hybrid approximate computing is introduced for both AP architectures in which bit trimming and voltage scaling (in SRAM)/cell scaling (in ReRAM) are combined and a design flow is proposed to optimize the system efficiency with a minimum impact on the accuracy.

- **Low-power Associative Processor:** A low-power SRAM-based AP implementation is suggested by proposing novel architectural improvements to decrease the switching activity. Furthermore, some traditional operations are modified to allow better energy efficiency. For ReRAM-based APs, a considerable energy reduction is provided by multi-compare architectures where ReRAM switching range is scaled without sacrificing the reliability constraints.

- **Software Framework for Associative Processors:** A cycle accurate simulator for Associative Processors is delivered along with the broad range of benchmarks such as large-scale data mining, signal processing, and deep learning. The simulator is highly configurable with more than 50 parameters and can facilitate circuit-level simulation as well as system-level simulations. The simulator works as fully automated in which the cooperation between the system-level simulator (Matlab) and circuit-level simulator (HSpice) is coordinated seamlessly.

- **Two-dimensional Associative Processor:** Even though traditional in-memory processor architectures together with emerging semiconductor technologies show promise for improving the efficiency of parallel computing, they lack some vital requirements such as flexibility and sequential execution. For this reason, a novel two-dimensional in-memory computing architecture is proposed. The proposed Associative Processing (AP) architecture is implemented by both CMOS/SRAM and ReRAM technologies and employed as an accelerator. The proposed architecture facilitates very efficient in-memory parallel computing together with a high degree of flexibility that results in faster running time in fundamental benchmarks. Furhermore, the developed architectural innovations reduce or eliminate the need for any supporting logic, thus addressing the barriers to adoption for more benchmarks.

1.4 Thesis Overview

After this brief introduction on the fundamentals, contributions, and previous works, the rest of the thesis is organized as follows. The thesis is divided into seven chapters; Architecture and trade-offs (Chapters 1-2), power optimization techniques for APs (Chapters 3-4), and architectural extension and future work (Chapters 6-7). Chapter 2 gives the detailed information on the architecture and operation of the APs in detail which is the fundamental background referenced through the thesis for in-memory computation. The chapter gives information for both SRAM-based and ReRAM-based designs. In Chapter 3, the APs implementations are compared with each other and with other processors to exhibit the trade-offs. The section also presents some studies to escalate the reliability issues. Next, Chapter 3 focuses on the energy efficient AP architectures and proposes some methods and modifications to decrease the power consumption and increase the efficiency of APs. The Chapter 4 also introduces the approximate in-memory computing. Then, Chapter 6 proposes a novel two-dimensional AP architecture which is capable of performing parallel operations on both horizontal and vertical directions. Finally, Chapter 7 concludes the thesis with a summary and starts a discussion on the potential directions as venues for future investigations.

Chapter 2

Associative Processor

In this chapter, the overall architecture of an associative processor implemented by ReRAM-based and SRAM-based CAM cells are described. The chapter aims to comprehend the understanding of the traditional APs by including the new materials such as newly defined operations and their corresponding look-up tables. The chapter also includes the proposed in-house simulator for associative processors.

2.1 Introduction

An associative processor can be considered as a variant of the single-instruction multiple-data processor (SIMD) that combines storage and processing in the same device. In APs, a key is matched by all the rows in memory, and bits are modified depending on (1) the key mask and (2) whether or not a match occurred. By correctly sequencing matching steps, vector based arithmetic operations can be performed in place. In this way, it performs the SIMD processing inside the memory directly.

The idea of Associative processors has its roots back in the 1970s and 1980s with the work of

Scherson, Elgin, Foster, and many others [144, 32, 147, 143]. In these studies, basic addition algorithms were derived and other operations studied for integer and floating point operations. One of the earliest commercial APs was STARAN [138] in the 1970s by Goodyear Aerospace Corporation. STARAN was interfaced with a conventional general purpose computer. STARAN's main component was the "main frame" memory which enables associative addressing and parallel processing capabilities for array arithmetic operations. An associative programming language (APPLE stands for Associative Processor Programming LanguagE [138]) as well as software for the standalone processing mode of STARAN [9] were developed. Further enhancements to STARAN include multi-dimensional memory access [9] to enable bit-slice and word-slice accesses. Applications such as FFT, Sonar post-processing and string search were accelerated on STARAN [9]. Since then, numerous studies of AP architectures [129, 32] have demonstrated their out performance of traditional processors on various applications ranging from searching and sorting, to FFT, matrix multiplication and sparse linear algebra [141, 143, 177, 178] as well as signal, image, and video processing [139, 6, 147, 3]. Algorithms for the acceleration of spares matrix multiplication using AP are described in [178] where four algorithms were studied, covering fully associative sparse matrix multiplication and a hybrid of AP and CPU computations reducing the computational complexity of such multiplication on AP to $O(m)$. In [177] the authors proposed replacing the on-chip last-level cache with an AP that would work as both memory as well as massively parallel SIMD accelerator.

All of the former APs rely on traditional old CMOS technology which limited the widespread adoption of APs due to large chip area and high power consumption which in turn limited the maximum parallelism practically achievable. The other drawback was the limited scope of compute operations that could benefit from existing CAM structures. This reality has been changing with the availability of new semiconductor technologies (such as ReRAM [154], STT-RAM [4], and MRAM [140]), materials, and ultra scaling in transistors that allow for

extremely dense memory structures (sub-16 nm). Memristor (ReRAM) [1] is a new device technology, which is a nonlinear passive component that changes its state according to the net charge passing through its two terminals, and maintains its state after the electrical bias is removed [19]. Therefore, It is possible to use memristor with two resistance levels to represent a single bit. Memristors technology has revolutionized memory manufacturing by enabling the accommodation of large memory sizes on small chip area with low power consumption. They offer a way out of the status quo due to their suitability to implement CAM structures and crossbar connections that are crucial components for Associative Processors. With the advent of emerging memory technologies, research efforts were geared towards harnessing the advantages of these technologies to make CAMs and APs more attractive. A resistive ternary CAM (TCAM) accelerator based on PCM, and using 3T3R cells (three transistors three resistors per cell) was designed as a DDR3-compatible DIMM [40]. The design allowed the resistive TCAM to work with a general purpose CPU as both content addressable memory as well as conventional address-based RAM memory. AC-DIMM, a further enhancement to this design [42, 41] uses STT-MRAM with 2T1R cells to extend its functionality to associative search and in-memory processing. More recent work on Associative Processors include [42], using STT-RAMs, and [176], using bipolar resistive RAM, to build CAM-based APs. In [176] a resistive AP using diodes is proposed with an area and power intensive CMOS-based reduction tree for population count and reduction operations. The study in [119] provides a matrix multiplication on ReRAM-based AP in $O(m)$ complexity for the traditional matrices. As another application from machine learning, the study in [77] proposes in-memory acceleration for K-means and K-nearest neighbors tasks.

In addition to APs, numerous significant contributions are made in CAM structures which is the principal component of an AP having the ability to perform a search on a large collection of words in a single step or cycle. An energy efficient TCAM, Multiple-Access Single-Charge

[1] In this thesis, the terms ReRAM and memristor are used interchangeably since memristor is referenced for AP implementation as the emerging memory technology.

(MASC), was proposed in [55, 54]. The proposed TCAM saves energy by using a search operation scheme that is contrary to the one used in traditional CAMs. By having the mismatched rows maintain the precharge voltage while the matching row discharge theirs.The voltage from mismatched rows is used to precharge the match rows again. This allows for additional search operations to be performed while skipping a precharge cycle. Search results are produced by the TCAM within 1-2 bit Hamming distances of the exact value. This TCAM doesn't have any processing capabilities on its own. It can be used as an accelerator along side a GPU. TCAM can be used to skip performing frequent floating point (FP) operations, such as addition, multiplication and square root by fetching FP operation results from TCAM instead of utilizing the FP unit.This is done by caching frequent patterns of operands and the results of those operations in the TCAM. Profiling is needed for each application in order to detect those frequent computational patterns used in FP operations. A low energy Resistive Multi-stage Associative Memory (ReMAM) was proposed by [53]. Energy saving are achieved by dividing searches in the TCAM onto smaller searches and performing searches in several stages. However, both MASC and ReMAM can only function as associative memory which provides storage and content-based search, requiring other computational blocks to perform near-memory computing.

2.2 Architecture

The idea of associative processing is based on a content addressable memory (CAM) which is a special type of memory used in computer systems that requires fast data searching operations [122]. As the building blocks of APs, CAMs have the ability to perform a search on a large collection of words in a single step or cycle. Unlike the traditional computer memories like random-access memory (RAM) where the data is located with a a memory address, the CAM memories search the content (data word) inside the memory to decide on its existence

and the location (address). Shortly, the RAM is optimized for address based access and the CAM is optimized for content/data-based access. For this reason, even though a search operation inside the traditional computer memory takes $\mathcal{O}(n)$ complexity, the CAMs decrease it to $\mathcal{O}(1)$. Since an associative processor requires to process the data inside the memory by using these fast data searching operations, the main component forming the architecture of an associative processor is the CAM. While CAMs have been studied and used, on a very small scale, for decades, their implementation in traditional technologies (e.g. CMOS) is very onerous in area and power. New device technologies however, such as Phase Change Memory (PCM), Spin-Transfer Magnetic RAM (STT-MRAM), and Memristors, offer the possibility of building large-scale memories that are much smaller in area and require less energy per bit, and are non-volatile; features that offer transformational potential in how CAMs can be used.

The architecture of an associative processor (AP) is presented in Figure 2.1. The AP consists of a content addressable memory (CAM), controller, instruction cache, specific registers, and an optional interconnection circuit. In the processor, instruction cache holds the instructions that are performed on the CAM. The controller generates the required mask and key values for the corresponding instruction. The *key* register is used to store the value that is written or compared against. *Mask* register indicates which bit or bits are activated during comparison or write. The rows matched by the compare operation are marked in the *tag* field. The rows tagged with logic-1 means that the corresponding CAM row has been matched with the given key and mask value. For example, if we use key 100 and mask 101 to the CAM, the tag bits of the corresponding rows whose first and third bits are logic-0 and logic-1 respectively becomes logic-1 as shown in the figure.

In order to process data inside the CAM, APs must have masked search and parallel column write operations. These requirements can be fulfilled by various CAM cells built using the gated memory cells. Figure 2.2 shows the two possible implementations of a CAM cell

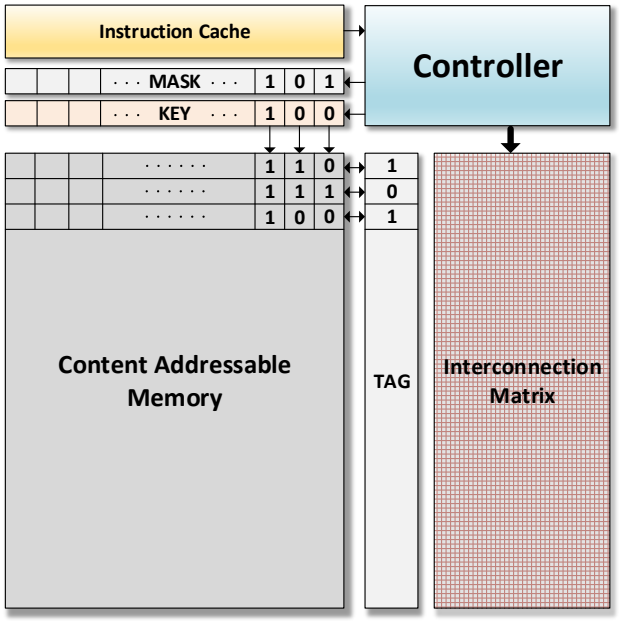
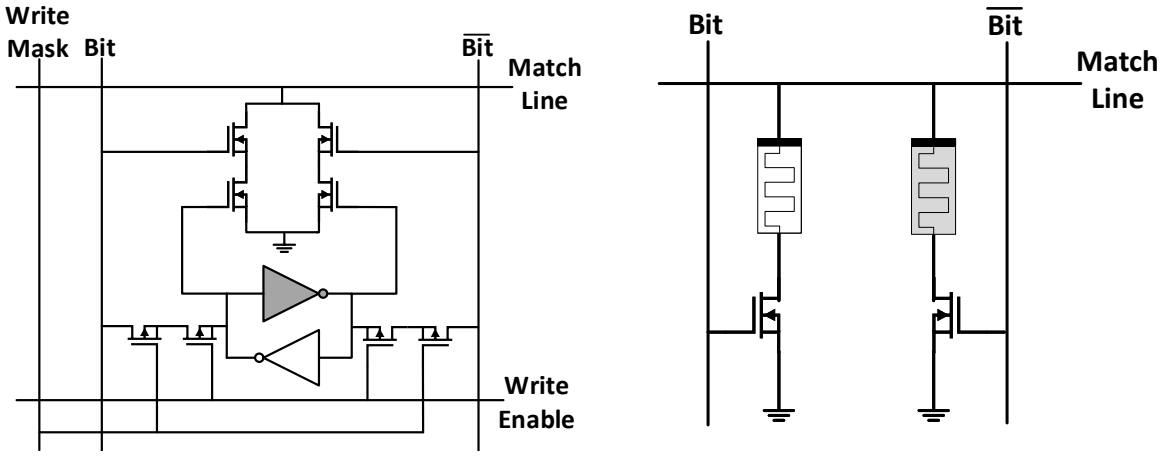


Figure 2.1: General architecture of an associative processor

which are SRAM-based and ReRAM-based CAM cells. When a CAM is implemented by resistive memory technology, it is named as *Resistive CAM (RCAM)* and the AP composed of RCAMs is named as *ReRAM-based AP or Resistive AP (RAP)* as opposed to the traditional CMOS/SRAM-based CAMs, which we refer to in this thesis as *SRAM CAM*, or SCAM, and the corresponding AP as SRAM-based AP, or *SAP*, while the terms CAM and AP refer to the generic content-addressable memory and associative processing architectures, respectively. The following two subsections explains the SAP and RAP architectures in detail.

2.2.1 SRAM Associative Processor (SAP)

In SAP, the CAM is consists of the SRAM-based CAM cells in which the storage is achieved through the SRAM-cells [91, 15]. As the traditional way, an SRAM-based cell can be used to store one-bit together with an additional logic for masked search and write operations (Figure 2.2a) [122]. In this cell, the one-bit data is stored by a coupled inverter where each inverter supports to other to keep its logical value. For this reason, the cell consumes the



(a) SRAM-based CAM Cell (b) ReRAM-based CAM Cell

Figure 2.2: CAM cell implementations

static energy as well as the dynamic energy. The data of the cell can be accessed through the outer circuitry. Figure 2.3 shows the detailed architecture of a SAP. For the sake of a better demonstration, the inverters which output logic-1 is shown as grayed and the ones output logic-0 is shown as white. As the fundamental operations of the associative processing, the compare and write operations on the CAM are performed as follows:

Compare: During the compare, a matching circuit attached to each row distinguishes the rows that matched with the combination of given key and mask values from the mismatched ones [176]. Basically, this circuit uses two phases to differentiate the matched and mismatched rows; *pre-charge* and *evaluate*. In the pre-charge phase, the capacitors at each row (row capacitors) of the CAM are pre-charged. During the evaluate, a search word is applied to the columns. Only rows carrying matching data will retain charge because their transistors switched off on the leakage path and the other rows leaks their charges since transistors conduct. On the other hand, the rows matched by a compare operation retain the charge on the capacitor. As seen in Figure 2.3, the key is masked and then applied with its reverse. In this case, to look for logic-1, "01" is sent to the columns and for logic-0, "10" is sent. To exclude the columns from the match operation, simply "00" (don't care) is applied. For both cell types, these data are applied to the matching transistors. Figure 2.4 shows these tree

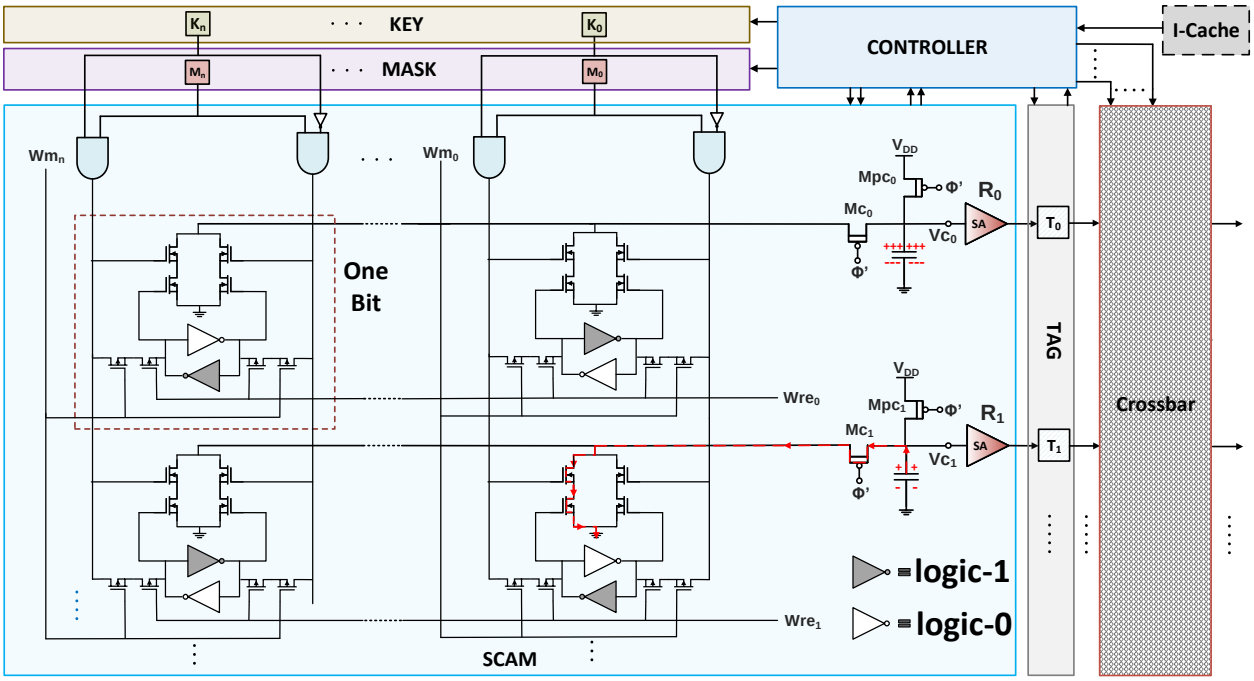
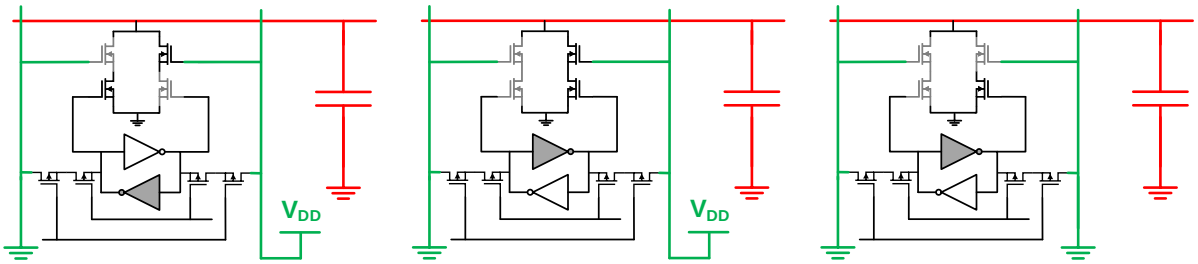


Figure 2.3: Architecture of an SRAM-based Associative Processor (SAP)

cases explicitly for the SRAM-based CAM (SCAM) cell. In the SCAM cell, if the stored bit is the reverse of the searched bit, the matching transistors (on the top of the coupled inverter in Figure 2.2a) forms a short path to the ground and the charge across the capacitor leaks on this path. If they are same, the path becomes closed and only a small amount of charge can leak. The figure indicates the paths of the closed transistors as black and the closed paths as gray. The charges on a row capacitance leak from the mismatched cell, where the both series transistor are of open state creating a path to the ground, as shown in Figure 2.4b. On the other hand, Figure 2.4a shows the state of the SAP cell in case of a match. In the figure, the CAM-cell stores a logic-1 value and the "01" pattern is applied to the cell to look for logic-1. Since this a match case, the inverter feeding to the right matching circuit closes the transistor while other one opens. On the other hand, "01" pattern closes the left matching transistor and opens the right one where no path to the ground is available through the matching transistors. SRAM-based cell cannot provide a don't care state. On the other hand, the cell can be excluded from the compare operation as shown in 2.4c where the columns are excluded from the compare operation. This provides a mechanism to

perform don't care as column-wise. Following the evaluate phase, a sense amplifier existing at each row senses the residual charge across the capacitor and compares it with a given reference voltage (V_{th}). As a result of comparison, it generates a logical correspondences of match and mismatch cases as logic-1 and logic-0 respectively. For example, if we set the key to 100 and mask to 110, the tag bits of the corresponding rows whose third and second bits are logic-1 and logic-0 respectively becomes logic-1 and the rest is logic-0.



(a) Match (b) Mismatch (c) Don't care

Figure 2.4: Typical evaluation phases of a SAP cell for the match (a), mismatch (b), and don't care (c) states.

Write: After a compare operation, the sense amplifier connected to the output of the matching circuit differentiate the matched and mismatched rows by tagging them either as logic-1 or logic-0 respectively. The rows matched by a compare operation are marked in the *tag* field, that is, the rows tagged with logic-1 means that the corresponding CAM rows have been matched with the given key and mask value. Following a compare operations, the associative processor needs to write to the specific columns of the tagged rws. In order to write to the specified columns of the matched SCAM rows, both write enable and write mask inputs of the cell is asserted. Then, the value and its reverse are applied to bit and $\overline{\text{bit}}$ columns respectively.

2.2.2 Resistive Associative Processor (RAP)

SRAM-based CAMs are traditional and well studied in the past. However, the cells are area inefficient since a single cell consists of 12 transistors. Fortunately, with the advent of

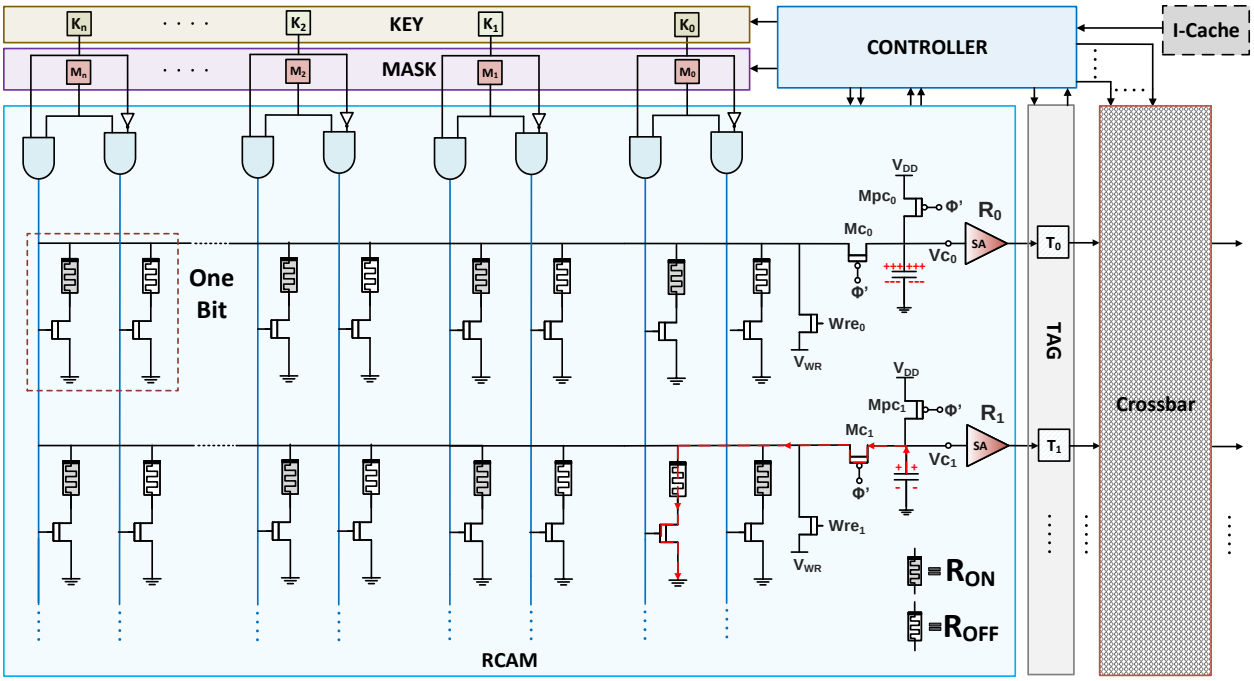
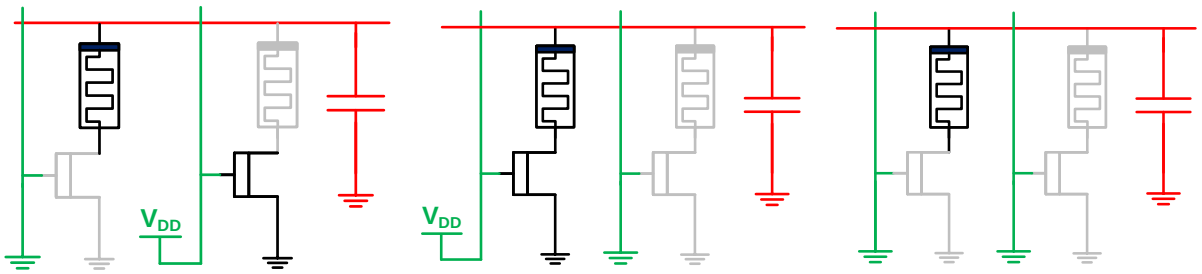


Figure 2.5: Architecture of an ReRAM-based Associative Processor (RAP)

new semiconductor materials, one of the most promising candidates for a basic CAM cell is presented in [103], which is made of two memristors (ReRAM) and two transistors (Figure 2.2b). This cell complies with the requirements of the associative computing and exists as a commercial product [33], [61], [123]. In this cell, binary data is stored in the form of "Low" (R_{on}) and "High" (R_{off}) resistances (i.e., they correspond to logic-1 and logic-0 respectively) in a complementary mode. Therefore, the device can work as a storage element and a switch at the same time, as in the SRAM cell. Unlike the SRAM cell which stores the data as binary, the ReRAM-based CAM cell stores the data as ternary since there are two ReRAMs inside a cell. In this case, a binary value is coded by exploiting these two ReRAMs. If the right and left ReRAMs are set as high and low respectively, the cell corresponds to logic-1. In the reverse case, the cell stores the logic-0. This cell also supports the ternary CAM operations where the cell can be excluded from a search operation on the corresponding column be setting the two memristors as both high (R_{off}).

Compare: Similar to the SRAM-cells, the compare operation in the RAPs is achieved by



(a) Match (b) Mismatch (c) Don't care

Figure 2.6: Typical evaluation phases of a RAP cell for the match (a), mismatch (b), and don't care (c) states.

pre-charging the capacitors on all the rows of the array, and then applying a search word (i.e., key) to the columns. However, in here, the charges on a row capacitance leaks the mismatched cell, where the memristor and the series transistor are of low resistance creating a path to the ground, as shown in Figure 2.6b. The data is stored in a "2T2M" cell in a complimentary mode, since the high resistance device will not leak charges to the ground even in case of mismatch, however its complement device will do so. Figure 2.6a shows the state of the CAM cell in case of a match, where no path to ground is available. In this case, the both columns of the cell form a high resistance path through either the closed transistor or the memristor with high resistance state (R_{off}). A "Dont care" state can be stored on the cell by setting its two memristors to High resistance, where no path is created to the ground independent of the search bit, as shown in Figure 2.6c.

Write: In ReRAMs, the write operations is performed by changing the resistance of ReRAMs either from R_{on} to R_{off} or R_{off} to R_{on}. If the used memristor has a threshold for the writing [105], the value of the ReRAMs can be set by applying a proper voltage across the memristor. The amount of the voltage should be higher than the write threshold voltage of the memristor and the polarity must be set according to the written memristance (high or low). Writing to RCAM-cell in an RAP system is performed using a one column at a time scheme. However, this is translated into two writing steps, since a complimentary data column is electively made of two columns of the CAM array. The bits to write are loaded to the match

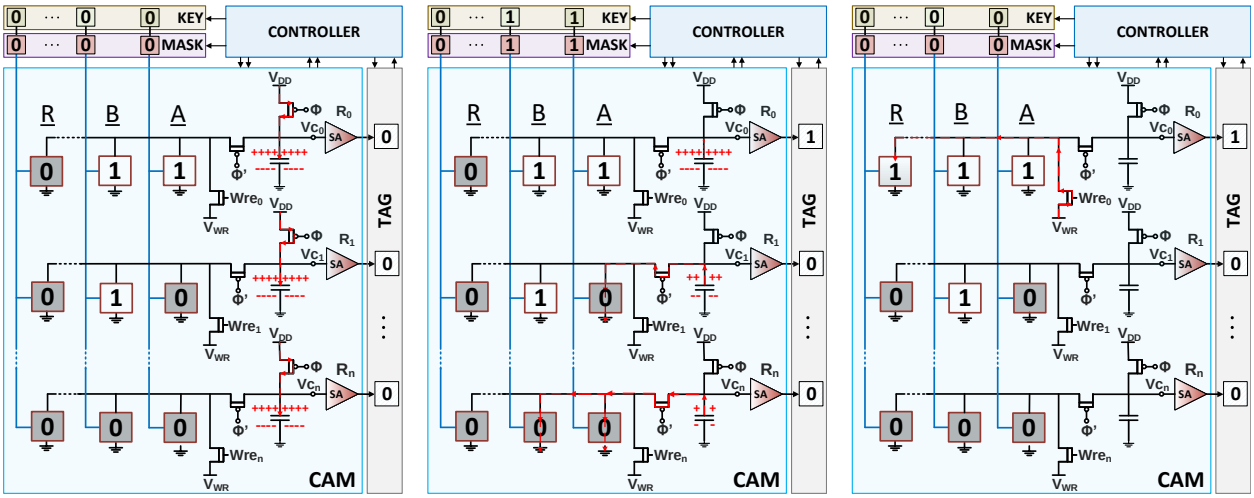
lines of the rows, with a search word of logic-0 or logic-1 at the column of interest and "Dont Care" elsewhere is written to the columns to activate the column of interest. This eliminates the need for any modification to the column driving circuitry used for compare.

2.3 Instructions

An instruction/operation on AP consists of consecutive *compare* and *write* phases. During the compare phase, the matched rows are selected and in the write phase, the corresponding masked key values are written onto tagged CAM words. Depending on the desired arithmetic operation, the controller sets the mask and key values by referencing a lookup table (LUT). In the compare phase, the key and mask fields are set and compared with CAM content, while in the write phase, tagged rows are changed with the key. In other words, the truth table of the function is applied (in an ordered sequence) to the CAM to implement the required function. Utilizing consecutive compare and write cycles with a corresponding truth table, any function that can be performed on a sequential processor can be implemented on the AP as a parallelized operation. In the following subsections, the basic logical and arithmetic operations performed on the AP are detailed.

2.3.1 Logical Instructions

As stated in the previous section, the operations are performed on the AP by utilizing their corresponding LUT tables irrespective of whether it is a SAP or RAP. As an example of a basic operation on the AP circuit, Figure 2.7 shows how a parallel AND ($R = A\&B$) operation is performed on the AP where the first and second columns corresponds to the A and B respectively and the last column is R (initially all 0). The LUT operation in the CAM can be simply performed by looking for "11" in the A and B columns of the CAM and



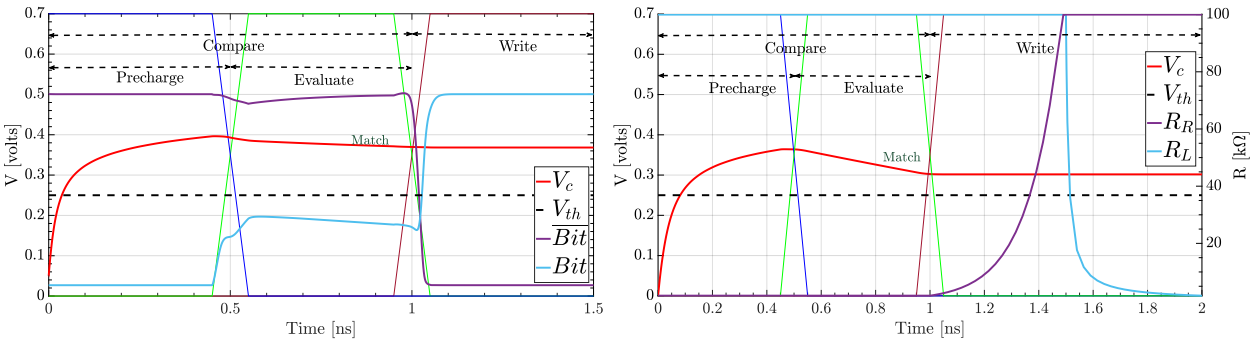
(a) Precharge (b) Evaluate (c) Write

Figure 2.7: AND operation on the AP

writing logic-1 to the result column of the matched rows as indicated by the AND. Since the other combinations of A and B have no effect on the R (i.e., already logic-0 as default), there is no need to look for them in the CAM. During the operation, firstly, all rows of the CAM is pre-charged (Figure 2.7a) to perform the subsequent evaluate operation. Then the LUT of AND operation is applied to the CAM where CAM is searched for "11" in the input columns (A and B) (Figure 2.7b). The second and third rows leak the charge stored across the capacitor, so only second row matches with the given key and mask values. Finally, "1" is written to the result column (R) of the tagged row in Figure 2.7c. Figure 2.8 shows the voltage and the memristance changes on a SCAM row (Figure 2.8a) and a RCAM row (Figure 2.8b) during the of AND. In both figures, the red line shows the voltage change on the input of the sense amplifier (i.e., the voltage change across the capacitor of the first row) where there is a match at the end of the evaluate cycle, that is, (V_c) voltage is bigger than the threshold (V_{th}). After the compare phase, the write operations are performed where the new data is written into the cell. In SCAM (Figure 2.8a), this operation takes one cycle and simply logical value of the coupled inverter changes. In RCAM (Figure 2.8b), the write operation is performed in two cycles where two complementary ReRAMs change their memristances from R_{off} to R_{on} and vice versa. The right vertical axis of Figure 2.8b shows

the memristance values of the left and right ReRAMs respectively.

In a similar way to the AND operation, other logical operations are performed on the CAM in a bit-serial, row-parallel manner by using their corresponding LUTs. Table 2.1) shows the LUTs of primitive logical operations on the AP which are NOT, AND, OR, and XOR. The other logical operations can also be performed on the AP easily by using their corresponding LUT tables. For the bit shifting operations, the location of the key column which points to the first bit of a value can be shifted to right or left simply, thus APs inherently support the bit shifting operations.



(a) SAP (b) RAP

Figure 2.8: Spice simulation of two consecutive cycles in SAP and RAP respectively corresponding to the AND operation in Figure 2.7

Table 2.1: Logical operations and their LUTs

(a) NOT

A	R	*Comment*
0	1	$1stPass$
1	0	*NC*

(b) AND

B	A	R	*Comment*
0	0	0	*NC*
0	1	0	*NC*
1	0	0	*NC*
1	1	1	$1stPass$

(c) OR

B	A	R	*Comment*
0	0	0	*NC*
0	1	1	$1stPass$
1	0	1	$2ndPass$
1	1	1	$3rdPass$

(d) XOR

B	A	R	*Comment*
0	0	0	*NC*
0	1	1	$1stPass$
1	0	1	$2ndPass$
1	1	0	*NC*

2.3.2 Arithmetic Instructions

Addition and Subtraction

In traditional computer arithmetic, 2's complement is the most widely accepted representation in signed arithmetic operations. For this reason, APs use this notation while storing the numbers. In the implementation of addition or subtraction, the result can be written into one of two locations; replace one of the inputs or a new location. The former one is referred to as in-place and later one is out-of-place.

Table 2.2a illustrates the look-up table (LUT) for both in-place and out-of-place additions. In all tables, R and Cr represent result and carry respectively. A and B indicate the inputs. Depending on the operation, the controller sets the mask and key values by referencing the corresponding LUT. In the compare phase, the key and mask fields are set and compared with CAM content according to the left side of the table. In the write phase, the mask and key values are set similarly by observing the right side of the table. However, in this cycle, values in the tagged rows are changed. The comment column in the table specifies the run order of this key combination where a NC (no change) means that the given input combination does not alter any content in the CAM. To ensure correct operation, entries must be appropriately ordered while applying the truth table, to avoid corrupting the values. For example, the operation order and truth table order for in-place addition is not the same. The fourth entry is run before the second entry. This is because that if "001" is searched first for Cr, B, A respectively, and B values of tagged rows changed to logic-1, then when "011" is searched as second, the same rows are tagged again and changed twice. Such situations cause erroneous results and can be avoided by correct sequencing.

The operations on the AP are performed by applying the truth table of the function in an ordered sequence to the CAM. To illustrate a complete operation on the AP, Algorithm

Table 2.2: LUT for addition and subtraction

(a) LUT for addition

			In-place			Out-of-place		
Cr	*B*	*A*	*Cr*	*B*	*Comment*	*Cr*	*R*	*Comment*
0	0	0	0	0	*NC*	0	0	*NC*
0	0	1	0	1	2*ndPass*	0	1	1*stPass*
0	1	0	0	1	*NC*	0	1	2*ndPass*
0	1	1	1	0	1*stPass*	1	0	5*thPass*
1	0	0	0	1	3*rdPass*	0	1	3*rdPass*
1	0	1	1	0	*NC*	1	0	*NC*
1	1	0	1	0	4*thPass*	1	0	*NC*
1	1	1	1	1	*NC*	1	1	4*thPass*

(b) LUT for subtraction

			In-place			Out-of-place		
Br	*B*	*A*	*Br*	*B*	*Comment*	*Br*	*R*	*Comment*
0	0	0	0	0	*NC*	0	0	*NC*
0	0	1	1	1	1*stPass*	1	1	1*stPass*
0	1	0	0	1	*NC*	0	1	2*ndPass*
0	1	1	0	0	2*ndPass*	0	0	*NC*
1	0	0	1	1	4*thPass*	1	1	3*rdPass*
1	0	1	1	0	*NC*	1	0	*NC*
1	1	0	0	0	3*rdPass*	0	0	4*thPass*
1	1	1	1	1	*NC*	1	1	5*thPass*

2.1 shows the control flow for an in-place addition of two vectors (A and B) in the AP. The algorithm accepts the LSB and MSB locations of the vectors together with the carry location (C) as inputs and applies the LUT for the in-place addition of these vectors. Figure 2.9 illustrates the corresponding complete flow for the in-place addition of two, 1×4, 4-bit vectors (i.e., $B[i] \leftarrow B[i] + A[i],\ i = 0, ..., 3$) when the algorithm is applied on the A and B vectors. To perform the operation, *AdditionIP(8, 7, 4, 3, 0)* instruction is executed where A and B vectors are between the columns of 3-0 and 4-7 in the CAM and C is in the 8th column. Initially, A contains (i.e., bits 3-0) the values of [6; 4; -5; -1] and B (i.e., bits 7-4) contains the values of [-8; 3; -3; 2] as shown in the upper-left (initial) CAM as binary. The four tables at the top correspond to the LUT for in-place addition (i.e., LUT_AddIP) where the shaded entry indicates the applied LUT entry. The truth tables consist of two parts; *compare* and *write*. The compare part shows the key value that is searched in the specified/masked columns of the CAM and the write part shows the value that is written into the specified/masked columns of the matched rows after the preceding compare operation. Other tables of the figure show the progress in the CAM content together with the key/mask values (in red and green boxes respectively) and the tag status. During the addition, the mask register is set to point the current bit locations on which the addition is performed and the key register is set to the corresponding LUT entry respectively. Then the compare operations are performed by the key value in the columns specified by the mask value. Lastly,

the write operations are performed on the matched rows of the CAM where the value in the write columns of the LUT replaces the old value. To illustrate, in the first CAM, "011" value is searched in the first bits of A and B together with C (carry bit) respectively where there is a match in the third row. In the following write phase, B and carry are changed as logic-0 and logic-1 respectively according to the truth table. This operation corresponds to the single-bit addition of A, B, and carry if A and B are logic-1 and the carry bit is logic-0. As a result of this addition (1+1+0 for A, B, and carry respectively), the C (carry) bit becomes 1 and B (result) becomes 0. In the figure, each row corresponds to the single-bit addition which is completed by applying all entries of the LUT on a masked column. The arrows indicate the flow of the operation where the masked column is shifted to left after each LUT pass. After finishing the operation on the fourth bit (columns 3 for A and 7 for B), the last CAM (i.e., lower-rightmost) shows the result where B becomes [-2, 7, -8, 1].

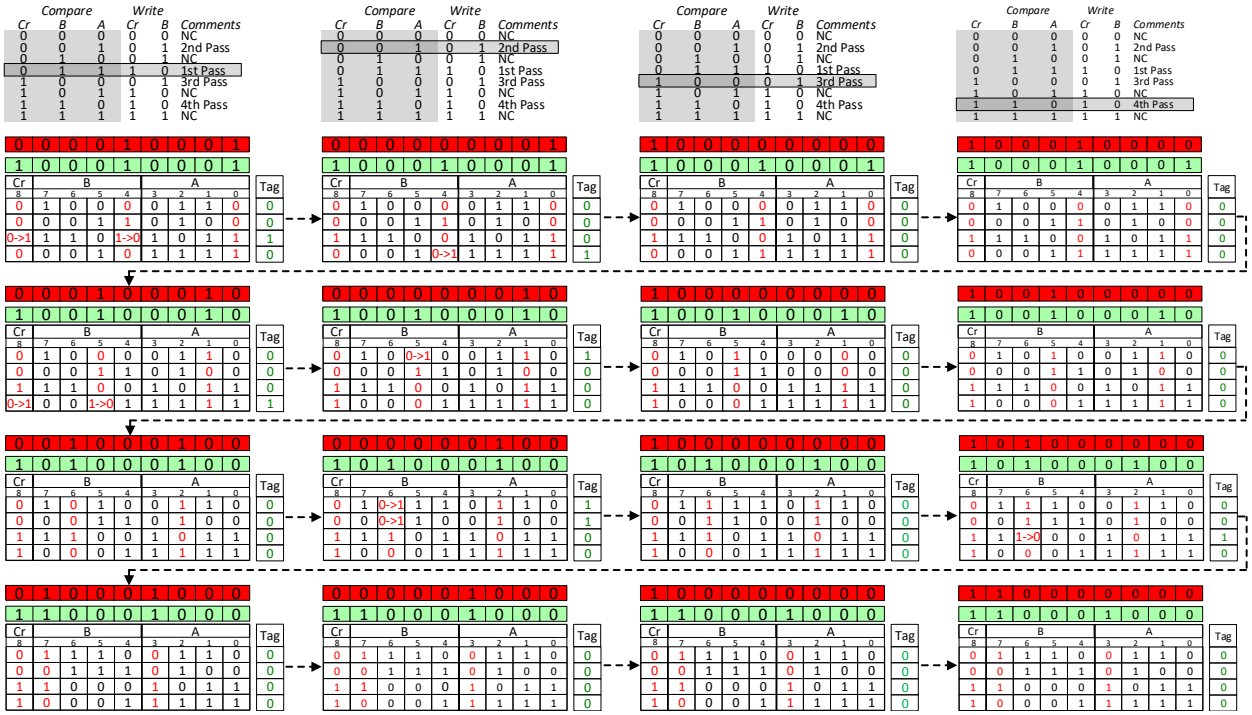


Figure 2.9: Vector Add Operation. The sequence of compare and write operations are shown for a complete vector addition.

Algorithm 2.1: In-place addition (B = B + A) in the AP

```
1  function AdditionIP(C, B.MSB, B.LSB, A.MSB, A.LSB)
2      for i = A.LSB to A.MSB
3          for pass = 1 to 4
4              SetMask(C, B.i, A.i);
5              SetKey(LUT_AddIP(pass), C, B.i, A.i));
6              Compare(Mask, Key);
7              Write(LUT_AddIP(pass), B.i, C);
8          end
9      end
10 end
```

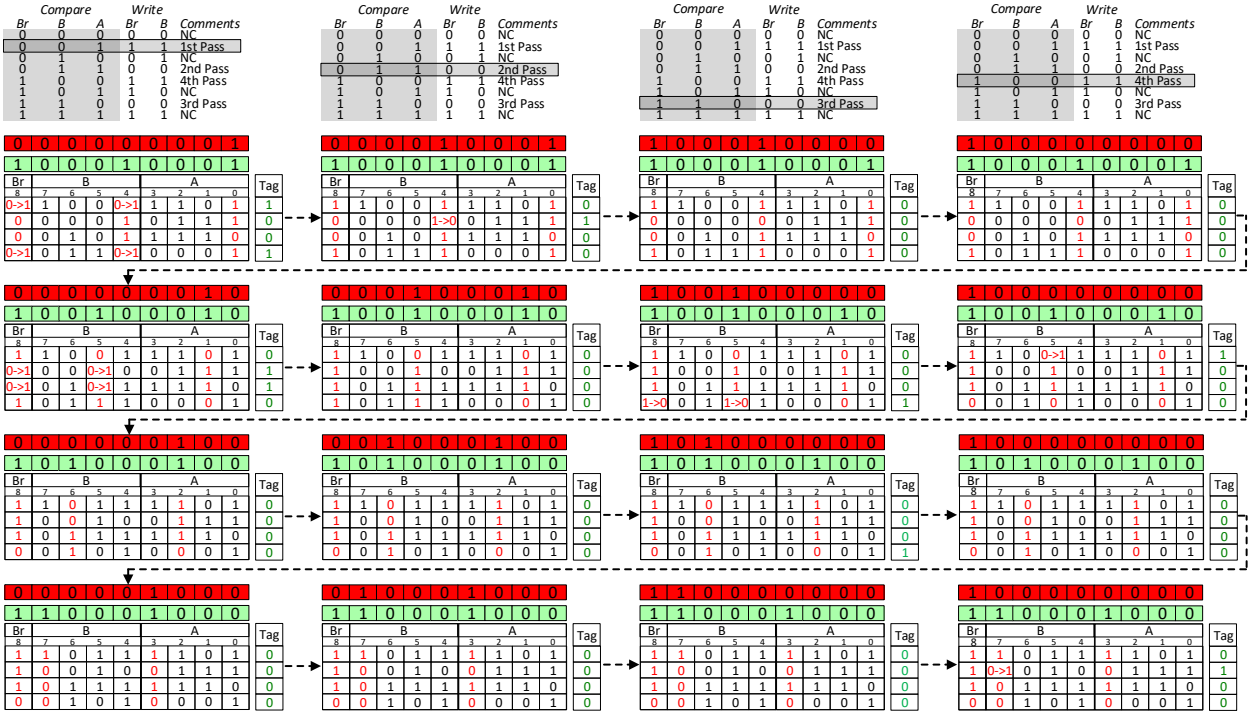


Figure 2.10: Vector subtraction operation on 4-bit four number pairs. The sequence of compare and write operations are shown for a complete vector subtraction.

In out-of-place addition, the sum of the inputs A, B are written into R. Before the addition, all bits of R are assumed to be logic-0 to minimize the number of cycles, by avoiding NC rows in the truth table. Due to the reuse of the B location in in-place addition, it requires fewer cycles than out-of-place addition. In both methods, the controller unit of the AP applies the truth table on each bit of the inputs and carry in order.

For the subtraction, Table 2.2b shows the LUT for in-place and out-of-place subtraction ($B = B - A$) in the same manner as addition, but, in this case, the borrow bit (Br) is used instead of the carry bit. Alternatively, subtraction can also be implemented by using addition in 2's complement representation; the complement of the subtrahend is added to minuend. However, this method comes with an additional area and time cost as detailed in the following section.

After that, the values specified in the Write columns of the LUT is written to the indicated columns of the matched rows. For example, in the first step, 001 is searched in the Br and the first bits of A and B (column 8, 4, and 0 respectively). There is a match in the first and fourth rows, so the B and Cr values of these rows are changed as logic-1 in the following write phase.

Similarly to Figure 2.9, Figure 2.10 shows the step by step execution of in-place subtraction (B = B − A) of two 1×4 4-bit vectors, A and B, i.e. $B[i] \leftarrow B[i] - A[i], i = 0...3$. The LUT for in-place subtraction has four entries in a specific order required to perform the operation correctly and mainly performs the single bit full subtraction. To perform the operation, *SubtractionIP(8, 7, 4, 3, 0)* instruction is executed where A and B vectors are between the columns of 3-0 and 4-7 in the CAM and Br (borrow) is in the 8th column. Initially, A contains (i.e., bits 3-0) the values of [-3; 7; -2; 1] and B (i.e., bits 7-4) contains the values of [-8; 1; 5; 6] as shown in the upper-left (initial) CAM. The subtraction is done in a bit-serial, world parallel mode, that is, a bit-wise subtraction is performed on each row of Figure 2.10. The mask locations are shifted left (except Br) at each row. This process is repeated for every bit position and for every LUT entry. Since there are four LUT entries and four bits, the total operation takes 16 steps. Finally, the value stored in B becomes [-5; -6; 7; 5] which is equal to B-A (i.e., [-8-(-3); 1-(7); 5-(-2); 6-(1)]).

The addition and subtraction operations take a total of 16 cycles irrespective of the number of A-B pairs. For the in-place addition and subtraction, m steps are needed, where m is the

number of bits per pass with a total of 4 passes per step. In the out-of-place addition or subtraction, m steps of five passes each is needed.

Absolute Value and 2's Complement

Absolute value and two's complement operations are very fundamental operations for FFT and many other algorithms. To find the 2's complement of a number, Table 2.4 is used. In the table, F stands for the flag and is stored in the CAM as a temporary value similar to carry and borrow in addition and subtraction.2's complement operation must be out-of-place; thus the result must be written to another place. During the operation, F becomes logic-1 at the first logic-1 bit of the input number (A) and after that, the truth table complements the A's remaining bits. This operation takes m steps of 3 passes.

The LUT in Table 2.5 shows the truth table for absolute value operation. This table is the revised version of the absolute value's LUT where the table does not perform 2's complement operations if the number is positive. In the table, S stands for sign bit and represent the sign of the number i.e. its most significant bit. If the sign is positive (logic-0), this truth table has no effect and simply copies the content of input A to result R. On the other hand, if the sign is negative (logic-1), truth table performs the 2's complement operation. This operation takes m steps where each step consists of 4 passes as shown in the table. Additionally, this absolute value truth table can also be used to revert a number to its original sign. For this operation, the original sign of the number is stored in a location and during the absolute value operation, this sign can be given as sign value instead of the real sign (last bit) of the number. This type of usage is exploited in signed multiplication of the numbers as detailed in the next section.

Table 2.3: LUT for multiplication

Cr	R	B	A	Cr	R	$Comment$
0	0	0	0	0	0	NC
0	0	0	1	0	0	NC
0	0	1	0	0	0	NC
0	0	1	1	0	1	$2ndPass$
0	1	0	0	0	1	NC
0	1	0	1	0	1	NC
0	1	1	0	0	1	NC
0	1	1	1	1	0	$1stPass$
1	0	0	0	0	1	NP
1	0	0	1	0	1	$3rdPass$
1	0	1	0	0	1	NP
1	0	1	1	1	0	NC
1	1	0	0	1	0	NP
1	1	0	1	1	0	$4thPass$
1	1	1	0	1	0	NP
1	1	1	1	1	1	NC

Table 2.4: LUT for 2's complement

F	A	F	R	$Comment$
0	0	0	0	NC
0	1	1	1	$3thPass$
1	0	1	1	$1stPass$
1	1	1	0	$2ndPass$

Table 2.5: LUT for absolute value

F	S	A	F	R	$Comment$
0	0	0	0	0	NC
0	0	1	0	1	$1stPass$
0	1	0	0	0	NC
0	1	1	1	1	$4thPass$
1	0	0	0	1	NP
1	0	1	1	0	NP
1	1	0	1	1	$2ndPass$
1	1	1	1	0	$3rdPass$

Multiplication

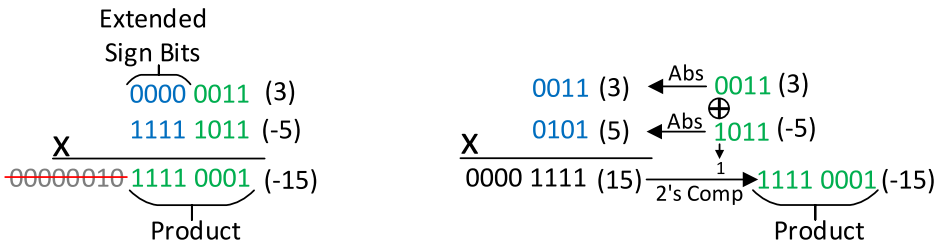
In unsigned multiplication ($R = A \times B$), the LUT shown in Table 2.3 is applied to CAM for each combination of input bits. In other words, LUT is applied to all bits of B for each bit of A. Indeed, this table performs the addition operation between B and R if the A's bit is logic-1. In the table, NP indicates the combinations that are not possible and they are also disregarded like NCs. By contrast to linear operations such as addition and absolute value, the multiplication operation is a quadratic operation. As an example, Figure 2.12 illustrates the complete flow for the unsigned multiplication of two 1×4 2-bit vectors, A and B, i.e. $B[i] \leftarrow B[i] \times A[i], i = 0...3$. The pseudo-code in Algorithm 2.2 shows the controller flow for the unsigned multiplication operation. For each bit of A, position of the carry is changed in accordance with current partial addition. While outer loop scans each bit of A, inner loop scans bits of B and performs partial addition. To perform the operation, *Multiplication(R, B, A)* instruction is executed where the parameters corresponds to the column numbers of the corresponding vectors (e.g., A is columns between 1-0). Initially, A contains (i.e., bits 1-0) the values of [3; 3; 2; 0] and B (i.e., bits 3-2) contains the values of [2; 3; 1; 2] as shown in

the upper-left (initial) CAM. Each row of the figure shows a partial multiplication operation between the single bits of A and B, and the partial addition of the result (e.g., the second row shows the multiplication between the second bits of A and the first bits of B together with its addition with the previous result came from the first row). The process is repeated for all the passes in the prescribed order shown in Figure 2.12. Finally, the value stored in R becomes [6; 9; 2; 0] which is equal to B×A (i.e., [2×3; 3×3; 1×2; 2×0]). Thus multiplying two vectors that are 2-bit wide takes 16 steps ($4m$ compares and $6m$ writes), independently of the vector size.

Algorithm 2.2: Unsigned multiplication ($\mathbf{R} = \mathbf{B} \times \mathbf{A}$) in the AP

```
1  function Multiplication(R, B, A)
2      for i = B.LSB to B.MSB
3          Set Cr
4          for j = A.LSB to A.MSB
5              Set R.k
6              for pass = 1 to 4
7                  SetMask(Cr, R, B.i, A.j);
8                  SetKey(LUT_Mult(pass), Cr, R.k, B.i, A.j);
9                  Compare(Mask, Key);
10                 Write(LUT_Mult(pass), Cr, R.k);
11             end
12         end
13     end
14 end
```

Additionally, *Multiply and accumulate* operation ($R = A \times B + B$) can be performed by performing an in-place addition following the multiplication. However, in this case, the R must be initialized as B before the operation starts.



(a) Sign extension (b) Sign conversion

Figure 2.11: Signed multiplication methods on the AP

For signed multiplication, two ways can be used in APs. The first one is sign extension method. In this method, sign bits of the inputs are extended to the number of bits in the result and then these numbers are multiplied. After, the most significant digits of the multiplication are discarded, the remaining ones become the product. Figure 2.11a exemplifies the sign extension method in multiplication. In the second method (see Figure 2.11b), firstly, absolute values of two inputs are computed.This operation can be performed as out-of-place by using Table 2.5. After that, the absolute values are multiplied by the unsigned multiplication function. This result is the absolute value of the real product. At the end, the absolute value of the result is converted to its original value by using the same absolute value function. During the conversion, sign bit is taken as the XOR of sign bits of inputs A and B. If the result is positive, the converted number remains the same, otherwise it is complemented.

The most commonly used operations in many applications are multiplication, addition, and subtraction. For this reason, we omitted the detailed description of the division algorithm, which can be easily derived from successive subtraction and mask shift. The expected complexity of such operation is $O(m^2)$.

2.4 System Architectures

Ideally, the researchers would like to consider general purpose architectures where the whole memory and permanent storage are combined in very large Resistive Random Access Memory (ReRAMs) structures without having to rely on SRAM, DRAM or Flash storage. This however will not be possible in the near future, therefore AP technology can be considered for accelerators. Accelerators do not exist in vacuum. They need to interface to CPUs and traditional memory and storage structures. The modality and structures of these interfaces are driven, to a very large degree, by a set of target applications. To that effect various

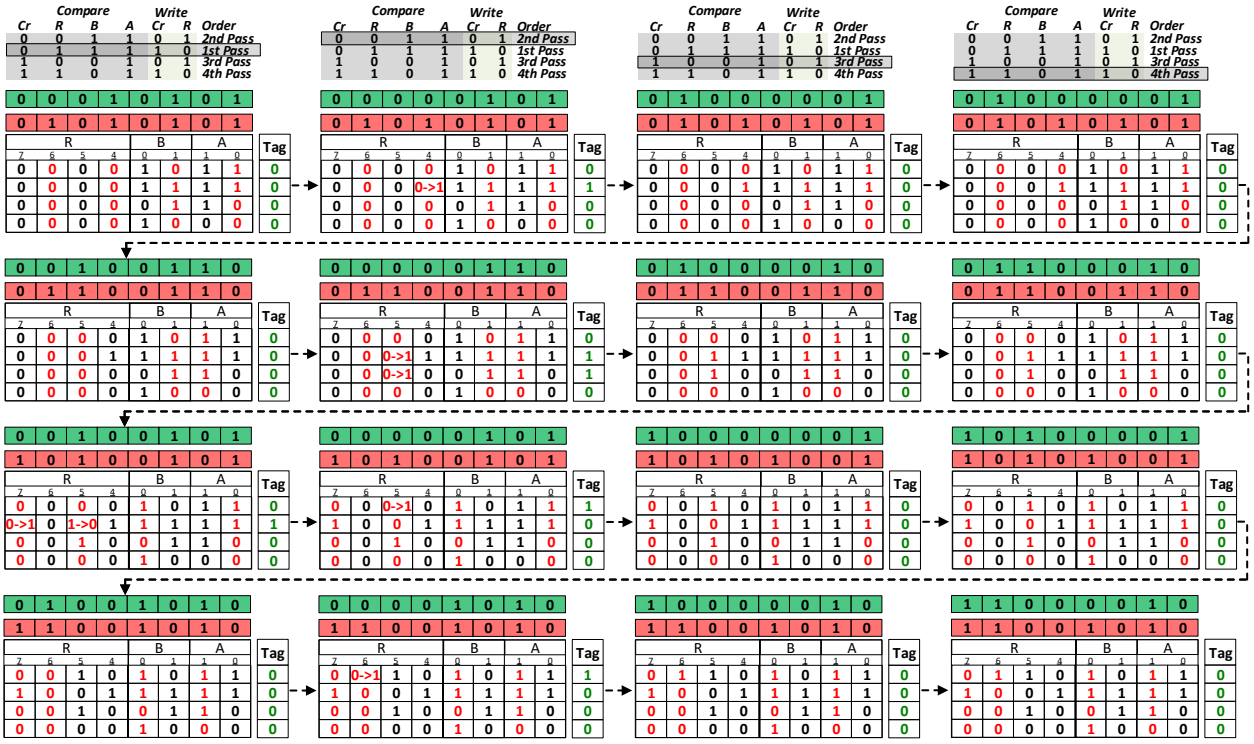
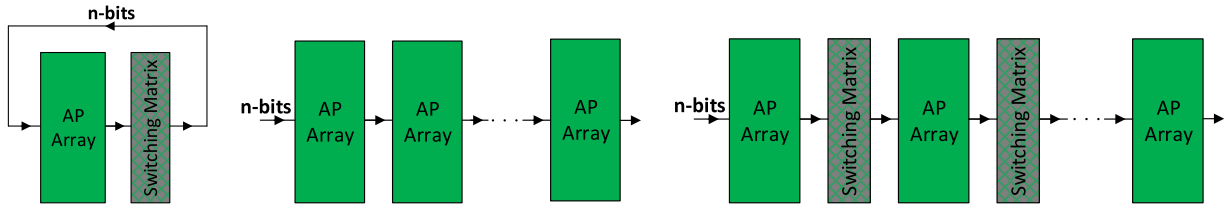


Figure 2.12: Vector multiplication operation. The sequence of compare and write operations are shown for a complete unsigned vector multiplication.



(a) Feedback (b) Hardwired Pipeline (c) Reconfigurable Pipeline

Figure 2.13: System-level AP Architectures

heterogeneous in-memory accelerator architectures must be designed and evaluated. This evaluation must focus on the achievable throughput as well as the energy and power. Equally important is the programmability of these architectures: the ease of creating compilation tools that can be used by application developers

On its own, associative processors are very efficient in terms of parallel processing. However, today's many performance demanding task requires a quite degree of communication through the processing. As an example, FFT requires a very good parallelism while performing the

butterfly operation in a stage. On the other hand, a communication and data exchange are required before the next butterfly stage. Therefore, APs must be connected to each other to provide an efficient communication not to harm the obtained gain from the parallelism. Furthermore, this communication must be configurable if the processor supports different kinds of tasks having different communication patterns. Reconfigurable associative processing is achieved via a combination of interleaved sets of programmable CAMs and programmable or fixed crossbar arrays (interconnection matrix/switch) as building blocks.

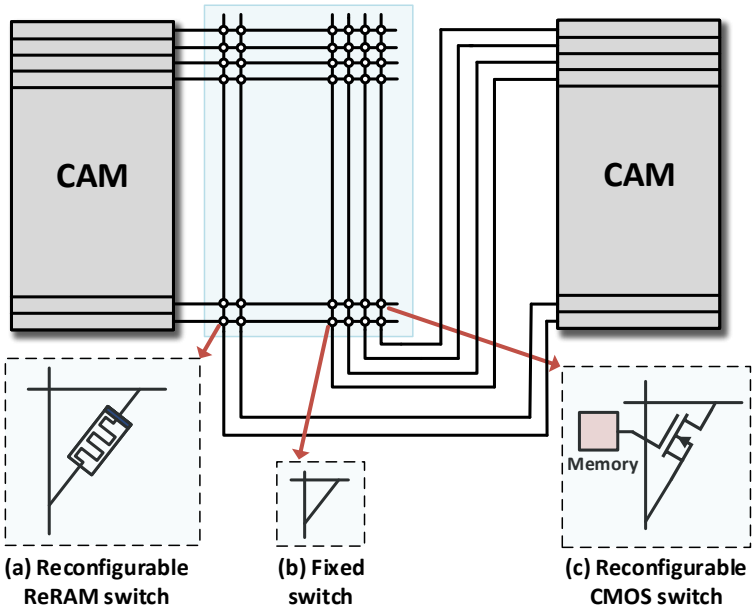


Figure 2.14: Interconnection matrix between the two CAMs

For the implementation of crossbar arrays, Figure 2.14 depicts the three different kinds of implementation for the interconnection matrix where the two APs are connected to each other as row-wise through the interconnection matrix. For the fixed type of connection (see Figure 2.14b), simply a wire connection exists at the cross point that connects a row of the left CAM to another row of the right CAM. For the reconfigurable switch, the two methods can be followed. In the traditional method, the output of an memory cell controls the gate of an transistor which is used to either allow or block the communication at a cross point [84]. This methodology is very widespread in the modern FPGAs where the FPGA bit-stream includes the configuration data of the switching matrix [135, 182, 30]. A

logic-0 stored in the cell corresponds to the open communication between the column and the row crossed at that point, and a logic-1 corresponds to the closed path where the two rows are connected to each other. Since this switching matrix requires a single memory element at each cross point, it becomes unfeasible for the huge CAMs. For an N-row APs, the cost of the switching matrix becomes N^2. In addition to area and energy costs of a such big matrix, its configuration requires lots of configuration bits which is very costly to control. For example, to maintain a full connection flexibility between 2 CAMs with 512-rows, a 512x512 switching matrix is required which occupies more area than the CAM itself generally. Therefore, in practice, a group of rows are connected to another group through the interconnection switch and thus provides a partial/limited connection flexibility. An example is the interconnection fabric inside the FPGAs which provides connection only towards to the neighbor logic blocks [65, 97]. The invention of the memristor has been provided a low cost alternative for interconnection matrix as it was in the implementation of CAM arrays [169, 26, 86]. The cross-bar nature of ReRAM arrays makes it possible to implement very cheap cross-bar switching enabling extreme flexibility in interconnecting ReRAM CAMs together to realize more complex functions. For the memristor based high-density crossbar implementation, interconnection matrix are built by replicating the memristor memory cell as described in Figure 2.14b in two dimensions, forming a programmable crossbar array. This cells are named as gateless cells where each memory cell is a single memory device located at the intersection of two lines as shown in the figure, provide the highest density. A logic-0 (R_{off}) at an intersection point closes the connection while a logic-1 (R_{on}) opens it. Once programmed, the crossbar will retain the routing patterns until re-programmed unlike the CMOS-based interconnection switch. By programming the crossbar arrays it is possible to realize different connection schemes between the RCAMs or the same RCAM. A control processor programs the RCAMs and crossbar switches, and provides the sequencing of the operations performed on the RCAMs, as well as managing the data I/O. One of the drawbacks of this architecture is the abundant availability of paths for sneak currents that

flow through the memory cells parallel to the desired one and significantly impact the readout operation. This problem is resolved by establishing only one connection per row/column. The crossbar settings of each AP are determined by the controller that also determines the sequencing of the operations performed on the CAMs and manages the data I/O. To demand all of these necessities, three different kind of system-level AP architecture are envisioned as seen in 2.13 and detailed as follows:

- *Feedback Pipeline Architecture:* (Figure 2.13a) In this system, the architecture consists of an AP connected to itself via a programmable crossbar. This configuration is suitable for applications that consist in compute/shuffle/compute sequences and where area constraints are more important than performance targets.

- *Hardwire Pipelined Architecture:* (Figure 2.13b) is suitable for tasks of high recurrence and a fixed number of inputs (e.g. FFT, CNN, etc.). In these cases, a configurable crossbar is unnecessary since there is a fixed connection between the APs exists. This architecture supports high throughput at the cost of reduced flexibility.

- *Reconfigurable Pipeline Architecture:* (Figure 2.13c) provides both flexibility and performance albeit with higher area and energy costs. It supports the dynamic reconfiguration of the pipeline by changing the interconnection patterns at runtime.

All these system architectures exhibits different characteristics and have a trade-off between them. If area and resources are a constraint, then implementation (a) is a possible choice as it uses only one stage implementation with feedback. If a task is of high recurrence with a fixed number of inputs (e.g. FFT) is considered, then implementation (b) is most appropriate, where each stage in the hardwired pipeline performs a butterfly and the data is transposed by wiring to the next stage. If flexibility and performance are more important than the area constraint then implementation (c) would be best as it allows for the change of interconnection patterns efficiently and reconfiguring the pipeline dynamically as needed.

Table 2.6: Reduction tree and switching matrix comparison

	The proposed	**[40]**	**[42]**	**[176]**
Type	Switching Matrix (Memristor)	Reduction Tree (CMOS)	Reduction Tree (CMOS)	Reduction Tree (CMOS)
Functional Capability	**Arithmetic Operations**, Priority Selection, and Population Count	**Only** Population Count and Priority Selection	**Only** Population Count, Priority Selection, and Addition	**Only** Addition (including Population Count)
$C_{PopulationCount}$	$\mathcal{O}(\log_2^2 n)$	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(\log_2 n)$
$C_{ArrayAddiition}$	$\mathcal{O}(\log_2 n)$	NP	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(\log_2 n)$
$C_{ArrayMultiplication}$	$\mathcal{O}(\log_2 n)$	NP	NP	NP
$C_{MatrixMultiplication}$	$\mathcal{O}(n \log_2 n)$	NP	NP	NP

A key consideration in the proposed architectures is the use of flexible memristor interconnection crossbar arrays instead of conventional CMOS-based reduction tree as in [176, 177, 43], thus improving both the utility of the AP and the area to performance and power metrics. In these system architectures the switching matrices can perform the functionalities of both reduction trees and interconnection network. In previously proposed architectures, the reduction tree implemented in traditional CMOS is detrimental for area to performance and power metrics. Table 2.6 compares the switching matrix (our study) and reduction tree exploited in previous studies. As shown, the memristor based switching matrix provides better functionality since reduction trees do not allow for intercommunication between the AP rows. Due to the rigid structure of reduction trees, certain operations cannot be implemented on traditional structures and are denoted by Not Possible (NP). This is not an issue for the proposed AP architectures due to the flexibility of the switching matrix. The switching matrix enables the reordering of data inside the CAM (reconfigurable feedback) or between the CAMs. Even though the switching matrix is implemented by memristors in this study, the same approach could be applied to [40] by using STT-MRAM for the interconnection matrix and it would provide similar advantages for both functionality and area.

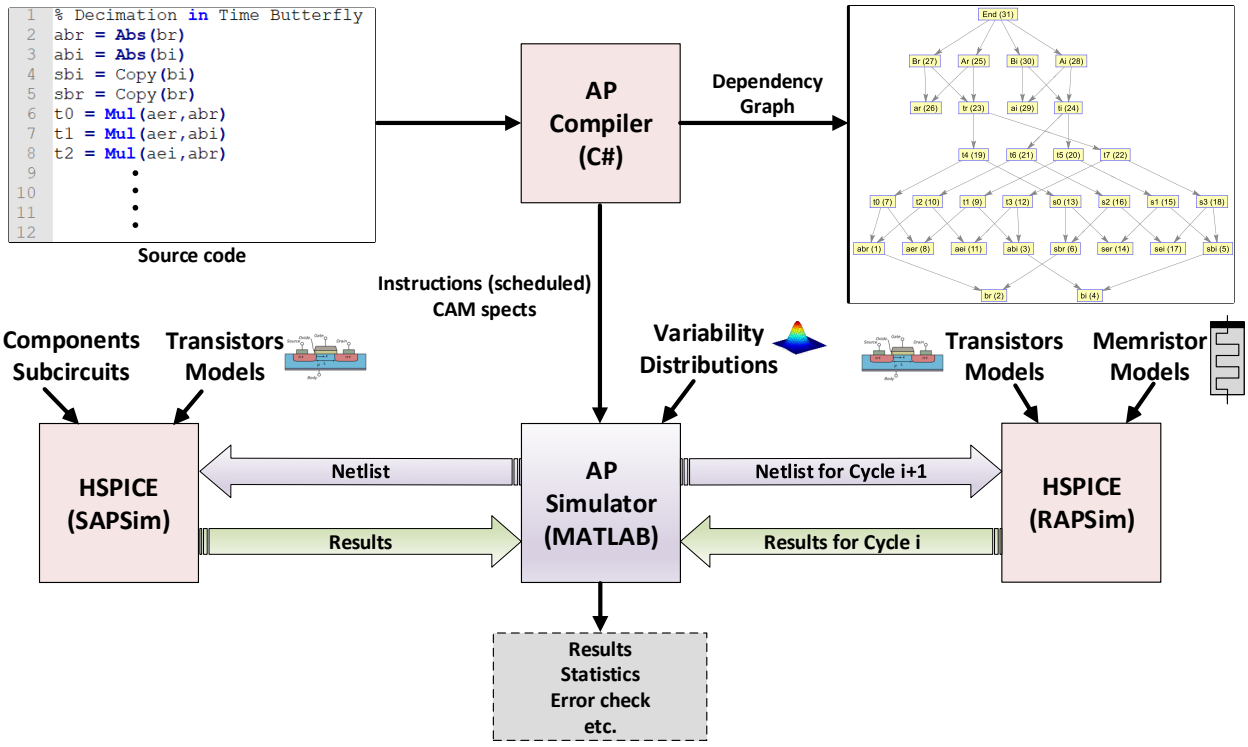


Figure 2.15: The simulation framework

2.5 Simulator

Figure 2.15 shows the realized simulator framework for the proposed AP architectures. A simulation on the AP starts with coding the required task by specific AP instructions. The source code is compiled by a basic compiler written in C#. During the compilation, the source code is scheduled and converted to a description that Matlab simulator can interpret. The compiler also determines the results for the CAM specifications such as size, length etc. At the same time, the compiler is capable of supporting visualization of the source code by generating dependency graphs. After compilation, the Matlab simulator can perform the simulation individually for both RAP and SAP architectures. The user has the option to choose the referenced architecture and technologies. After that, the simulator can perform both system-level and circuit-level simulations at the same time in the MATLAB and the HSPICE respectively. As parameters, the circuit level simulator accepts circuit models (i.e., transistor, memristor), sub-circuits (e.g., sense amplifiers), CAM features (width and

height), sweep parameters, initial data, and the instructions. Then the MATLAB part of the simulator creates netlists that iteratively drive the HSPICE simulators. In here, the simulation for the RAP is performed as cycle accurate and temporal results between the cycles can be evaluated. On the other hand, for the SAP, a single netlist file is generated that corresponds to the whole simulation period. This is because that the SAP circuit consists of volatile memories, that is, if the simulation is disrupted before finishing, all temporal status of the CAM cells are lost.

Chapter 3

Tradeoffs in APs

In this chapter, tradeoffs of APs are discussed in detail to form the basis of a new class of in-memory accelerator. The performance, energy, and reliability aspects of the associative processors are inspected. Furthermore, solutions are proposed for the vulnerable points such as memristor variability and reliability issues.

Even though, this chapter provides the introductory information about the tradeoffs, the other chapters provides much more detailed information and comparison in the aforementioned aspects.

3.1 Introduction

The research on in-memory computing paradigms aims to overcome the barrier of memory bottleneck by processing the data inside the memory without moving it back and fort between the memory and the processor. The paradigm proposes replacing the logic with memory structures, virtually eliminating the need for memory load/store operations during computation as discussed in Chapter 2. To this point, *Associative processing* (AP) seems as

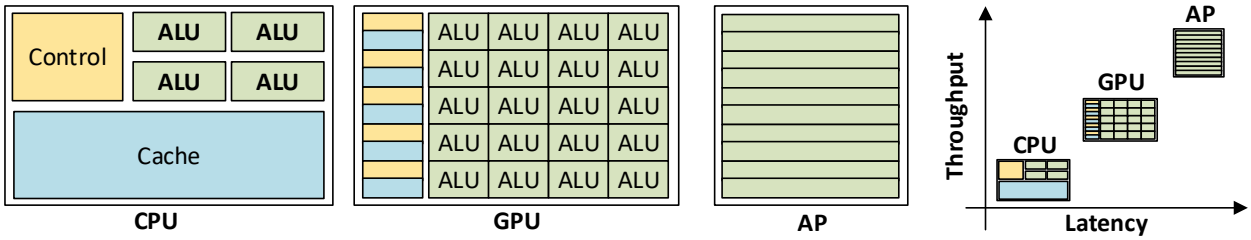


Figure 3.1: The comparison of CPU, GPU, and AP architectures

a very promising computational paradigm for in-memory computing. However, an in-depth analysis of associative processing paradigm must be evaluated to find its optimal usage areas.

3.2 Performance

In terms of performance, the APs requires high amount of data to outperform the traditional processors. For this reason, their best usage is in the computational benchmarks that has an inherent SIMD computational pattern. In an analogy, APs can be considered as a next step on the path of the CPU (central processing unit) to GPU (graphical processing unit) transformation. When compared with CPUs, GPUs have simpler processing cores, however, they are throughput optimized on the contrary of CPUs which are latency optimized. When compared GPUs, APs have much simpler cores (i.e., a single memory row), so its throughput becomes more than GPUs since more cores can be placed in a chip. On the other hand, its overall controller is very small and simpler that includes the LUTs for the operations and a basic logic that applies these LUTs to the key and mask registers in order. As an example, for an FFT operation on 1K input, the controller of the circuit corresponds to less than 2% of the overall AP area. Figure 3.1 illustrates the architectural comparison of these three processors. Figure 3.1 illustrates the architectural comparison of these three processors.

Table 3.1 shows the runtime (total number of passes), area usage per CAM row (in terms of bits), and algorithmic complexities of each arithmetic and logical operations in the AP. As

Table 3.1: Running time and area evaluation of primitive AP operations/instructions

Function	Runtime	Area/Row	Complexity
NOT	$2m$	$2m$	$\mathcal{O}(m)$
AND	$2m$	$3m$	$\mathcal{O}(m)$
OR	$6m$	$3m$	$\mathcal{O}(m)$
XOR	$6m$	$3m$	$\mathcal{O}(m)$
Addition (IP, S/U)	$10m$	$2m+1$	$\mathcal{O}(m)$
Addition (OOP, S/U)	$11m$	$3m+1$	$\mathcal{O}(m)$
Subtraction (IP, S/U)	$10m$	$2m+1$	$\mathcal{O}(m)$
Subtraction (OOP, S/U)	$11m$	$3m+1$	$\mathcal{O}(m)$
2's Complement	$6m$	$2m+1$	$\mathcal{O}(m)$
Absolute Value	$8m$	$2m+1$	$\mathcal{O}(m)$
Multiplication (U)	$10m^2$	$4m$	$\mathcal{O}(m^2)$
MAC (U)	$10m^2+10m$	$4m$	$\mathcal{O}(m^2)$
Multiplication (S)	$10m^2+4m-14$	$8m+4$	$\mathcal{O}(m^2)$

* **IP:in-place, OOP:out-of-place, S:signed, U:unsigned, m:bitwidth.**

stated in the table, the runtime of a vectorial operation on m-bit n numbers depends only on the number of bits (m), not on the number of vectors (n). To exemplify, if the number of vectors in Figure 2.9 from Chapter 2 was 1024 instead of 4, the total run time would be the same.

Figure 3.2 shows the performance results of the all operations separately in AP when the compare and write times are assumed as single cycle. The figure shows the total number of execution cycles with respect to the bit width of the operands. As inferred from Table 3.1, the multiplication operations are more sensitive to the bit width since its complexity is squared. Figure 3.3 compares the some 16-bit AP operations with a sequential processor. As shown in the figure, the AP outperforms the single processor after a threshold value of vector size. Moreover, the data moving costs (e.g. data access, cache misses) imposes additional overhead to single and vector processors for which AP is cost-free and they are not taken into account for the sequential processors. On the other hand, all data in the AP are processed in-place so that there is no need for data moving operations.

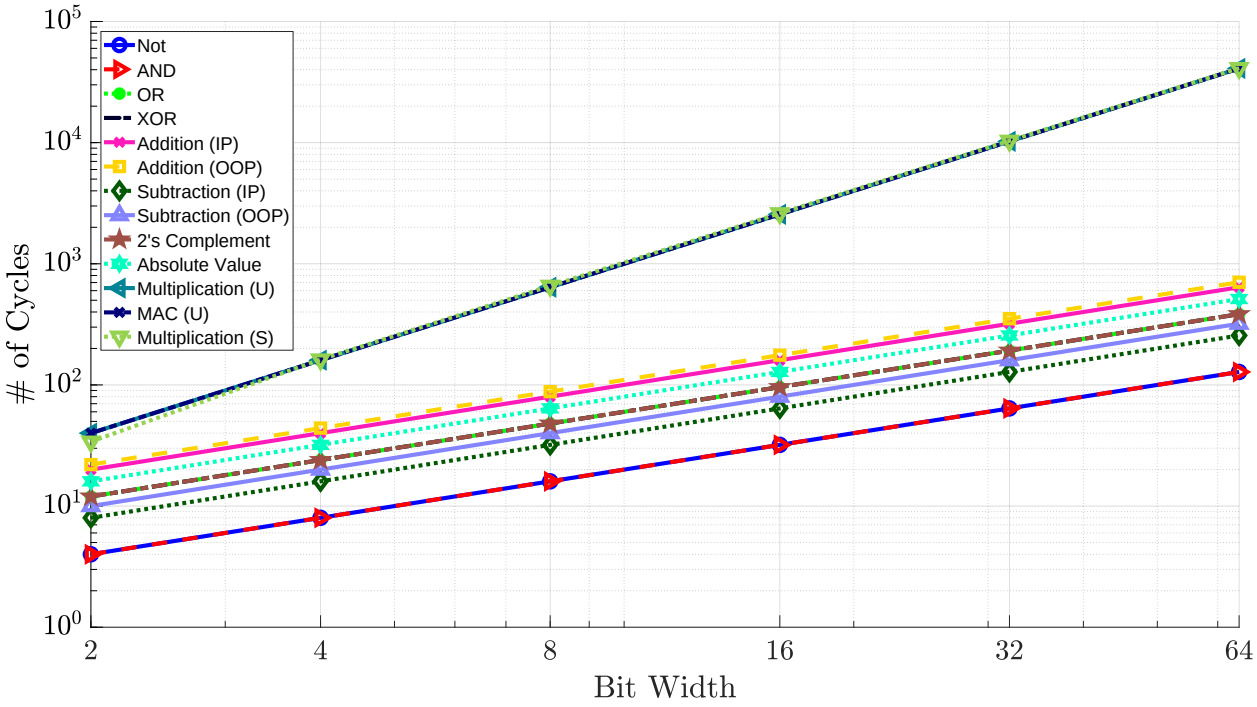


Figure 3.2: The comparison of operations on the AP

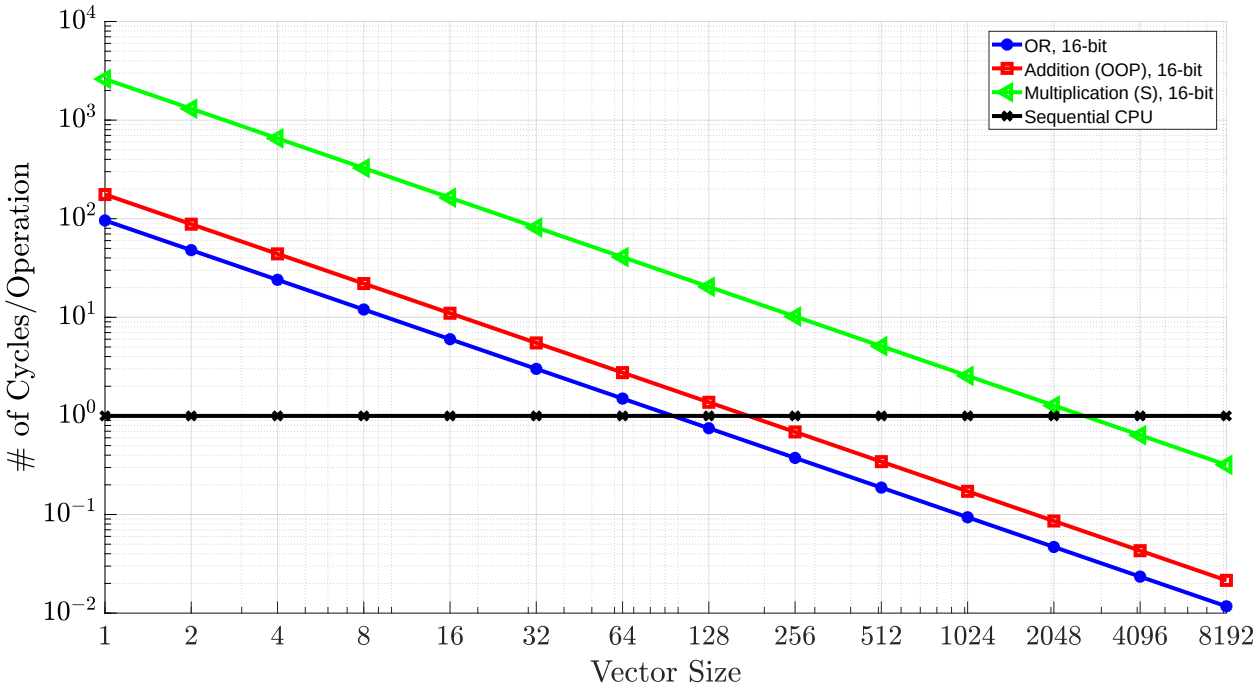


Figure 3.3: The comparison of a sequential processor with some AP operations on 16-bit operands

3.3 Energy

Energy consumption on the APs highly dependent on the technology and parameters referenced in the circuit design. As an example, SRAM technology used in the CAM cell implementations are quietly different from the memristor technology and these difference has a big impact on the energy consumption. For this reason, in this subsection, the comparison between the SAP and RAP implementations are performed by using the nominal circuit parameters and keeping the other circuit parameters as constant for both. In the table, the actual energy consumption for compare operation is also depends on the residual charge coming from the previous cycle. For this reason, the reported consumption corresponds to the maximum consumption where there is no initial charge across the capacitor.

The two archtectures also differs in terms of nonvolatility which has a significant effect in terms of energy. When the system is powered down traditional SRAMs require refresh to load program data or to backup contents. This is eliminated in memristors allowing for finer grain power shutdown and/or voltage scaling. When not accessed, a ReRAM bank can be powered off. This is particularly attractive when a large memory space is banked as leakage power can be significantly reduced compared to SRAMs.

A distinct advantage of APs is that the architecture can almost seamlessly evolve from today's SRAM and ReRAM implementations to tomorrow's emerging storage technologies, possible more energy efficient ReRAM implementations by simply transplanting the memory cells to achieve higher scalability and performance while maintaining the same top level system architecture and the corresponding investment in software. The detailed comparison with other architectures (i.e., state of the art processors) are also provided in the following subsequent chapters.

(a) SAP cell

V_{cdd}	P_{static}	T_{write}	E_{write}	$E_{compare}$
0.7 V	0.52 µW	0.5 ns	0.24 fJ	5.425 fJ

(b) RAP cell

R_{on} (Ω)	R_{off} (Ω)	T_{write}	E_{write}	$E_{compare}$
100	100 k	2 ns	21.7 pJ	4.908 fJ

Table 3.2: Energy consumption results for both SAP and RAP cells

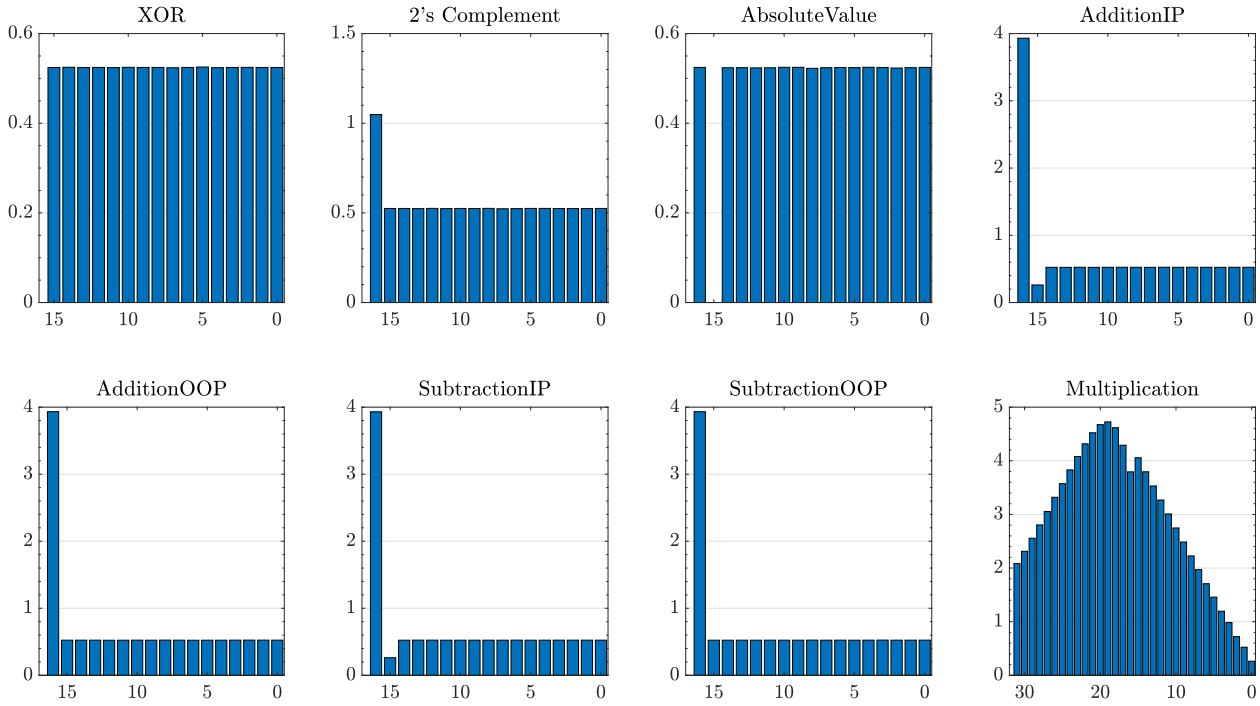


Figure 3.4: Column write density of the fundamental operations on the AP

3.4 Reliability

3.4.1 Write Endurance

The most important drawback of the newly proposed memristors was the switching endurance which negatively affects the applicability of this technology into the applications that requires frequent writing to the memory (e.g. meromorphic computing) [158]. Even though, the first memristor devices exhibit an endurance of around 10^6 [16], later the two studies on the TaOx-based memristances in [172] by HP and in [100] from Samsung demonstrated endurances of 10^{10} and 10^{12} respectively. The desired endurance of the memristor

devices is 10^{15} which is the order of endurance of the traditional memories (Table 1.1) [98]. Even though this is a problem that needs to be addressed in the device level [95], there are many studies proposed by the researchers to compensate the endurance deficiency of the memristors [88, 16, 130] which positively improves the endurance. In addition to these studies, in this section, the endurance of the memristors specifically under the associative computing is evaluated and some methods are proposed for reliability improvement.

The associative processing requires consecutive compare and write cycles to process data inside the CAM, thus requires nearly a write cycle after every compare cycle. Even though all rows participate to the compare (read) operation, the only rows who matched during the compare are written. This situation decreases the number of writes considerably because only a sub portion of the cells are written. For example, the probability of an LUT entry of in-place addition (Table 2.2a) is matched with the content of the row is 0.125 if the data is assumed as random. As a result, only 1/8 of the cells are written. This ratio decreases to 1/16 in the multiplication operation. In addition, after the iteration of a LUT, the location of the result bit is changed so the number of the writes on the RCAM cells are distributed. On the other hand, some bit locations such as carry bit in the addition can come over more writes than the others. In such cases, the columns that are written relatively more frequently than the others can be analyzed and a load balancing between the RCAM rows can be applied to diminish the effect of write endurance.

Figure 3.4 shows the column write density as the results of some logical and arithmetic operations on the RCAM. To obtain the results, the corresponding operations are performed on 1M (2^{20}) 16-bit operands or pair of operands which are randomly generated. The total number of writes to each column is profiled. The figures shows the column numbers in the x-axis (from MSB to LSB) and the total number of writes in the y-axis (in terms of millions). In the figure, it is explicit that the some bits are exposed to more writes than the others. In XOR and absolute value, the write density seems as equally distributed. Only

15th bit in the absolute value has no write applied since it is the sign bit which is already logic-0 since the resulted number is positive. On the other hand, in addition, subtraction and 2's complement, more write operations are performed on some specific bits than the others. These bits correspond to the carry and borrow bits for addition and subtraction respectively, and flag bit in 2's complement. The multiplication operations shows a different pattern than the others where the bits in the middle faces with more writes than the others and the distribution exhibits a pattern like Gaussian.

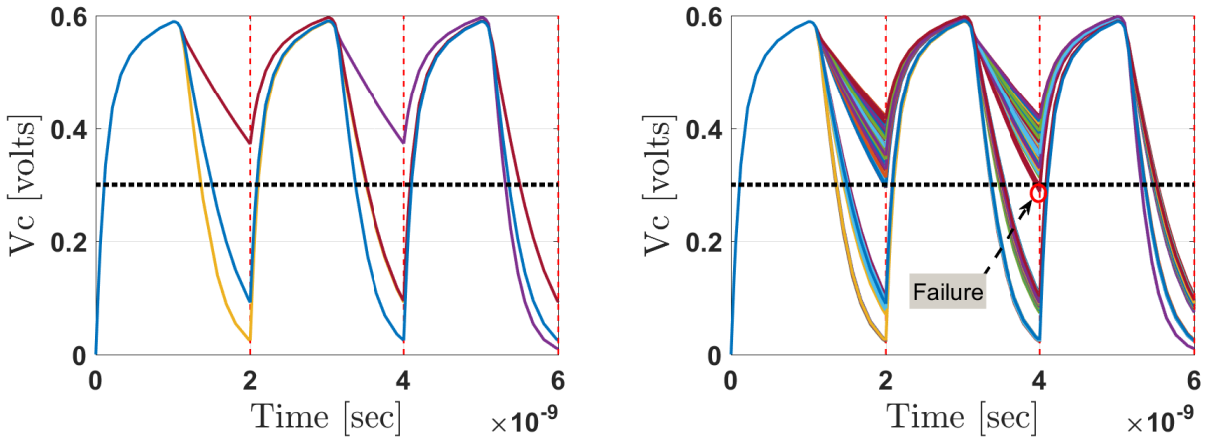
In associative processing, the operations are performed as bitwise where the data does not have to be stored adjacently. For example, a 16-bit number is stored in columns 0-15 while carry bit can be stored in another column rather than column 16. Unlike the number representations in traditional memory where the data is stored in a row or group of adjacent rows, the numbers are represented as vectorial as a combination of columns. This situation provides flexibility to address even bits of the vectorial data. As a result, if the density of the operations is known as a priori, a smart algorithm can extend the life-time of an AP by equally balancing the write density through the columns. As an example case study, the endurance of the APs performing out of place addition operation can be inspected. During the addition, the average writes per the results bit are 0.25 writes/cell (see Figure 3.4). On the other hand, this number becomes 3.75 writes/cell for the carry bit, so it poses a thread for the reliability of the RCAM. If endurance of memristor is taken as 10^{15} (as stated in [45, 28]), the carry (i.e., that is, overall AP operation) bit fails after 1.96 years. If the carry bit is relocated to balance the write density through the result bits, the lifetime of the AP increases to 7.94 years without using any additional extra columns. This corresponds to a more than 3x increase in the life time. In a similar manner, the same methodology can be applied to other arithmetic operations regarding their write density patters and a considerable advantage in the life time can be obtained.

3.4.2 Process Variations

The resistance of a memristor depends on the history of current flux that had previously flowed through it. Electrical characteristics of memristors are mainly determined by the material characteristics of what they are made of and the fabrication processes by which they are made. Like any other nanodevices, memristors also suffer from process imperfections [27]. Due to the fabrication processes, memristors can have a large variance in their low resistance state (LRS, R_{res}, or R_{on}) and high resistance state (HRS or R_{set}, or R_{off}) due to the random motions of particles and the variations in the writing and reading voltage [36]. Depending on the application, these variations might result in the malfunction of the whole system, some studies therefore have been conducted previously on this issue. In [120], the impact of geometry variations (one of the results of process variation) on the electrical characteristic of the memristor is studied. In [102], an RRAM SPICE model that can capture main device characteristics including variations is developed and this model is used to design variation-aware RRAMs. In [134], variance in the lower memristance state is compensated for by using the redundancy technique. The effect of memristor state resistance change on a memristor-based tunable amplifiers are studied in [153]. The study in [111] touches on the non-uniformity in memristor-based memory cells due to the placing and tries to minimize it by changing the data mapping. In [110], the design considerations for variation tolerant multi-level CMOS/Nano memristor memory is studied. The study states that between the different layers, there can be 10x difference in memristor resistances which is demonstrated experimentally as well in [39]. In all of the published papers so far, the focus has been on studying the impact of process variation on the memristor as an individual device or its evaluation in a different context. However, the impact of memristor process variation at the system level is very crucial when used in a RAP architecture which is a very promising emplacement field for memristors [176] [69].

Any variation on the memristor devices affects the level of the capacitor voltage after a

compare operation. As an example, if the memristance value becomes lower than the normal value due to the process variation, the voltage level after a match case could be lower than the expected and the result can be misinterpreted as mismatch. At this point, the optimization of sense amplifier voltage threshold is an important goal to obtain the least error with the highest accuracy for compare operations in associative processing utilizing memristors. For this purpose, an adaptive threshold finding technique can be used by following the machine learning practices. To achieve this; first system is trained to obtain the optimum threshold voltages and then, these values are tested.



(a) Without variability

(b) With variability

Figure 3.5: Effect of process variation on Vc

Process Variations and Effects

Figure 3.5 shows the row capacitor voltages (V_c)s of the first three cycles during the compare phase of a RAP circuit for in-place addition operation. Under ideal conditions for memristors without any variability, the response of the circuit should be as shown in 3.5a. After the compare phase, V_c voltages have only some values that depend on the number of mismatched cells and the circuit works error-free at a defined threshold voltage of 0.3V. However, as stated in the previous section, due to the process imperfections and other reasons, memristor values are prone to vary during the circuit operations. Taken from experimental measurements of

the fabricated HP memristor [88], memristance variability can be approximated as a normal distribution which is conceivable considering the random nature of a resistance where σ can be changed depending on the process. When this variability is taken into consideration with the real sigma and mu values obtained after performing distribution fitting in MATLAB on these data, the real implementation of the same circuit behaves as shown in 3.5b on the same data set. In this case, the defined threshold may sometimes cause errors since within a given time, the discharging current depends on the total resistance connected to the capacitor, and these resistances vary. At this point, it becomes crucial to select a reasonable V_{th} (i.e., sense amplifier reference voltage) for the sense amplifiers based on the variability.

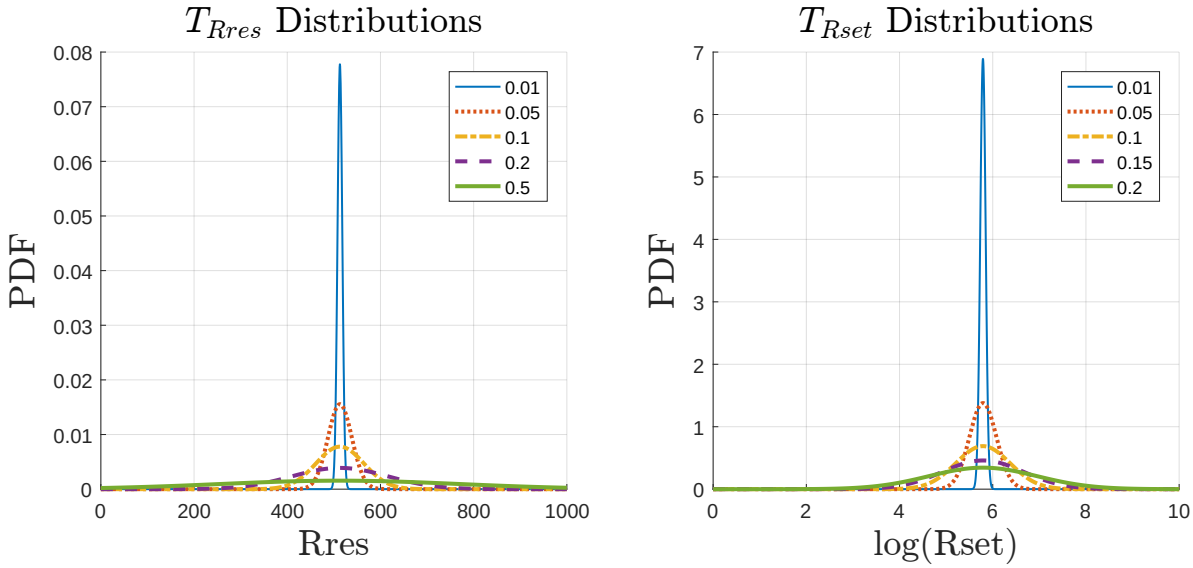


Figure 3.6: The distributions for Rres and Rset tolerances

Methodology

In order to find the best V_{th} values with respect to a process variation, the methodology involves first observing the system, using the observations to apply a learning method to classify the two regions with the highest accuracy, and finally testing it on a new set of data to prove success. The simulation results are divided by the rule of thumb in machine learning (50:25:25) [112]. The first 50% of the results are separated for the learning phase

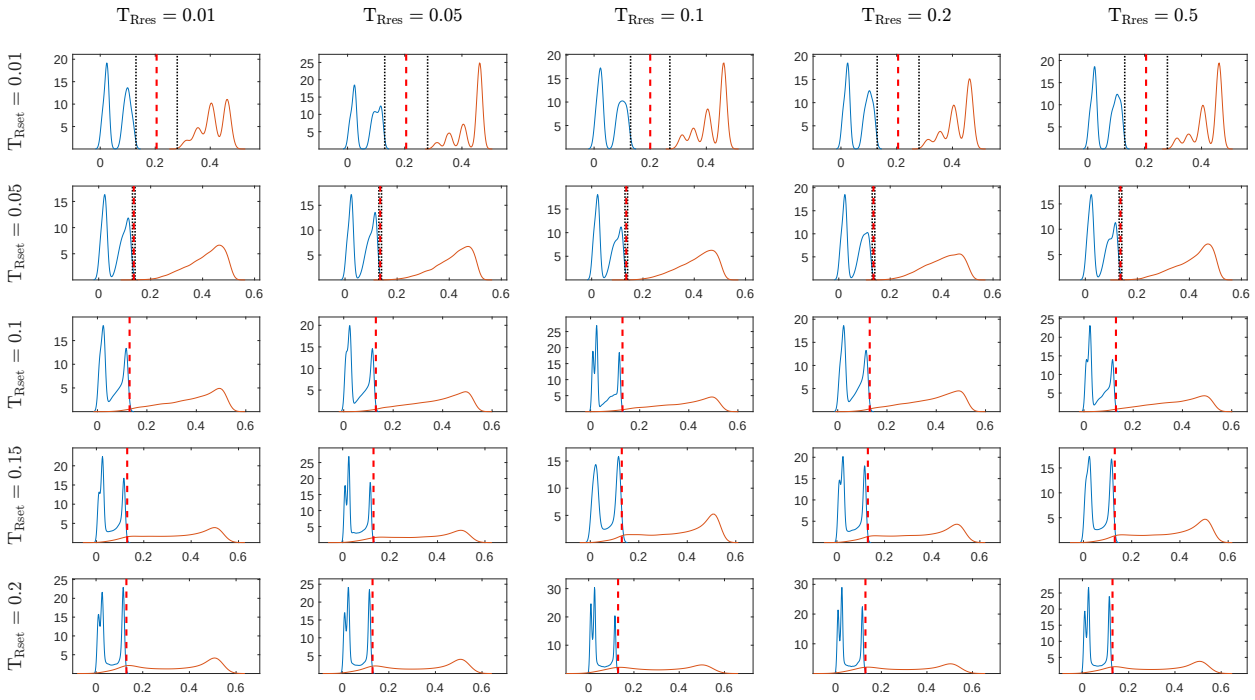


Figure 3.7: Kernel densities with respect to tolerances and obtained V_{th}s (red vertical line)

(adaptive threshold finding), the other 25% is used for cross-validation, and the remaining 25% is separated for evaluation (testing). In the learning phase, the adaptive threshold finding method based on the brute force method is employed since design space is relatively simple.

To prove the concept, the methodology is tested on an experimental platform on the simulator described in Section 2.5. For the transistors, Predictive Technology Models (PTM) is used to simulate high-density memories with sub 20nm feature sizes [159]. For the memristor element, the platform allows plugging in any model for any two terminal resistive devices and we adopt the device mode l presented in [25]. In the circuit, we use 0.9 volt voltage source (Vdd). For distributions of memristance values, the mean of the memristance is kept as constant (obtained from the HP study [88]) and the sigma is changed depending on the tolerance. Equation 3.1 shows the formula of the normal distribution.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{3.1}$$

Equation 3.2 presents the relationship between tolerance and sigma.

$$Tolerance = \frac{\sigma}{\mu} \tag{3.2}$$

For the tolerance range, the tolerance is swept between 0.01 and 0.5 for Rres and between 0.01 and 0.2 for Rset as shown in Figure 3.6.

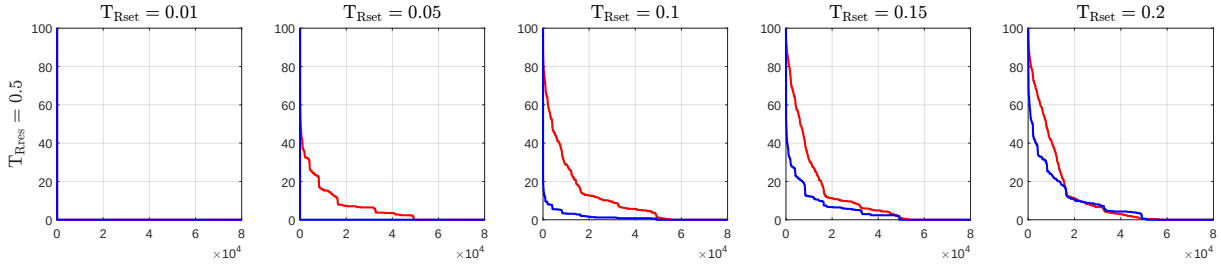
Learning (Adaptive Threshold Finding)

For the learning phase, a 512x48-bit RAP is run with different set of key and mask values for 128 cycles that generates 65,636 Vc values (sense amplifier inputs) for each distribution with its corresponding tags. The content of the CAM is generated randomly. For the mask, the maximum mask width is taken as 4 bits since the widest LUT is 4-bit length [32] (e.g., multiplication uses 4-bit masks, addition uses 3-bit masks). Figure 3.7 shows kernel density functions of V_c values of randomly generated compare operations on the training set with different tolerance values for all R_{res} (R_{on}) and R_{set} (R_{off}) combinations. For example, the resulting plot at the second row and third column corresponds to the kernel density of V_c voltage values for tolerance 0.02 and 0.1 for Rset and Rres respectively. For Rset tolerances of 0.01 and 0.02, the match and mismatch areas are totally separable with a wide margin for $T_{Rset} = 0.01$ and a very narrow margin for $T_{Rset} = 0.05$. However, as tolerance for R_{res} increases, error-free design becomes unfeasible. As seen in the figure, the kernel densities are nearly the same for different tolerance values (0.01 to 0.5) in Rres and changes by the tolerance of Rset considerably.

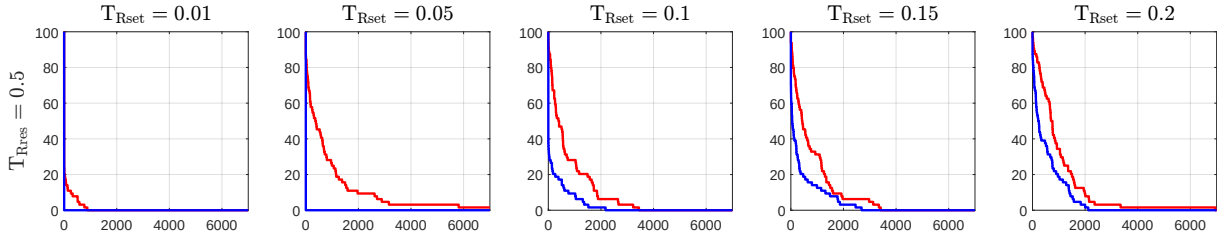
In order to find the best V_{th} value for each process variation combinations, the brute force approach is utilized. Briefly, the best accuracy point is sought by scanning the values between 0 and 0.6 volts which is the voltage range that V_c voltage can take. We assume that precision of the sense amplifier can be 10 mV and select this value as a step size, so it scans the range

Table 3.3: Optimized V_{th} values in volts

T_{Rset} / T_{Rres}	0.01	0.02	0.1	0.15	0.2
0.01	0.21	0.21	0.21	0.21	0.21
0.02	0.14	0.14	0.14	0.14	0.14
0.1	0.13	0.13	0.13	0.13	0.13
0.2	0.13	0.13	0.13	0.13	0.13
0.5	0.13	0.13	0.13	0.13	0.13



(a) In-place Addition



(b) Unsigned Multiplication

Figure 3.8: The cumulative distribution functions (CDFs) of deviation in the results. (The red line corresponds to untrained results and the blue one is the trained results)

by increasing by 0.01 V. The algorithm can scan all possible combinations in a matter of milliseconds for 65,536 data points, so it is preferred since it gives the optimum results and does not constitute an overload for such design space. The resultant points are close to the intersection points on kernel density plots as shown in Figure 3.7. If there is no intersection between two regions (e.g., the kernel density function of the first row and the first column), in other words, an error-free selection is possible, then the middle point is selected to provide the highest margin to the sense amplifier. Table 3.3 shows the extracted V_{th} values in volts for each tolerance combination. Additionally, percentage of getting logic-1 instead of logic-0 in the tag field because of the variation in R_{res} is zero. For this reason, we continue to test

the system by regarding highest R_{res} tolerance ($T_{Rres} = 0.5$) and changing R_{set} tolerances between the provided range.

Validating

In the validation phase, obtained V_{th}' values are inserted in the circuit, and the accuracy of the design is measured on a different set of data whose size is half of the training set data (25% of the total data). Broadly speaking, in associative processing, the number of mismatched rows is almost more than the number of matched rows; in other words, the compare operation is logic-0 biased. If the sense amplifier is set to always predict logic-0 after the compare phase (setting V_{th} to a large value out of the possible V_c range e.g., 1 V), the accuracy of the system becomes 87%. For this reason, in reporting results, the performance (accuracy) of the system is only measured for match cases, and mismatch cases are ignored.

Table 3.4 presents results of the validation phase. Table second column shows the results for the case in which the threshold voltage is set to a single value (0.3 V) without considering the variability. The third column shows the results for the case in which the threshold voltages are set according to the process variations by taking the variability into the account (i.e., as presented in table 3.3). According to the results, adaptive V_{th} selecting by the process variations provides significant improvement in RAP operations.

Table 3.4: Tag Prediction Accuracies and Improvement Percentage for $T_{Rres} = 0.5$

T_{Rset}	$V_{th} = 0.3$ V	Adaptive V_{th}s	Improvement %
0.01	0.99	1	0.72
0.05	0.92	1	8.40
0.1	0.78	0.98	25.42
0.15	0.67	0.92	37.84
0.20	0.60	0.88	45.77

Evaluation

For the evaluation, the new set of simulations are run with both configurations and the results are compared. Instead of running random compare operations as done in the previous section, the real applications are used to evaluate the proposed method. For the simple arithmetic operations, the in-place addition and unsigned multiplication are tested.

Figure 3.8a shows the comparison of the cumulative distribution functions (CDFs) of traditional (untrained) and trained systems for in-place addition on 512x12-bit number pairs. These figures simply represent the probability that the difference between the actual and computed results of the addition operation takes a value less than or equal to the value on the x axis. The adaptive selecting of Vths provides an error-free system implementation, especially for $T_{Rset} = 0.05$. Figure 3.8b shows the same results for unsigned multiplication of 64x8-bit number pairs.

In order to witness the effect of adaptive threshold selecting,2D DCT block of the Joint Photographic Experts Group (JPEG) algorithm [125] is replaced with its RAP counterpart. JPEG is a commonly used method for lossly compression of digital images. Figure 3.9 demonstrates the block diagram of the algorithm. To describe basically, the algorithm partitions the input image into 8x8 pixel blocks and centers these pixel values around 0 by subtracting 127 from each pixel. After that, 2D DCT transform is performed on each block to calculate the frequency components, both horizontally and vertically. If the image is concentrated in fewer coefficients of the DCT, the compression becomes more efficient. In the third step, quantization is performed in which a range of values is compressed into a single

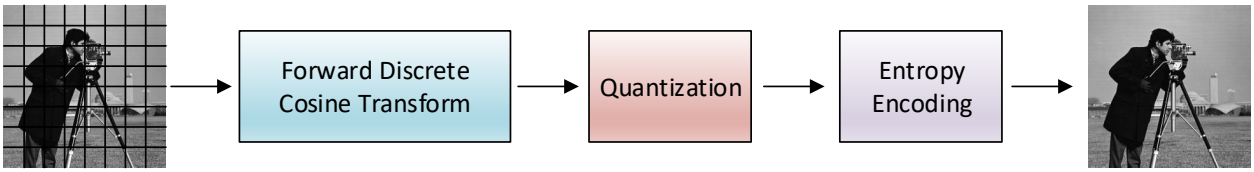
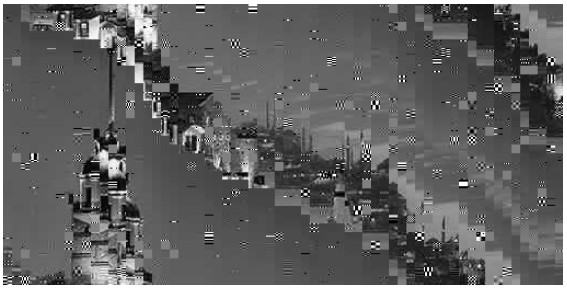


Figure 3.9: JPEG Block Diagram

(a) Adaptive Vth (PSNR = 43.09)



(b) Single Vth (PSNR = 13.73)

Figure 3.10: Comparison of JPEG results for $T_{Rset} = 0.01$ and $T_{Rres} = 0.5$

value. Lastly, the output of the quantization step is encoded by Huffman coding algorithm [50].

Discrete Cosine Transform (DCT) [2] is a technique for converting a signal or image into the summation of a series of cosine waves oscillating at different frequencies (Equation 3.3).

$$C(k) = w(k) \sum_{n=0}^{N-1} x(n) \cos\left(\frac{\pi k}{2N}(2n+1)\right) \tag{3.3}$$

DCT is correlated to Fast Fourier Transform (FFT); they are both used to convert date from spatial-domain into frequency domain, but DCT uses only cosine functions and real coefficients. This method is called Fast Cosine Transform [109]. Equation 3.4 shows the implementation of 1D DCT by using FFT operations.

$$\begin{aligned} y(n) &= \begin{cases} x(2n), & n = 0, 1, ..., N/2 - 1 \\ x(2N - 1 - n), & n = N/2, ..., N - 1 \end{cases} \\ C(k) &= Re\left\{e^{-j\pi\frac{k}{2N}} FFT\left\{y\left(n\right)\right\}\right\} \end{aligned} \tag{3.4}$$

To implement the complete system, we firstly implemented Fast Fourier Transform (FFT)

on RAP and then we implemented 2D DCT by exploiting the existing FFT. In the implementation, we store the values in two different ways. For pixel values (generally grater than 1), we used 13 bits for integer part and 2 bits for fractional part. For complex coefficients (eg. twiddle factors in FFT and scaling factors in DCT), we used 2 bit for integer part just keep the sign and 8 bits for fractional part. Therefore, the designed RAP is capable of processing 19-bit precision for pixel values and 10-bit precision for complex coefficients. During the compression, addition, subtraction, absolute value, 2's complement, and multiplication operations required to perform 2D DCT are run on the RAP.

Figure 3.10 shows the output images of the compression. As seen in figure 3.10b, the variability in memristor values along with unconstrained V_{th} selection lead to unacceptable results in image quality. On the same image, selecting a V_{th} adaptively depending on the variations provides enough result is in the acceptable range of taste of human perception. For the PSNR results, the output images of the RAP-based JPEG compression are compared against the output of the unmodified JPEG i.e. Matlab *dct*2 function that runs the 2D DCT with full computer precision (64-bit floating point). While the adaptive V_{th} provides a PSNR rate of 43.09, the single V_{th} (0.3 Volt) gives 13.73 which is another indicator of high noise level.

3.5 Conclusions

In this chapter, APs are inspected by presenting its trade-offs in terms of performance, energy reduction, and reliability. For the performance, AP is compared shallowly with the traditional processors and a significant advantage on vectorial operations (SIMD processors) are proven. Then, the energy consumption of RAPs and SAPs are compared. Finally, the reliability concerns related with the memristor-based APs (RAPs) are inspected together, and the possible solutions are proposed for the vulnerable points such as memristor variability, reliability, and retention issues .

Chapter 4

Approximate In-Memory Computing

Associative processors are a promising candidate for in-memory computation, however the existing implementations have been deemed power hungry as touched in Chapters 2 and 3 . Approximate computing is another promising approach for energy-efficient digital system designs where it sacrifices the accuracy for the sake of energy reduction and speedup in error-resilient applications. In this chapter, approximate in-memory computing is introduced on both RAP and SAP architectures. Two approximate computing methodologies are proposed; bit trimming and CAM-cell scaling. In addition, a hybrid approximation methodology is proposed for both architectures together with a design flow to optimize the degree of approximation.

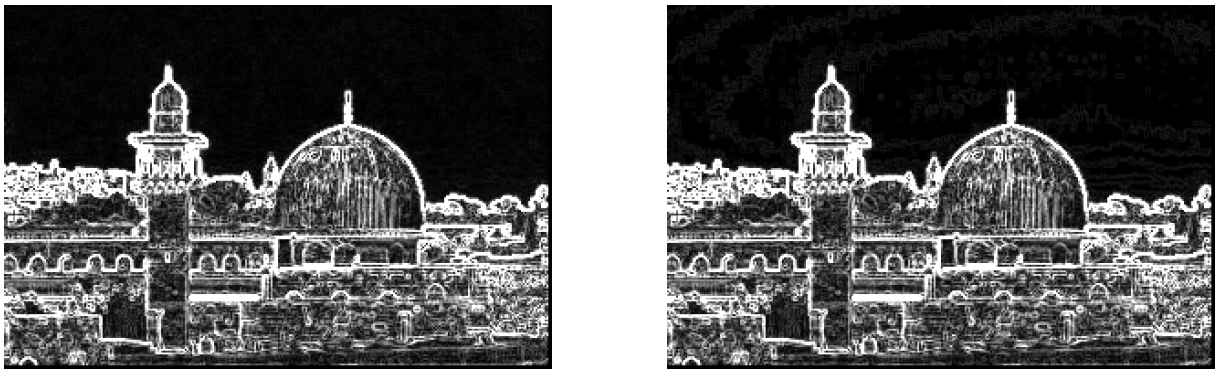
4.1 Approximate Computing

Approximate computing is a computing paradigm that aims to increase the energy efficiency of the computing systems by relaxing the correctness constraints [116] and best suited for intrinsically error resilient applications such as image vision, multimedia, signal processing,

etc. As an example, the approximate computing can be employed in image processing and the results may not harm the taste of vision since human perception system is limited. To illustrate, Figure 4.1 shows two outputs from the JPEG algorithm where the cameraman image is compressed. Figure 4.1a shows the result of the compression when all pixels are represented in their full range (i.e., 8 bits integer). On the other hand, Figure 4.1b shows the result of the JPEG algorithm that applies on the same input image approximately where the pixels are approximated by trimming their last three bits so that each computation is performed on 5-bit. The peak signal-to-noise ratios (PSNR) of two images are 31.74 dB and 31.52 dB respectively, where around 0.7% difference exists between them. As another example for the error-resilient Sobel Filter application which is an edge detecting algorithm. Figure 4.2 shows two output images. Figure 4.2a shows the result of the filtered image when all pixels are represented in their full range (i.e., 8 bits integer). On the other hand, Figure 4.2b shows the result of the same algorithm where the pixels of the same input image is approximated by trimming their last two least significant bits (i.e., each computation is performed on 6-bit). The figure shows that the approximately processed image exhibits a PSNR of 33.17 dB with an average error of 6.20%. It can be visually inferred that those two figures are almost indistinguishable. However, the approximate system allows the utilization

(a) (b)

Figure 4.1: Full (a) and approximate (b) compressed images where PSNRs are 31.74 and 31.52 respectively



(a) Full (b) Approximate

Figure 4.2: Full (a) and approximate (b) outputs of the sobel filter algorithm on a image where PSNR is 33.17 for the approximate image

of approximate adders and multipliers which performs the operations on fewer bits, therefore provides energy reduction and performance improvement.

Inside a system, approximate computing can be introduced at two different levels; logic-circuit and algorithm-architecture levels[137]. At the circuit level, the most common method is designing functionally approximate circuits that utilize less components than the fully accurate counterparts [180]. Other ways of hardware approximations are over-scaling the circuit timing and/or voltage [117] and approximation in memory [78]. In this level, approximate computing can be realized by the design tools that support synthesizing the approximate correspondence of the implemented circuits [161]. For example, a VLSI design software can include the approximate versions of the some arithmetic circuits addition to accurate ones and these approximate circuits can be used in the error resilient parts of the chip as determined by the system level simulations. At the algorithm-architecture level, significant components (significant, implies impacting a metric of interest) in the overall system are favored over less significant components while satisfying the target metric (e.g., performance, energy budget, area) [115]. For example, while compiling a source code for an algorithm, the corresponding binary code selectively ignores some insignificant computations and/or skip some less important memory accesses.

Approximate computing can also be applied to memristor-based digital systems. In [133], an architectural extension to GPUs are proposed to avoid the frequent re-executions by recalling their results directly from a cache-like resistive CAM and memory structure tied to the every FPUs in the GPU. The RCAM structure facilitates approximate matching, therefore provides energy consumption with the expense of acceptable decrease in quality. The same idea is improved in [59] and [56] by enabling different degree of approximation for the rows and bit columns and adding single-charge multiple-access RCAMs respectively. In [56], the resistive CAM accelerator approach is proposed for tunable CPU approximation. The paper proposes an architectural extension to GPUs to avoid the frequent re-executions by recalling their results directly from a cache-like RCAM-based structure tied to every FPUs. The RCAM structure provides an approximate matching up to 1-HD or 2-HD, therefore provides energy consumption with the expense of acceptable decrease in PSNR. However, this method does not pay attention to the bit locations (whether it is MSB or LSB) and inserts arbitrary errors to the system and requires a training phase. In [155], an approximate in-memory hamming distance calculation is proposed based on a memristive associative memory. The study in [90] performs the two important subtasks of a visual recognition system as approximate inside the memory to get an advantage in energy and performance. In [183], the approximate storage of images in STT-RAM by writing the data approximately and then managing the memory access adaptively with respect to write mode (approximate vs precise). In all these studies, the computation is done inside the memory partially and the remaining parts are done outside. On the their hand, AP performs the every operation inside the memory without any interference with the other resources, so a new approximation methodology implemented inside the AP is required for approximate in-memory computing .

4.2 Approximate Memristive In-memory Computing

In this section, for the performance/energy management strategy through approximate computing in RAPs, we introduce the two methodologies. One can be applied in system level (bit trimming), and other can be applied in circuit level (memristance scaling). In Section 4.2.3, the effect of this approximation in the state of the art applications are inspected.

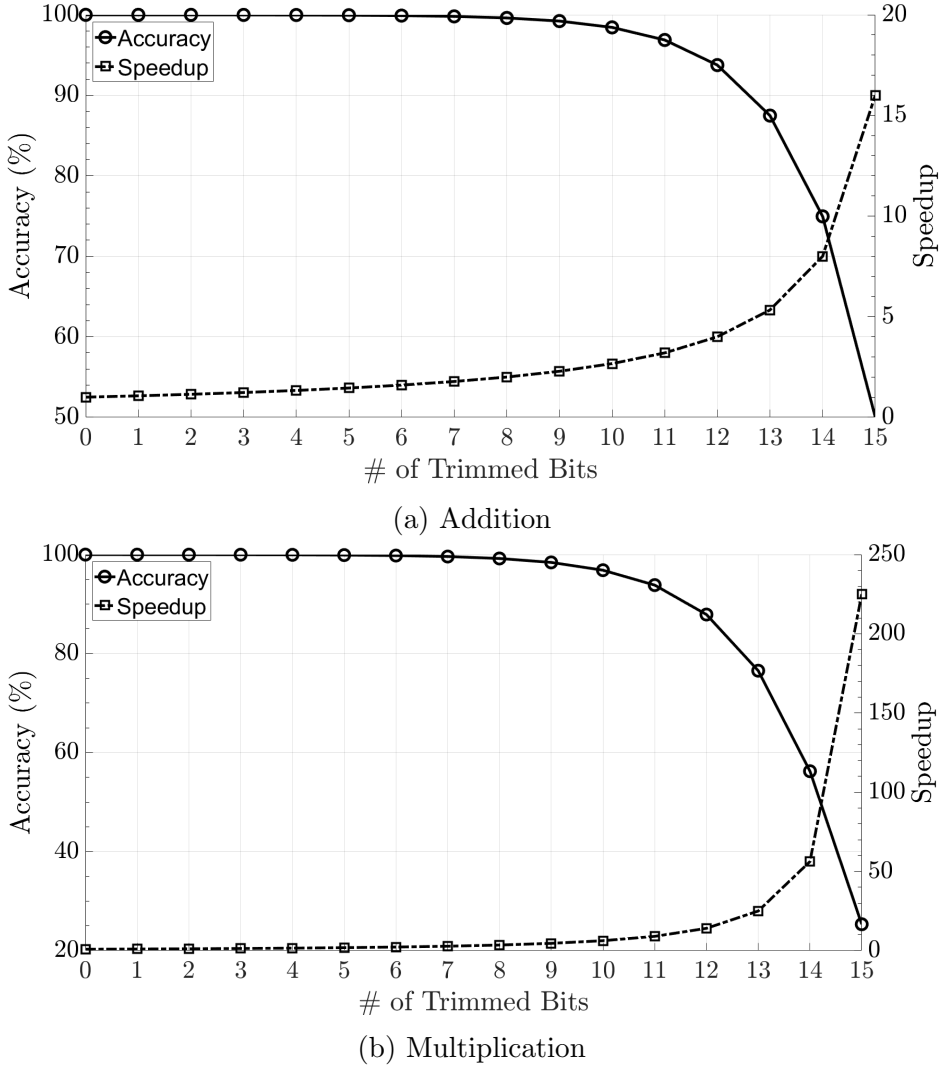
(a) Full value (b) Approximate value

Figure 4.3: Bit trimming in the MAP

4.2.1 Bit Trimming

In associative computing, an arithmetic operation are normally performed by starting with Least Significant Bit (LSB) and progressing bit by bit to the Most Significant Bit (MSB). If one wishes to involve fewer bits in the operation, say to discard the k least significant bits, for example, then this can be accomplished by simply skipping those k LSBs and starting the operation at bit $k + 1$. For this reason, associative computing provides an inherent support for bit-scale approximate computing. In this way, the stored value in the RAP is approximated by throwing its some less significant bits. As an example, Figure 4.3a shows a fixed point representation of a real number (6.6521) in the RAP in big-endian notation. During an operation, if the last 4-bits of this number is trimmed, 0.06% percent accuracy of the number is lost. For such a number representation (fixed-point with 4-bit integer, 12-bit fractional parts), the numerical accuracy loss is limited by at most ± 0.0037 when all trimmed bits (i.e., the last four bits) correspond to a value (logic 1). As seen in Table 3.1, the cost of an operation in MAPs depends on the number of bits (m) rather than the vector sizes (n). For this reason, this loss in precision as a result of smaller m provides savings in run time

(i.e., number of cycles). Since a cycle in the RAP corresponds to either compare or write operations, the total number of compare and write operations decreases as well and these results in energy savings at the same time. For example, if the instruction that performs the vector add operation in Figure 2.9 was called as *AdditionIP(8, 7, 5, 3, 1)*, the controller would skip the LSB of A and B and perform the same operation as approximate where the operation works 25% faster and consumes less energy.



(a) Addition

(b) Multiplication

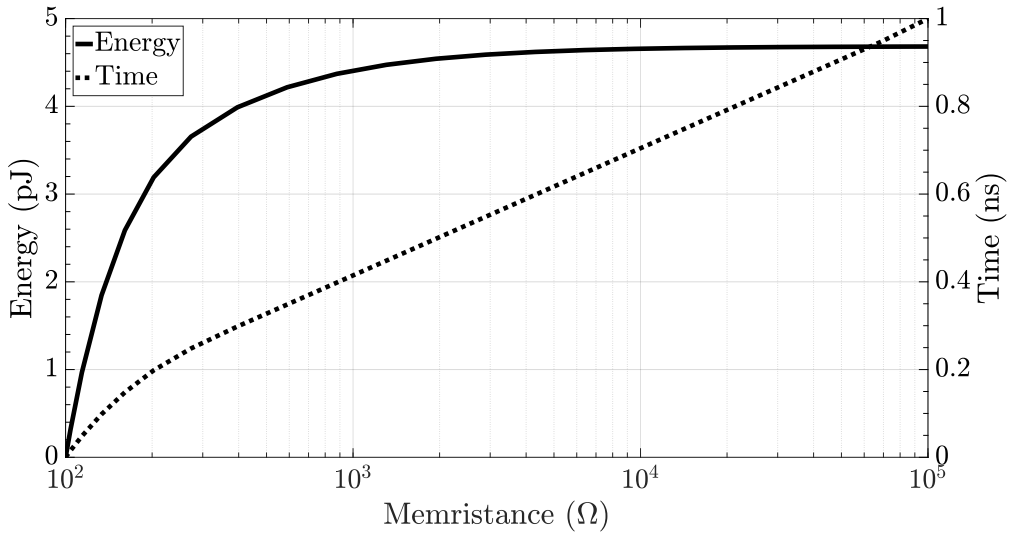
Figure 4.4: Number of trimmed bits vs. accuracy & speedup for the addition and multiplication operations in the RAP

Figure 4.4 shows the effects of bit trimming (value approximation) on speedup and accuracy for addition and multiplication operations on 1000, 16-bit random numbers. The Matlab-

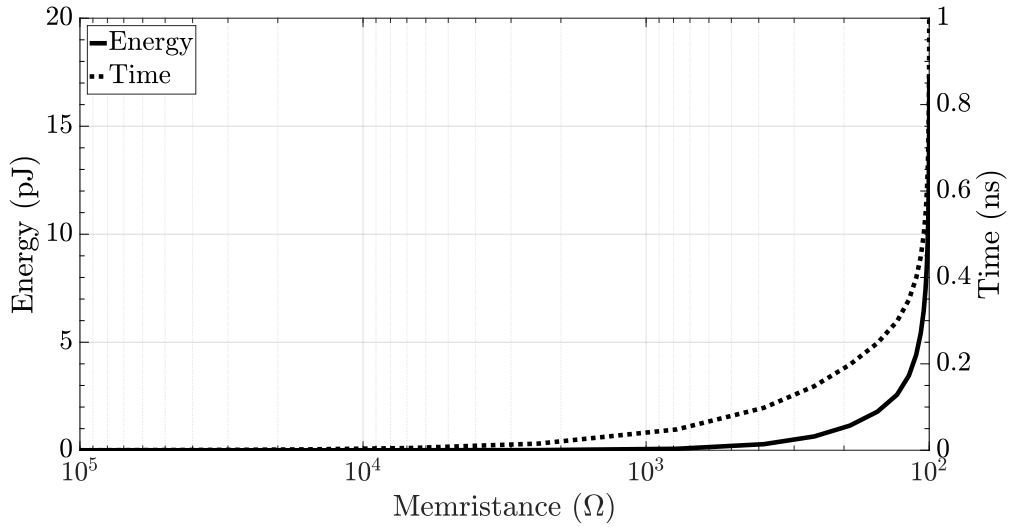
based RAP Simulator detailed in Section 2.5 is used to obtain these results. The figure shows the number of trimmed bits from these 16-bit numbers versus accuracy and speedup. As presented in the figure, until about 9-10 bits of precision, the approximation still provides relatively enough accuracy (i.e., the accuracy within some error resilient system requirements) together with considerable speedup advantages. To illustrate, even though half of the numbers are trimmed, the accuracies are still more than 97% and the system provides more than 2x speedup for addition, and thus positively influences the energy consumption as well. The speedup and reduction in energy consumption in multiplication is expected to be much more since its runtime (and also energy consumption) is quadratic (see Table 3.1). Moreover, since the starting point of the operations can be modified in software by a controller, this approximation method in the RAPs is also tunable, so that anytime the system can increase the accuracy by shifting the starting bit of the operations to the right bits (LSBs) at the expense of performance and energy or vice versa. Depending on the characteristics, resiliency, tolerance, and environmental conditions of the application, the system accuracy can be set by tuning the number of bits included into the computation.

4.2.2 Memristance Scaling

As stated in Chapters 1 and 2, memristors are switched between the full range in the logical usage, so that R_{on} takes the minimum and R_{off} the maximum memristance values corresponding to logic-1 and logic-0 respectively. However, setting the memristor exactly to these values could be costly. For instance, Figure 4.5 shows the writing energy consumption and time results of a memristor in [114] with respect to memristance while it is switching from R_{on} to R_{off} and vice versa. As seen in the figure, the memristor consumes more energy when it becomes close to the R_{on} value (100 ohm). This is intuitively meaningful since the current increases more if memristance value gets smaller for a fixed voltage. On the other hand, the memristor switching rate decreases when it becomes closer to R_{on} as well. Overall, this



(a) $\mathbf{R_{on}}$ to $\mathbf{R_{off}}$ switching



(b) $\mathbf{R_{off}}$ to $\mathbf{R_{on}}$ switching

Figure 4.5: An example case showing memristance-energy and memristance-time relations for switching the memristor in [114] between Roff (100 k$\mathbf{\Omega}$) and Ron (100 $\mathbf{\Omega}$)

region causes most of the energy consumption. In addition to energy consumption, switching the memristor between the full range of R_{on} and R_{off} increases the timing and affects the performance negatively. For this reason, avoiding costly writing operations could provide considerable amount of energy saving and performance benefits in the MAP.

Memristance scaling is a method for switching the memristors state within a subrange of the attainable minimum and maximum memristance values [45, 28]. This can be performed by

the scaling both the memristance write voltage and write time. Even though the RAP stores, represents and processes the data as digital, the compare operation in the RAP is based on an analog matching circuit (see Figure 2.5), with the sense amplifiers converting the output of this analog circuit back to the digital domain. In the proposed approximate computing method based on memristance scaling, the range of memristor values can be shrunk to allow approximation as a result of compare operations. This shrinking in the range provides less write energy and shorter write time in the writing operations. However, this idea comes with some problems; first of all, like any other nanodevices, memristors has also variations due to the process imperfections [27]. When all bits in the RAP are scaled aggressively, any variability can cause the wrong computation and the situation becomes even more serious if it occurs in the MSBs. For this reason, to ensure the accuracy vs approximation trade offs, the memristors that correspond to the LSBs can be scaled more aggressively than the memristors in the MSBs, so that the MSBs become more error resilient and even error free. In this way, the accuracy can be guaranteed at the loss of some degree of quality. The bit trimming can be considered as a kind of memristance scaling where memristors are scaled so that every compare operations results a mismatch, so there is no need to perform the compare operations.

4.2.3 Experimentation

Simulation Framework

To evaluate the proposed approaches, the SPICE-based in-house simulator is utilized that can efficiently simulate realistic RAP architectures with different functions as illustrated in Figure 2.15 in Section 2.5. During the experimentation, the Matlab simulator does the same operation as logical with perfect conditions as error-free. By comparing the results of both Matlab and HSPICE, the statistics about the processor are obtained and some metrics

are assessed. For the transistors, Predictive Technology Models (PTM) from [159] are used to simulate high-density memories with sub 20nm feature sizes [151]. For the memristor element, even though there are some memristor models switching within huge memristance values such as 125k - 125G and provides better scaling ratios, they are not practical for RAPs since their switching speed is relatively slow (e.g., 10 ns) [107, 121]. For this reason, a fabricated nano-second switching time memristor is adopted which has a size of 50 nm [114] since it is best suited for such a processor targeting high-performance error-resilient applications. In the simulations, its corresponding SPICE model in [168] is used. For the sense amplifier, a low-power, sub-ns amplifier design in [145] is employed in the circuit.

Reliability

As stated in Section 3.4, memristors suffer from process and operational imperfections which lead to a variance in their R_{on} and R_{off} values, Additionally, process imperfections in other semiconductor materials (e.g., transistor, capacitor, etc.), fluctuation in voltage rails, etc. can have effects not only on the memristor but also on the overall mechanism of the system. In addition to all these impairments, approximate computing can be regarded as another source of operational imperfections. In all approximate computing studies, the system is forced to run under the abnormal operational conditions where the factors deemed safe under normal conditions start to be effective against the correctness of the system. In this case, taking the other reliability issues into account together with approximate computing becomes a must.

To perform the evaluation of the overall approximate system accurately, the system is inspected under the presence of different sources of variations. Even though there are also other reliability problems of memristors aside from variability such as retention time and endurance, they are addressed in some studies [17, 28, 176, 88]. Therefore, the reliability problems that affects the computational accuracy of approximate computing under mem-

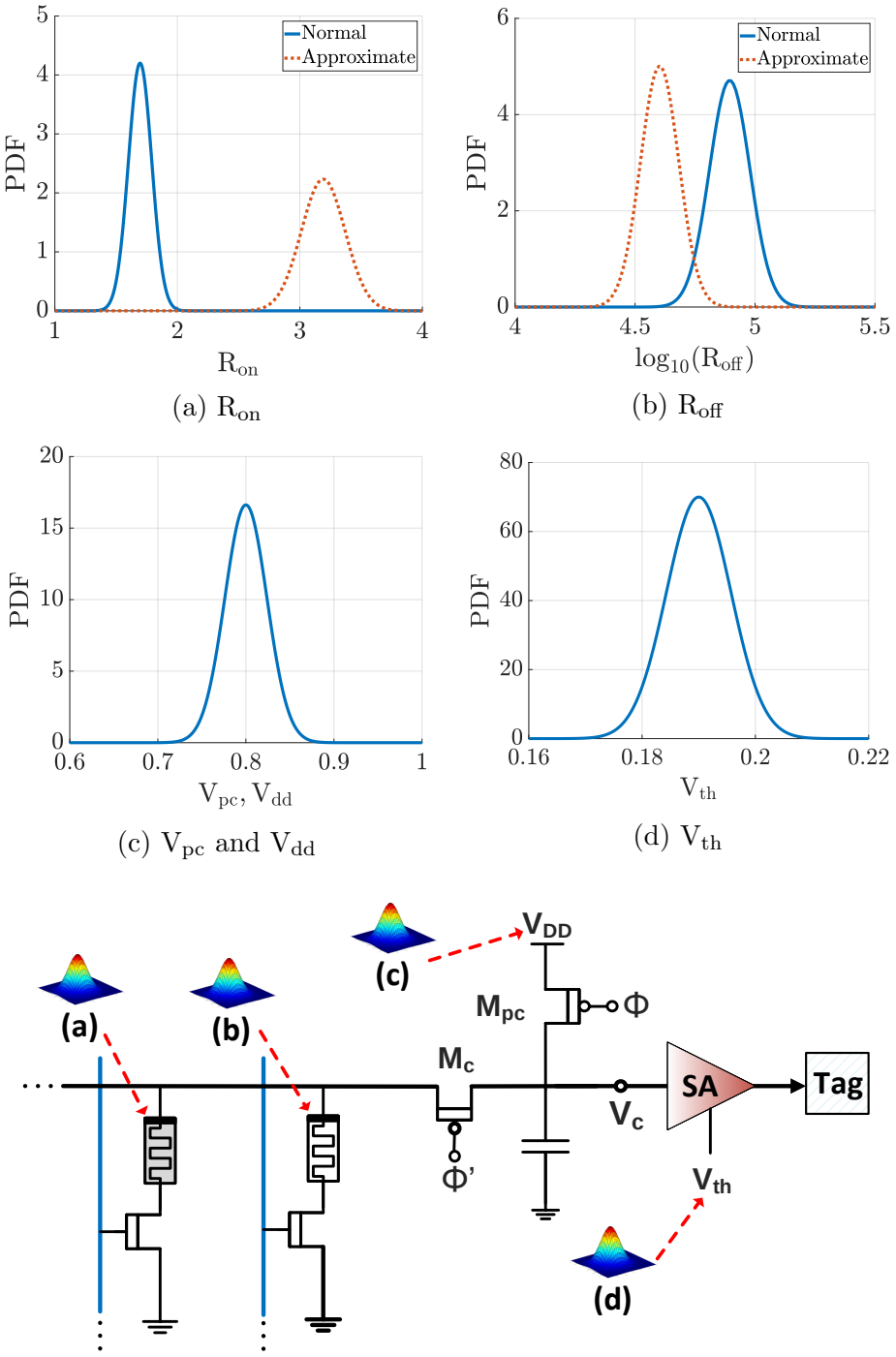


Figure 4.6: Variability sources and variations in RAP

ristance scaling is evaluated. The method presented in [175] is used to make the processor resilient to these overwhelming variations as much as possible. As a result of these conditions, in the RAP, the functional errors occur in scenarios where, as a result of compare

cycle, a logic-1 is misinterpreted as logic-0 or logic-0 is misinterpreted as logic-1 in the sense amplifier. These functional errors can be contributed by memristor and voltage variations as well. For example, exact write to a memristor is not possible and generally resulted memristance follows a normal distribution [88]. For this reason, while evaluating the circuit, some variations are inserted into the circuit. Figure 4.6 shows the variability distributions for memristances (R_{on}, R_{off} in Figure 4.6a and 4.6b), pre-charge voltage, supply voltage (V_{pc} and V_{dd}), and threshold voltage (V_{th}). Since the variations in the memristance also reflects the write voltage variations as well, an additional variation for the V_{wr} is not inserted into the system. For memristance distributions, the distribution is taken from an experimental data on the fabricated memristors by fitting it into normal distribution where R_{off} distribution is in the log scale [88]. For voltage sources, the sigma of variations is taken 3% [79] [79] [80]. Figure 4.6e shows the insertion points of these variations to the RAP circuit for better understanding.

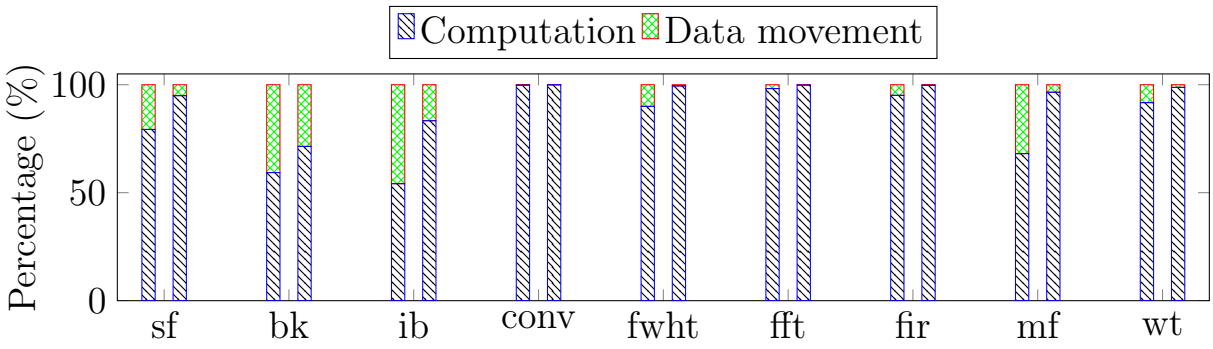


Figure 4.7: Data movement vs. computation percentage for the benchmarks where left and right bars correspond to energy and time respectively for each benchmark

Evaluation

For the evaluation of approximate computing in RAPs, energy reduction, speedup, and energy × speedup results for 9 benchmarks are presented in Figure 4.8. For the benchmarks, the AX-BENCH tool [179] that includes benchmarks for approximate computing is used together with specified input sizes to evaluate the approximation methods together with

Table 4.1: The evaluated benchmarks, their platforms, and quality metrics from [179]

Benchmark	**Domain**	**Platform**	**Quality Metric**
Sobel Filter (sf)	Image Processing	CPU, GPU, ASIC	Image Diff.
Brent-Kung (bk)	Arithmetic Computation	ASIC	Avg. Relative Error
FIR	Signal Processing	ASIC	Avg. Relative Error
Wallece-tree (wt)	Arithmetic Computation	ASIC	Avg. Relative Error
FFT	Signal Processing	CPU	Avg. Relative Error
Image Binarization (ib)	Image Processing	GPU	Image Diff.
Convolution (conv)	Machine Learning	GPU	Avg. Relative Error
FastWalsh (fwht)	Signal Processing	GPU	Image Diff.
Mean Filter (mf)	Machine Vision	GPU	Image Diff.

the corresponding architectures. Since the RAP architecture mainly provides advantage on vectorial benchmarks, a subset of the applications which are suitable for the RAP are evaluated from all benchmarks as presented in Table 4.1. In the results, two approximate methods on the RAP, namely bit trimming (RAP+BT) and memristance scaling (RAP+MS) are compared with loop perforation (LP) [149], neural processing unit (NPU) [29, 181] on both GPU and CPU platforms, resistive CAM accelerator [56] on CPU, and approximate ASIC designs synthesized with Axilog HDL [180]. In the comparison, the evaluated results from [179, 56] are compared against RAP+BT and RAP+MS results. The specifications of GPU and CPU platforms are also described in [179, 56]. Even though all these architectures seem non-comparable on the same basis, in reporting the results, the relative speedup and energy reduction are presented which shows how well approximate computing achieves on these architectures with different methodologies. In other words, the aim in here is to prove that memristive in-memory computing has inherent advantages that make it better suited for approximate computing, achieving higher relative improvement in all three figures of merit (performance, energy and energy performance product).

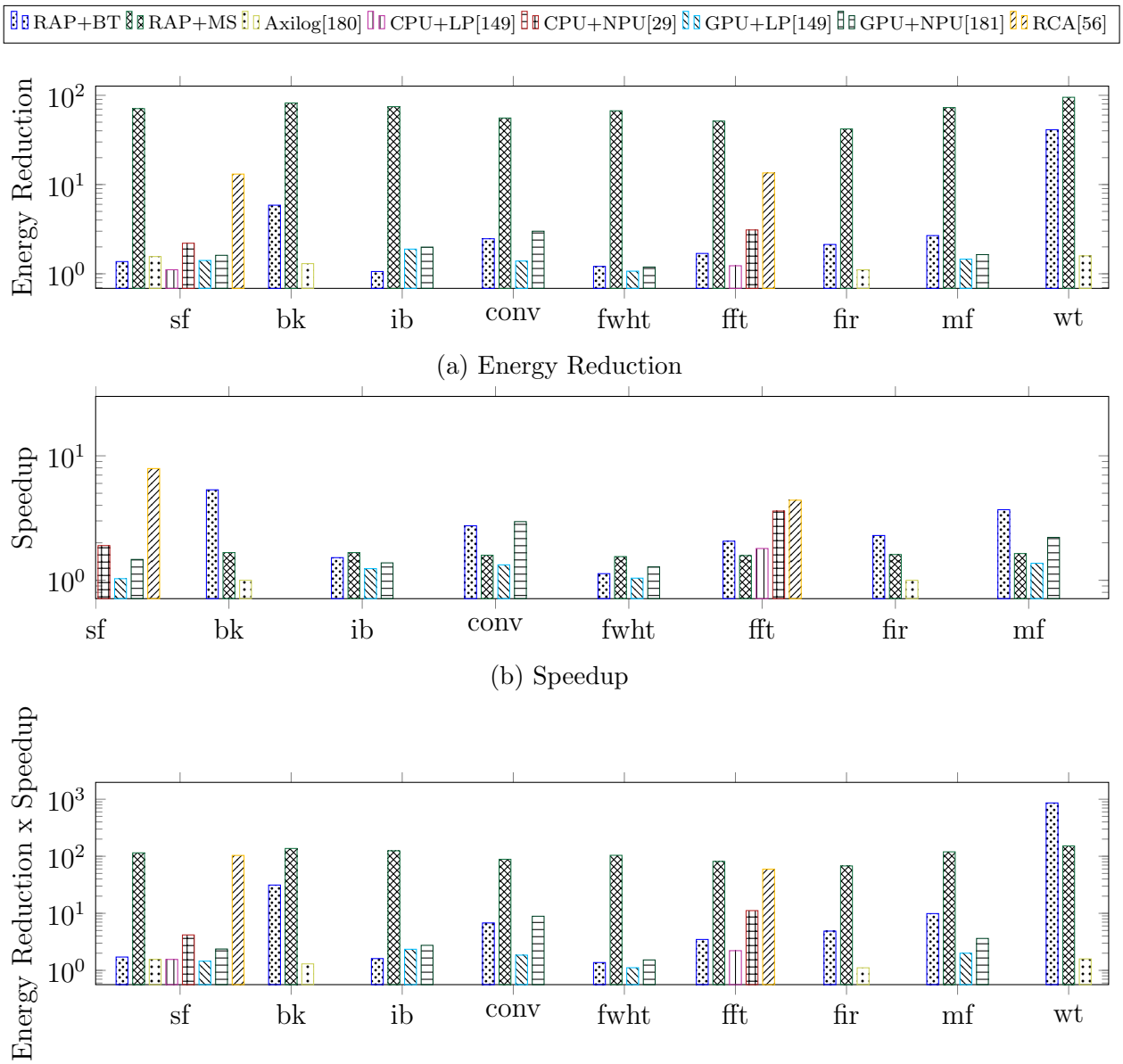
In the simulations, a 121x512 RAP is taken which is enough to perform the most costly (in terms of area) benchmark (i.e., FFT). The execution flow of a application in the RAP consists of two phases; data initialization and computation. Figure 4.7 shows the computation/data movement percentages of each benchmark in terms of energy and time respectively. In reporting results, both costs are taken into account. For computationally intensive bench-

marks (e.g., conv, fft, fir), the percentage of the computation is much higher than the data movement. For lighter benchmarks (e.g., Sobel Filter, Brent-kung, image binarization), this paradigm shifts a little towards to the data movement. Since simulation time takes days for the successive iterations on such computationally intensive benchmarks (e.g., 10s thousand cycles for FFT), first, the accurate execution profile (energy, time, variation, error probability, etc.) depending on the different conditions are obtained from the simulator (Figure 2.15), and later these numbers are used in the Matlab simulator to get the precise results.

Table 4.2: Cases for memristance scaling and their corresponding energy and timing results

Case	$\mathbf{R_{on}}$ (Ω)	$\mathbf{R_{off}}$ (Ω)	$\mathbf{T_{write}}$	$\mathbf{E_{write}}$	**Error**
Full	100	100 k	2 ns	21.7 pJ	No
Normal	1.7 k	79 k	1 ns	349.6 fJ	No
Approximate	3.2 k	40 k	0.5 ns	121.8 fJ	Yes

For the memristance scaling based approximation (RAP+MS), memristor switching range is scaled from both directions by shrinking the write time and voltage. Table 4.2 shows the three different operational cases (*full*, *normal*, *approximate*) of a memristor used in the experimentation. The first row gives the result when the memristor is switched within the full range (i.e., binary range of 100 kΩ - 100 Ω). The normal case shows a scaled memristor; however, this range still does not pose a threat to the normal operation of the processor on its own when all presented variations are taken into the consideration. The last case presents another scaled memristor where its switching range is the most aggressive, but consume the least energy. This last scaled memristor write results in errors that propagate to the digital results (i.e., flipping a bit). In the experimentations, The "Full" write is used for the accurate case where there is no approximation. For the approximate computing case, the "Normal" write is used for the most significant digits and the "Approximate" write is used for the least significant digits which are approximated. In both the bit trimming and the memristance scaling, the parameters (i.e. number of bits trimmed in the bit trimming case, and bit splitting between Normal and Approximate write in the memristance scaling case) were set so as to keep the maximum quality degradation (based on the the metric in [179])



(a) Energy Reduction

(b) Speedup

(c) Energy reduction × Speedup

Figure 4.8: Comparison of approximation methods on RAP, ASIC, CPU, and GPU platforms with different benchmarks with 10% maximum quality degradation

for each benchmark less than 10%.

The relative energy reduction ratios are presented in Figure 4.8a. Since memristance scaling allows good energy reduction in writing even for the non faulty cases (i.e., the normal-case in Table 4.2), on the overall, memristance scaling provides a significant reduction on the energy. The speedup of RAP+MS ranges from 42x in fir to 95x in multiplication (wallace tree) with a geometric mean of 66x. On the other hand, the RAP+BT in some benchmarks becomes less error resilient so that it can not allow more approximation. For example, some image processing applications (e.g. Sobel Filter, Image Binarization) process the 8-bit inputs and the degree of approximation becomes limited up to 1.4x.

Figure 4.8b shows the relative speedup of each approximation technique. In the figure, Axilog's results for the applicable benchmarks are 1 since it does not provide a speedup. As presented in the figure, RAP+BT method outperforms the other methods in the most of the benchmarks where speedup ranges from to 1.1x to 20.9x with a geometric mean of 2.8x. Even though RCA also performs well for Sobel Filter and FFT, these energy reduction and speedup include the benefit coming from the acceleration as well as approximation. Moreover, RCA, CPU+NPU, and GPU+NP require a learning phase to provide acceleration. In here, it is worth to note that, when approximation is applied to other architectures, the relative speedup becomes proportional to number of vectors (n). However, in the RAP, the relative speedup becomes limited since speedup is based on bit-length of the vectors (m). Additionally, if the amount of required computations is relatively less in some applications, the RAP processor (aims to vectorial operations) may not provide a significant speedup benefit (e.g., Sobel Filter, FWHT, Image Binarization require only addition and subtraction as seen in Figure 4.7). In here, it is worth to note that while comparing with the base case (error free), the RAP uses the as least digits as it can. To illustrate, the Sobel Filter runs on 8-bit grayscale images.

Approximate computing provides two advantages; energy reduction and speedup. For this

reason, another figure of merit can be defined as the multiplication of these two ratios, namely, energy reduction $\times$ speedup to evaluate the two metrics together. Figure 4.8c presents these results where RAP+MS outperforms all others for all benchmarks except wallace-tree where RAP+BT provides the best result.

In ASIC design, approximate computing can also be used for area reduction. In the same manner, the bit trimming in the RAPs provides area reduction as well when it is used at the design time as a static method. Figure 4.9 shows the area reduction of the comparison between RAP+BT and Axilog. In all benchmarks except Sobel Filter, RAP+BT provides better area scaling when the error rate is set at 10% maximum. However, if dynamic (tunable) approximate computing is preferred, this benefit must be sacrificed.

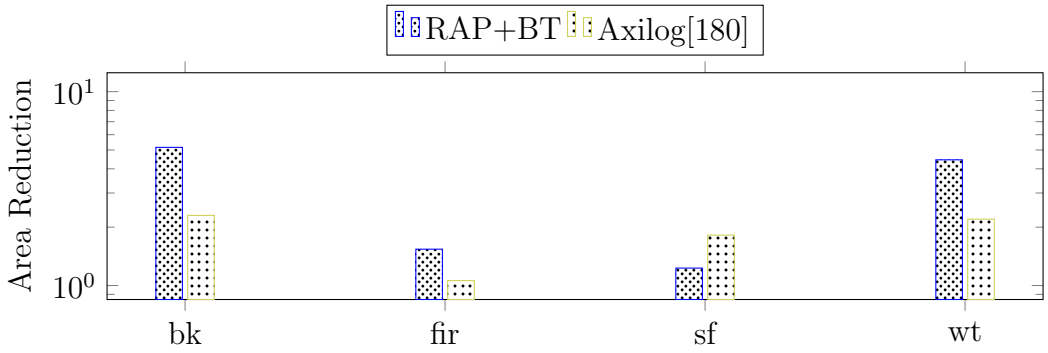


Figure 4.9: Area comparison of RAP+BT & Axilog[180]

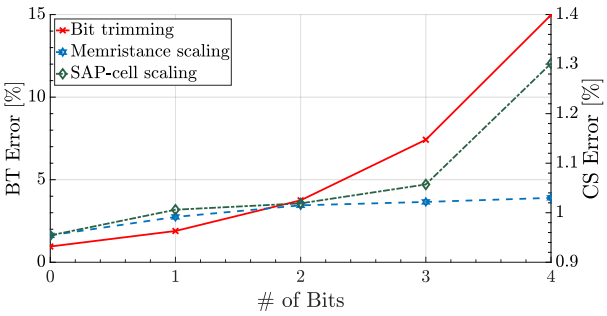
Overall, it can be concluded that the RAP architecture provides good approximation. This knowledge can be important while selecting an architecture for error resilient applications. To exemplify a use case, under a highly variable transmission medium where PSNR rate is changing in high rate, the RAP architecture can be employed to get maximum benefits from the approximate computing as well as the in-memory computing, so that precision of the data stored/processed in the RAP is dynamically tuned depending on the PSNR rate.

4.3 A Hybrid Approach

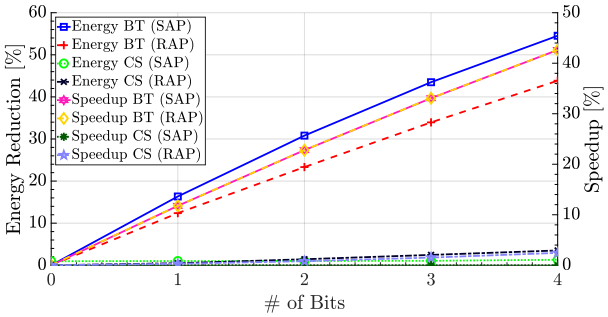
4.3.1 Motivation

As stated in Section 4.2, approximate computing in APs can be accomplished in two ways; bit trimming (BT) or cell scaling (CS) [174]. In bit trimming, one or more least significant bits of a number are disregarded and the operations start from the relatively more significant bits since the operations on the AP are done bit by bit from the least significant bit (LSB) through the most significant bit (MSB). In this way, the accuracy of a value in the AP is approximated by ignoring its less significant bits. This situation provides both energy saving and the speedup since the operations require fewer cycles (both compare and write). The second method introduced previously is the cell scaling. Although cell scaling can be used to denote scaling of many parameters (e.g., voltage, value range, area, etc.), in here, it corresponds to voltage scaling. This method can be applied to the ReRAM-based APs by scaling the R_{on} and R_{off} values of the ReRAMs in the approximate RCAM columns. In this case, the ReRAMs in the scaled columns (i.e., the approximated columns) are switched within a narrower range rather than the full range. This method necessities low write voltages (V_{WR}) together with shorter write pulses. In turn, it provides both energy savings and speedup. However, a compare operation on the columns including these approximated columns results in less robust matching and likely errors at the output of the sense amplifier. Similar to RAP cells, the SAP cells can be approximated by lowering their supply voltages, thus providing energy savings. The decrease in the supply voltage results in a decrease in the write energy and static energy. On the other hand, the cells whose supply voltage is less than the nominal voltage can introduce error during the operation, just like the RAP case.

Figure 4.10a shows the accuracy lost in an 1K, 16-bit FFT algorithm simulated on the environment described in Section 4.3.4. The figure shows how bit-trimming and cell scaling affect the accuracy in both RAPs and SAPs. The x-axis of the figure shows the number of



(a) BT and CS errors vs approximation methods



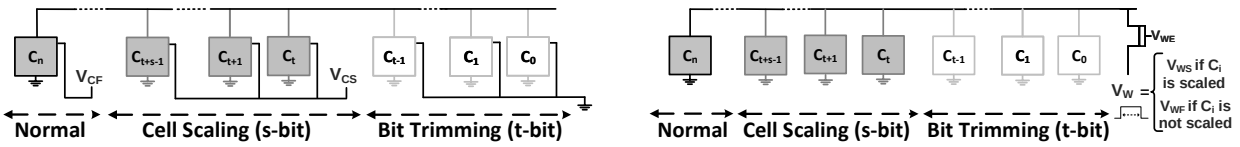
(b) Energy reduction and speedup vs approximation methods

Figure 4.10: The effect of bit trimming and cell scaling on SAPs and RAPs

trimmed or scaled bits and the y-axis shows the resulting accuracy in percentage where left-y axis shows the result for BT and right-y axis shows the result for cell scaling. The figure proves that the trimming single bit introduces more error to the system than the scaling same bit. When compared with the cell scaling (both ReRAM and SRAM scalings), the bit trimming leads to coarser grain approximation since bit trimming discards the column totally. On the other hand, the cell scaling still contributes to the operation somehow correctly or incorrectly in a probabilistic manner, so it becomes a finer grain approximation than the bit trimming. For instance, even though trimming 4-bit leads to 15% error in the result, scaling the same bits result in 1% and 1.3% error for memristance and SAP-cell scalings respectively. From the other point of view, given a quality expectation, the number of scaled bits can be more than the number of trimmed bits. Figure 4.10b shows the similar figure for energy reduction in left-y axis and speedup in right-y axis. The figure depicts that bit trimming provides more energy reduction and speedup when compared with cell scaling. As an example case, while trimming the LSB bit in SAP results in 15% energy savings and 12% speedup, scaling same bit provides 1% energy reduction and 2% speedup.

From Figure 4.10a, it can be inferred that bit-trimming provides more speedup and energy at the expense of more decrease in the accuracy. On the other hand, cell scaling exists as a more robust approximation technique than bit trimming. In approximate computing systems, the main goal is getting the optimum performance with the attainable highest acceptable quality.

For this purpose, bit-trimming in APs can be more reasonable when applied to the LSBs which have less effect on the computational accuracy than the MSBs. On the other hand, cell-scaling can be done to tune the degree of approximation (in a finer grain manner) in the remaining bits after applying bit-trimming on some LSBs. Even though cell scaling provides a finer grain approximation with much lower improvements in performance and energy, this method can be very useful for such cases where the bit-trimming methods decrease the overall quality lower than the expectations. As a combination of these two techniques, the proposed hybrid approximate computing approach targets to find the optimum degree of approximation in APs with respect to a given quality metric by efficiently exploiting both methods.



(a) Approximate computing in SAP (b) Approximate computing in RAP

Figure 4.11: Hybrid approximate computing in associative processors (APs)

Figure 4.11 shows the proposed hybrid approximation technique on SAPs (Figure 4.11a) and RAPs (Figure 4.11b) respectively. In both APs, the bit-trimming is done by disregarding the LSBs from the computation. In this way, the computation in the APs starts from relatively more significant bits. In RAPs, the trimmed bits do not cause an additional overhead since ReRAMs are non-volatile storage. Specifically, in SAPs, the supply voltage of the trimmed columns can also be cut off in order not to cause excessive static energy consumption. After trimming some LSBs, the cell scaling is applied on the number of columns which has relatively higher significance. However, cell scaling introduces less error to the system than bit-trimming, so the system can still get more advantage from the approximate computing still by complying with the quality metrics specific to the application. In SAPs, the scaling is done by applying a scaled voltage (V_{CS}) to the scaled cells that is less than the full cell voltage (V_{CF}), i.e., $V_{CS} < V_{CF}$. In RAPs, the scaling is performed by changing the write

voltage where V_{WF} is the write voltage for the full cells and V_{WS} is the write voltage for the scaled cells. For example, if cell-0 (C_0) is scaled, the V_{WS} voltage is applied to write to the corresponding memristor. Apart from the voltage levels, the pulse widths are different for each cases since memristance scaling decreases the write time.

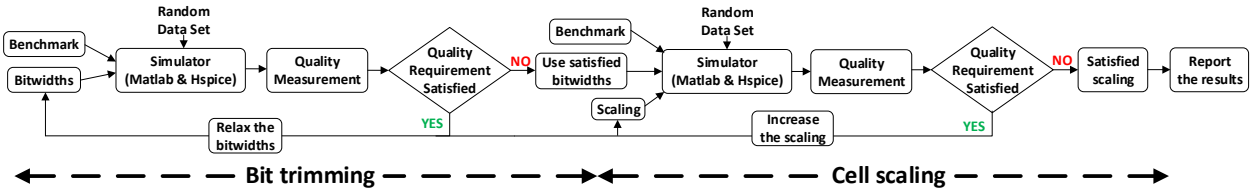


Figure 4.12: The design flow for approximate AP systems

4.3.2 Design Flow

Deciding on the number of trimmed bits and scaled columns is a design parameter and depends on the application and its requirements. The design flow shown in Figure 4.12 demonstrates a process to select the optimum configuration. The main item in the flow is an in-house simulator developed specifically for both APs (see Section 2.5). As input, the simulator accepts the benchmark (instructions), system/circuit parameters (bit width, variations, scaling, transistor/memristor models, etc.), and input data. The simulator outputs the energy consumption, accuracy (quality), and timing results of the simulated application.

The design flow consists of two phases; configuring the bit trimming, and configuring the cell scaling. Since bit trimming can be considered as a coarser grain approximation when compared with the scaling, the flow firstly targets to find the optimum bit width approximation in the first phase, and then apply optimum scaling on top of bit trimming approximation in the second phase. First, both approximation parameters (bit widths and scaling) are set to the most relaxed case so that the system performs the operation on the full range without any scaling. After the simulation, the quality is assessed for at least n runs on different random input sets (we use $n = 10$ in our experiments). If the quality is satisfied, the bit

width constraint is relaxed (more bits is trimmed) and a new simulation is done with the new constraints. This loop continues until finding a bit trimming case which does not satisfy the quality requirements. After finding this case, the parameters set to the optimum bit widths (the preceding one) and the second phase (cell scaling) starts. During the cell scaling, the same flow is performed by relaxing the scaling parameters (i.e., the number of the scaled columns). The key point in this flow is setting the quality requirement which is a configurable parameter depends on the application itself. After the optimum scaling parameters found within the given degree of bit trimming, the results are reported in the last stage of the flow. The results show a map to indicate which bits are trimmed and which ones are scaled.

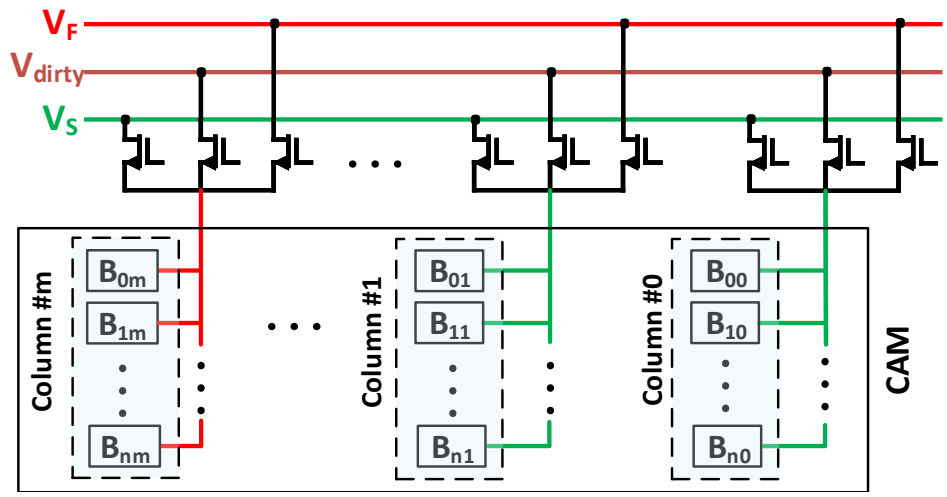


Figure 4.13: Dynamic cell scaling in APS

4.3.3 Dynamic Approximation

Both the cell scaling and the bit trimming can be done on LSBs and can be defined in the design time by using the explained design flow. Moreover, since the starting point of the operations can be modified on the fly by the controller, approximation of the bit-trimming can be done dynamically. For bit-trimming, the controller simply shifts the starting bit position of the operands. For example, to trim a single bit from the operand A in

Figure 2.12, the controller runs the instruction of *Multiplication([7,5],[3,2],[1,1])* instead of *Multiplication([7,5],[3,2],[1,0])*, that is, the operands can be changed dynamically by the controller. Additionally, the degree of scaling can be controlled by a circuit during runtime by using adaptive supply voltages. For example, the method in [128] proposes a fast technique to change the voltage supplies. Figure 4.13 shows the general methodology of this technique where each column can be supported by a different voltage source. The controller unit of the APs decides on the cell supply voltage of the SAP cells and the write voltages of the RAP cells for writing logic-0 and logic-1. The applied voltage pulses also differ in RAP cells. Even though there are more than two voltage levels with different polarities for the memristor scaling, only one voltage control circuit is required per column. Therefore, the additional overhead is minimal and largely amortized over the thousands of rows comprising the AP. In the figure, it is shown that while column-0 and column-1 are supplied at the scaled voltage V_S (V_{CS} in SAPs and V_{WS} in RAPs), the last column (i.e., the most significant one) is supplied by full voltage V_F (V_{CF} in SAPs and V_{WF} in RAPs). Accordingly, the proposed hybrid approximation method in the APs is also dynamically tunable for both the bit trimming and the cell scaling. Therefore, at any time the system can increase the accuracy by shifting the starting bit of the operations to the right (i.e. trimming fewer LSBs) at the expense of increased operation delay and energy. This can be partially counterbalanced by changing the degree of cell scaling. Depending on the characteristics, resiliency, tolerance, and environmental conditions of the application, the system accuracy can be set by tuning these two approximation parameters dynamically.

Figure 4.14 shows an example waveform of dynamic voltage scaling by using the short-stop technique in [128]. The figure shows the two compare cycles on four bits that belongs to the multiplication operation in Figure 2.12. In the first cycle (0 ns - 1 ns), three of the columns are supplied with the full voltage (V_{CF}) and the least significant one does with scaled voltage (V_{CS}). As a result, the noise margin between the match and mismatch operation is resulted as 143 mV. At some point, the system (or controller specifically) decides to change the

degree of approximation by increasing the accuracy of the scaled bit in the expense of more energy consumption. During this change, the supply voltage of these columns are boosted from 0.5 V to 0.7 V. In [58], the time required to boost voltage takes less than 5 ns. In the second compare operation on the same bit (between 6 ns - 7 ns), the system results in a more robust operation with a noise margin of 197 mV. At any time, this voltage can be lowered for the sake of less power consumption. The same voltage boosting technique can be applied to RAP systems but to change the write voltage to set the memristance range.

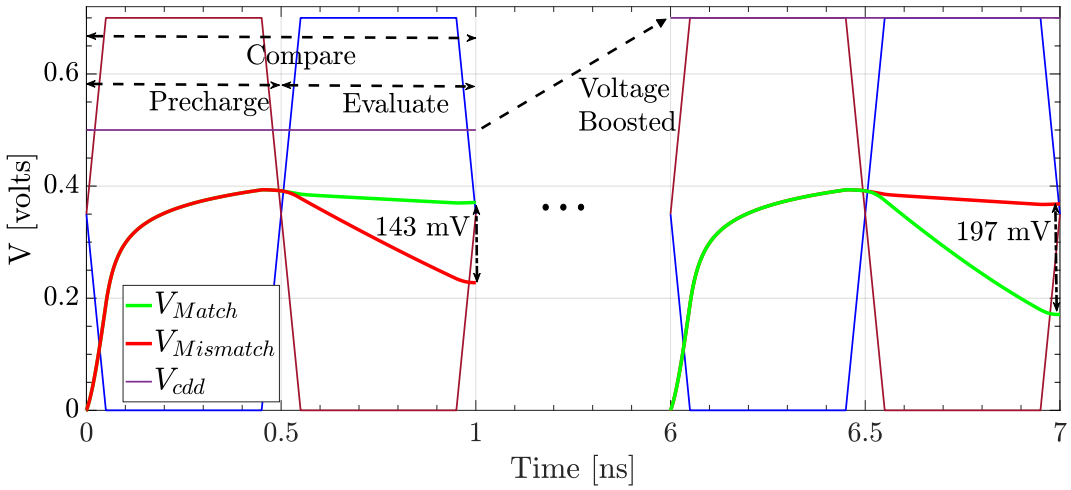


Figure 4.14: Dynamic approximate computing in SAPs

4.3.4 Experimentation

Experimental Setup

For the evaluation of the hybrid approximate computing in the APs, energy reduction, speedup, and energy×speedup results for 7 benchmarks from AX-BENCH (approximate computing benchmark) tool [179] presented in Table 4.3. The table also presents the number of input bits of the benchmarks. In some cases, the benchmark accepts more than one input with different bit widths. For this reason, the type of the input is specified with a letter (e.g., d for data and e for twiddle factor in FFT). Even though the AP accepts these bit

Table 4.3: The evaluated benchmarks, their platforms, and quality metrics from [179]

Benchmark	Domain	Bit Size	Quality Metric
Sobel Filter (sf)	Image Processing	p8 (12)	Image Diff.
FIR (fir)	Signal Processing	d8 & c8 (17)	Avg. Relative Error
FFT (fft)	Signal Processing	d16 & e16 (30)	Avg. Relative Error
Image Binarization (ib)	Image Processing	p8 (9)	Image Diff.
Convolution (conv)	Machine Learning	d24 & e16 (43)	Avg. Relative Error
FastWalsh (fwht)	Signal Processing	p17 (35)	Image Diff.
Mean Filter (mf)	Machine Vision	p8 & c16 (28)	Image Diff.
CNN	Deep Learning	d8 & k8 (18)	Avg. Relative Error

* **d:data, p:pixel, e:twiddle factor, c:coefficient, k:kernel, (): intermediate bit width**

sizes as input, during the computation the bit widths of the intermediate results can be more (e.g., multiplication of two m-bit numbers results in a 2m-bit number). The maximum bit widths of the intermediate results are specified within the parentheses. In the system level and circuit level simulations, the cycle accurate simulator in [174] is used. For the transistor in the circuit level simulation, Predictive Technology Models (PTM) from [159] are used to simulate high-density APs with sub 20nm feature sizes [151]. In the RAPs, we adopt a fabricated nano-second switching time memristor which has a size of 50 nm [114]. In the simulations, its corresponding SPICE model in [168] is used. For the sense amplifier, a low-power, sub-ns amplifier design in [145] is employed in the circuit. For both APs, the pre-charge voltage is set to 0.5v. Since the AP operation outputs more mismatches than the matches statistically, the V_{th} is selected to make the mismatch error rate as low as possible [175]. In all simulations and comparisons with other architectures, the maximum quality degradation is set as less than 10% (based on the metric in [179]).

Approximate computing forces a system to work on the boundary conditions. Under these conditions, the factors deemed to safe under normal conditions endanger the correctness of the system. For example, a scaled SRAM-cell can continue normal operation without any error in the circuit simulations. However, if the variations are taken into account, the system becomes more vulnerable to the errors. In a similar manner, a scaled ReRAM cell can decrease the voltage level across the capacitor under the V_{th} level and it is misinterpreted as

logic-0 even though it would be logic-1 (i.e., there is a match). Therefore, an approximate system must be evaluated by regarding the variation sources which does not pose a threat to the non-approximate one. For this reason, to make the results as reliable as possible, the variations arising from the voltage sources and the memristive devices are inserted into the system. For memristance distributions, the distribution is taken from an experimental data [88] on the fabricated memristors by fitting it into the normal distribution. For voltage sources (pre-charge voltage V_{pc}, SRAM-cell supply voltage V_{cdd}, write voltages V_{wr}, and threshold voltage V_{th}), the normal distribution with a 3% sigma is assumed even though sigma of variations are reported as 2.5% in other studies [79] [80]. Since memristor variations is also a result of variations in the write voltage, another variation onto the ReRAM-cell write voltage is not included. During the simulations, these variations are all modeled as different distributions and the random circuit parameters are generated from these distributions. The circuit simulator (HSPICE) is run several hundred thousands of cycles to obtain the metrics under these variations. The method presented in [175] is used to make both RAP and SAP as resilient as against these variations where V_{th} of the sense amplifier is set to make the error probabilities as few as possible.

Table 4.4 shows the cases and their corresponding parameters and results for both SRAM and ReRAM cell scalings. The last column in the table shows the probability of average error

(a) SAP cell scaling cases

Case	$\mathbf{V_{cdd}}$	$\mathbf{P_{static}}$	$\mathbf{E_{write}}$	$\mathbf{P_e}(avg)$
Full	0.7 V	0.52 µW	0.24 fJ	0
Approximate	0.5 V	4.66 nW	0.06 fJ	0.021

(b) RAP cell scaling cases

Case	$\mathbf{R_{on}}$ (Ω)	$\mathbf{R_{off}}$ (Ω)	$\mathbf{T_{write}}$	$\mathbf{E_{write}}$	$\mathbf{P_e}(avg)$
Full	100	100 k	2 ns	21.7 pJ	0
Normal	1.7 k	79 k	1 ns	349.6 fJ	0
Approximate	3.2 k	40 k	0.5 ns	121.8 fJ	0.027

Table 4.4: Cell scaling in AP for both SAP and RAP

($P_e(avg)$) which the corresponding case causes after a compare operation. For the scaling in the SAP (Table 4.4a), the supply voltage of the scaled columns is decreased to 0.5V from the nominal voltage of 0.7V. This decrease leads to a decrease in both static energy and the write energy. On the other hand, the reliability of the cells decreases as well. During a compare operation, in the case of a mismatch, these cells increases the resistivity through the leaking path, and in some cases (e.g., more than one scaled columns are included in a compare operation), the compare operation may result incorrectly. The cell scaling in RAP is done by shrinking the write time and voltage. Table 4.4b shows the three different cases (*full*, *normal*, *approximate*) for the ReRAM scaling. In order to find these cases, the design space exploration is performed on the memristor model in [168]. The important concern in here is that the switching between R_{on} and R_{off} must be symmetric, that is, the state of the memristor can be switched in both directions within the same period. On the other hand, the voltage levels and polarities can be different depending on the memristor model as usual. In the table, the first row corresponds to the full case where the ReRAM is switched within its binary range (100 kΩ-100 Ω). The normal case shows a not-aggressive scaled case that does not pose a threat on its own still under the variability conditions. The last case presents the approximate case where the ReRAMs are scaled aggressively. This case results in the least energy and the most speedup with the expense of errors. During the experimentations, the full case is used for the most accurate case where there is no approximation. For the approximate computing in RAPs, the normal case is used for the most significant digits and the approximate case is used for the least significant digits.

Evaluation

In the evaluation of the proposed hybrid approximate computing methodology, firstly, the proposed architecture is compared with previous work which proposes the bit-trimming and the cell-scaling individually for APs [174]. Table 4.5 shows the comparison in terms of energy

(a) SRAM-based associative processor (SAP)

	Hybrid				Bit Trimming				Cell Scaling			
	Configuration	Speedup	Energy	E x S	Configuration	Speedup	Energy	E x S	Configuration	Speedup	Energy	E x S
SF	10s2t	1.24	3.41	4.24	2t	1.24	1.46	1.82	12s	1.00	2.54	2.54
IB	5s3t	1.60	2.40	3.85	3t	1.60	1.83	2.93	8s	1.00	1.45	1.45
CONV	4s16t	2.74	4.61	12.64	16t	2.74	4.04	11.08	26s	1.00	2.10	2.10
FWHT	14s2t	1.13	2.72	3.08	2t	1.13	1.22	1.38	14s	1.00	2.05	2.05
FFT	17s10t	2.07	6.47	13.37	10t	2.07	2.57	5.31	24s	1.00	2.40	2.40
FIR	4s5t	1.95	4.72	9.22	5t	1.95	2.40	4.69	16s	1.00	2.41	2.41
MF	10s14t	3.29	10.15	33.34	14t	3.29	5.09	16.72	24s	1.00	3.06	3.06
CNN	6s7t	2.74	6.88	18.83	7t	2.74	4.06	11.12	16s	1.00	4.48	4.48
Average		2.10	5.17	12.32	-	2.10	2.83	6.88	-	1.00	2.56	2.56

(b) ReRAM-based associative processor (RAP)

	Hybrid				Bit Trimming				Cell Scaling			
	Configuration	Speedup	Energy	E x S	Configuration	Speedup	Energy	E x S	Configuration	Speedup	Energy	E x S
SF	10s2t	2.04	48.16	98.11	2t	1.75	37.92	66.47	12s	1.64	37.61	61.63
IB	5s3t	2.67	55.54	148.11	3t	2.29	41.02	93.77	8s	1.67	41.68	69.46
CONV	4s16t	3.95	52.48	207.34	16t	3.89	51.19	199.31	26s	1.64	22.93	37.71
FWHT	8s2t	1.71	41.27	70.49	2t	1.59	35.11	55.70	12s	1.55	36.92	57.06
FFT	9s16t	3.14	39.03	122.39	12t	2.93	35.61	104.42	20s	1.63	21.44	34.92
FIR	4s5t	2.91	49.52	144.05	5t	2.79	45.39	126.53	12s	1.67	26.65	44.41
MF	8s12t	5.24	97.46	510.62	14t	4.69	81.58	382.91	23s	1.71	35.40	60.58
CNN	6s7t	4.26	89.45	381.35	7t	3.91	73.31	286.81	15s	1.72	37.47	64.36
Average		3.24	59.11	210.31	-	2.98	50.14	164.49	-	1.65	32.51	53.77

Table 4.5: Comparison of approximation methods on SAP (a) and RAP (b) with different benchmarks with 10% maximum quality degradation

* **t: trimmed, s: scaled**

reduction and speedup for SAPs (Table 4.5a) and RAPs (Table 4.5b). In addition to the convenient AxBenchs in Table 4.3, the figure also includes the result for the convolutional neural network (CNN). For CNN implementation on the AP, only parts of the neural network are implemented since the whole CNN simulation (as cycle-accurate) would take months. The table also includes the configuration information for the approximation technique obtained from the design flow (see Figure 4.12). In the table, if bit is trimmed, the actual bit-width becomes less than the one presented in Table 4.3. The *t, s* stands for the trimming and scaling respectively and the number before these abbreviations indicate the number of trimmed and scaled bits of the corresponding input, respectively. The number of trimmed or scaled bits are shown over the total number of bits of the intermediate results presented in Table 4.3. In energy reduction, when the hybrid methodology is applied to the system, it outperforms in both SAP and RAP architectures when compared with the other methods. For example, the hybrid approximation provides an advantage in energy reduction between 14.1% to 151%

for the SAPs and 2.5% to 35.4% for the RAPs over the bit trimming. As presented in the configuration of each approximation method, this savings mainly comes from less error contribution of the cell scaling method which in turn allows to scale more bits. Over the cell scaling, the advantage in energy reduction becomes between 32.5% to 231.7% for SAPs and 11.8% to 175.3%. The main reason of the higher energy savings in the SAP architectures is that the SAP architecture provides more scaling than the RAP architectures since its error probability is less. Additionally, the cell scaling in the SAPs contributes to both compare and write cycles as opposed to RAPs in which only write energy reduced. Furthermore, scaling in RCAM cells increases the compare energy of RAPs since the residual charge across the capacitor goes to a lower level than the normal case after a compare operation due to the more leaky memristors. This was a side-effect of memristance scaling, but in overall the scaling provides an advantage. It is worth to mention in here that hybrid approach is done on top of the AP architecture where bit trimming is applied. In other words, after applying bit trimming, the cell-scaling is performed to fill the remaining quality allowance to obtain more energy reduction and performance within a 10% quality degradation limit. For the speedup in the SAPs, the hybrid approximation has no advantage over the bit trimming method since the scaling in the SRAM cells does not provide a speedup, on the other hand, there is an advantage gain of average 9.3% in the RAP architectures of the hybrid approximation. If the application allows further approximation through the cell scaling after bit trimming, the hybrid method can get more benefit from it. As an example, in FFT, after trimming four bits from the data, the four more bits from the remaining number can be scaled to maximize the energy reduction and the performance. However, an error introduced at the beginning of the in-memory computations propagates through all computations until the end. For this reason, if the benchmark is complex and requires relatively more operations, the benefit from the approximate computing becomes limited. For instance, a separable convolution operation consists of 2 pipelined FFT operations and requires data to pass through many butterfly stages. For this reason, this application allows limited approximation and its energy

reduction and speedup are less than the other benchmarks'.

Table 4.5 proves that the hybrid approximation technique has advantage over both previous approximation techniques applicable to in-memory associative processors. On the other hand, from the broader perspective, the indication of how APs performs in approximate computing is another topic. Figure 4.15 demonstrates the performance results of the different architectures and approximation techniques. In the results, hybrid approximation on the RAP and SAP architectures are compared against loop perforation (LP) [149], neural processing unit (NPU) [29, 181], resistive CAM accelerator [56], and Axilog HDL [180] a design tool for approximate ASIC designs. In the comparison, the evaluated results from [179, 56] are compared against hybrid approximation results. In reporting results, the relative energy reduction and speedup are reported which are the ratio of non approximate case over approximate case. Figure shows how hybrid approximation has a benefit over the reported other approximations. When compared with SAP, approximation in ReRAM-based AP provides better results in terms of energy reduction and speedup. This is because that a scaling in the write operation of the ReRAM cells returns a huge advantage in both factors. Even though all these have different architectures and technologies and seem as non-comparable on the same basis, the results shows shows how well approximate computing achieves on these architectures with different methodologies. In other words, the aim in here is to prove that associative in-memory computing with hybrid approximation has inherent advantages that make it better suited for approximate computing, achieving higher relative improvement in all three figures of merit (performance, energy and energy performance product).

4.4 Conclusion

In this chapter, the two promising computing methodologies, which are approximate computing and in-memory computing are combined which facilitates the approximate in-memory

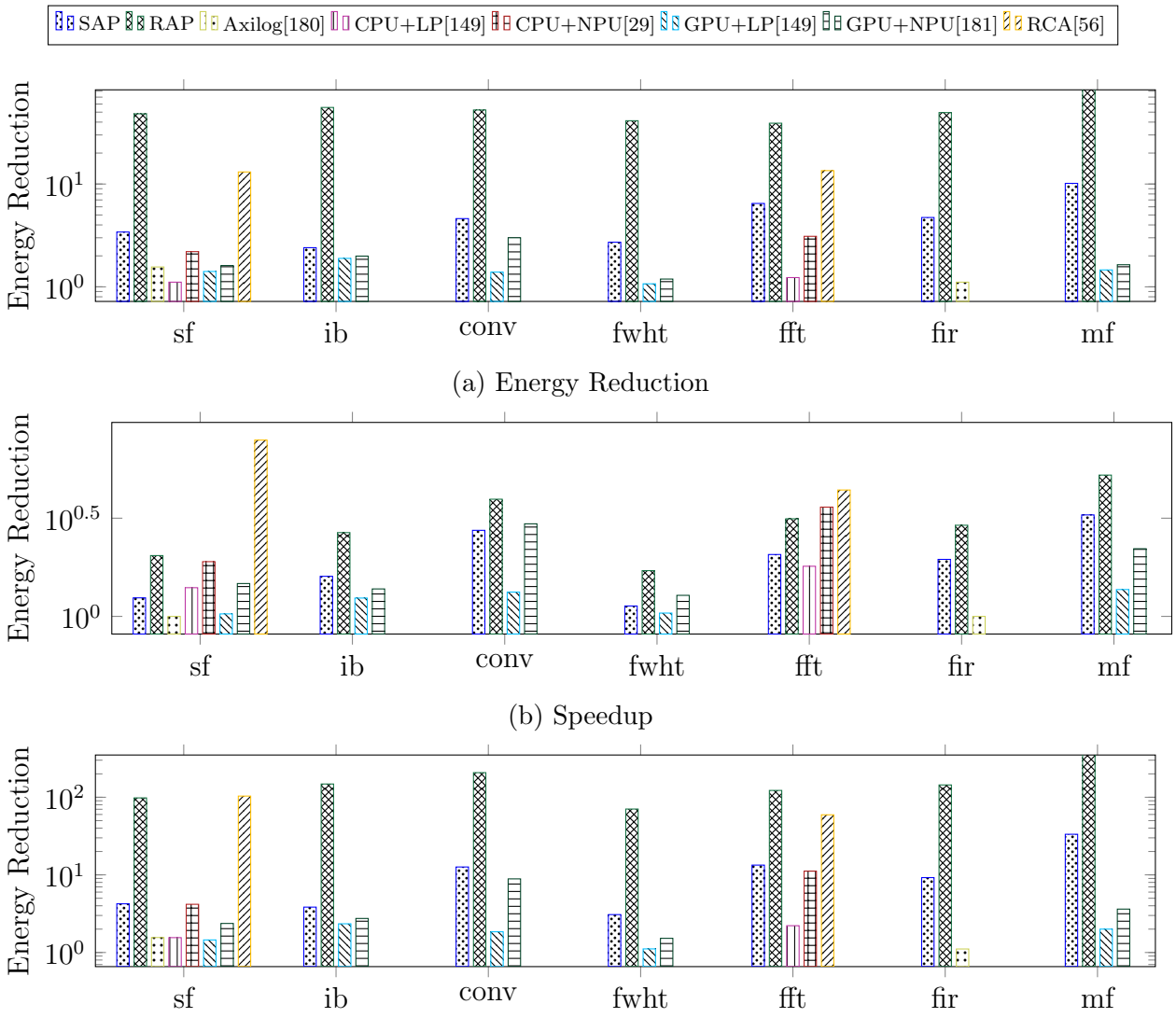


Figure 4.15: Comparison of approximation methods on SAP, RAP, ASIC, CPU, and GPU platforms with different benchmarks with 10% maximum quality degradation

computing. Two approximate computing methodologies in APs are proposed which are cell scaling (both SRAM and ReRAM) and bit trimming. The proposed methods are naturally supported by the APs as dynamic and tunable. The reliability concerns coming with the approximation are also inspected. The suitability, practicality, and reliability of this methodology are investigated through the SPICE-based circuit simulations. The competency of the proposed method is proven by benchmark results. It is also proven that APs supports approximate computing inherently and provides a better employment place than other architectures for approximate computing since it facilitates dynamically (run time) tunable approximation.

Chapter 5

Methods for Low-power APs

In this chapter, a low-power AP implementation by proposing architectural and instructional improvements to decrease the switching activity are proposed. This chapter focuses on some methodologies to decrease the energy consumption of for both SAPs and RAPs.

5.1 Low-Power SAP

Even though APs are a good solution for the memory bottleneck, their energy efficiency is limited. However, there are limited studies on this problem since this approach has been regaining importance recently. As an example, in [174], the energy savings in the APs are obtained by approximate associative computing where some bits in the CAM are either extracted from the computation or the switching range of the memristor is shrunk. Similar to APs, there are other low-power methods applied to other in-memory computing architectures or CAMs used for similar purposes. For example, in [57], an architectural extension to GPUs are proposed to avoid the frequent re-executions by recalling their results directly from a cache-like resistive CAM (RCAM) and memory structure tied to the every FPU in the GPU.

The CAM structure facilitates approximate matching, therefore providing energy savings at the expense of an acceptable decrease in quality. In [124], a low power in-memory computing platform using a novel 4-terminal magnetic domain wall motion (4T-DWM) device is proposed for in-memory computing. An approximate processing in-memory architecture (called APIM) is proposed in [51] which uses the analog behavior of memristors in addition and multiplication operations.

To realize the low-power associative computing methods to aid in fulfilling the limitations of current computing paradigms, the cycle accurate behaviors of the AP operations can be analyzed to detect the possible deficiencies which cause surplus energy consumption. Even though APs propose a good solution for the memory bottleneck, the huge switching activity on the rows happens during the operations poses an issue on power density. If these activity can be diminished through the circuit or operational modifications, the power consumption of APs can be decreased as well. The following section (Section 5.1.1) presents the main motivation behind this idea.

5.1.1 Motivation

Figure 5.1 shows the waveform of one-bit subtraction where the operations corresponds to the second row in the 4-bit subtraction example (see Figure 2.10). The figure shows the voltage changes across the row capacitors (V_{SARX} where X corresponds to the row number) together with the reference threshold voltage (V_{th}). In the figure PC, E, WR labels correspond to the pre-charge, evaluate, and write phases respectively. After an evaluate cycle, if the voltage drops below the threshold, the sense amplifier outputs a logic-0 and logic-1 if the voltage is above the threshold.

The figure shows that row 0 matches in the first compare phase. However, in the usual process of APs, it is impossible to get a second match within a LUT pass. In other words, it

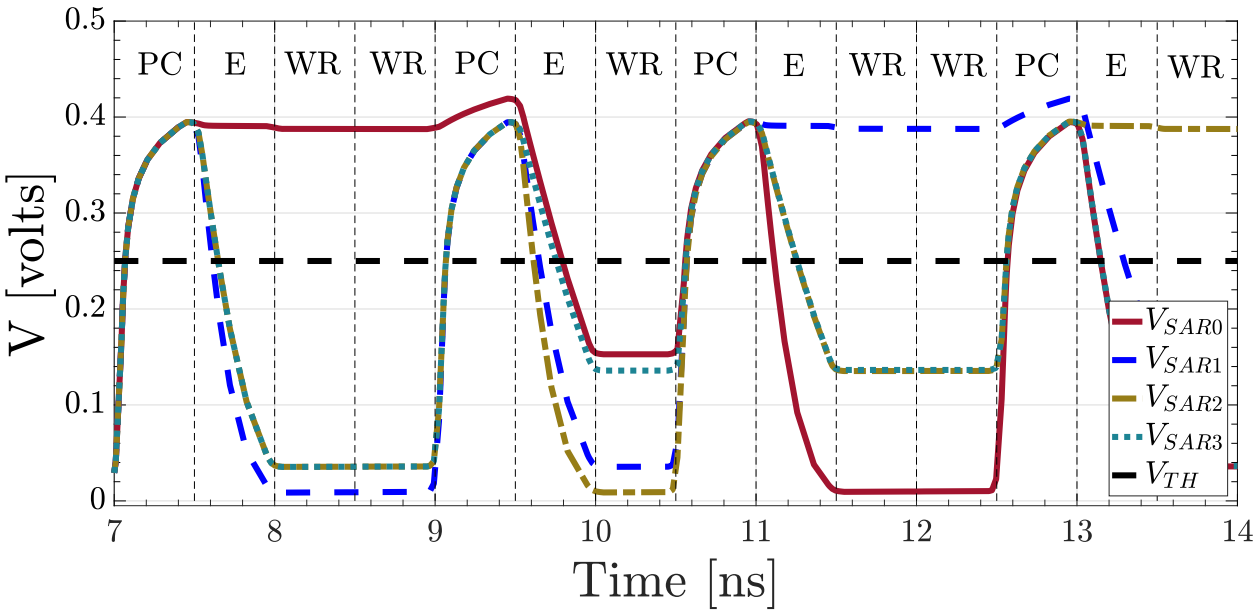


Figure 5.1: Waveform of single-bit subtraction which corresponds to the second row of Figure 2.10

is not possible to get another match for the row 0 between the interval 9 ns and 14 ns. For this reason, the other cycles are simply wasted and the capacitor is charged and uncharged unnecessarily during these cycles.

If the data is assumed as randomly distributed with equal probability, the equation 5.1 gives the number of wasted cycles. In the equation, m, n_r, n_c, n_{lut} represent the bit width of the operand, the number of rows in the CAM, the number of compared columns, and the number of entries in the LUT of the corresponding operation respectively. If the operation is multiplication (not linear time), the m in the formula must be replaced with m^2. According to the formula, 18.75% of the compare cycles are wasted for the in-place addition operations on m-bit, n numbers.

$$f_w(m, n_r, n_c, n_{lut}) = m \cdot \sum_{i=1}^{n_{lut}} \frac{m}{2^{n_c}} \cdot (n_{lut} - i) \tag{5.1}$$

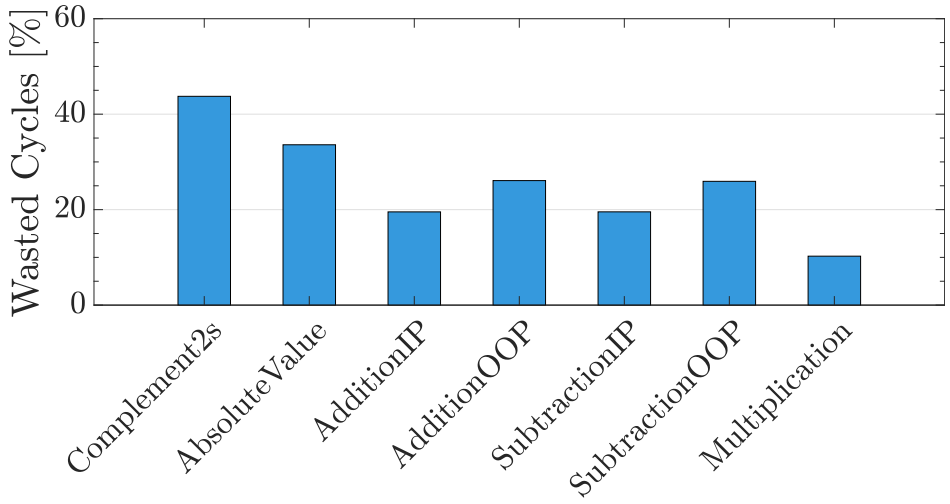


Figure 5.2: Unnecessary (wasted) cycle percentages of the fundamental arithmetic operations in the AP.

Figure 5.2 shows the percentage of the wasted cycles when these operations are performed on the AP simulator detailed in Section 2.5 on 16-bit 1024K random number pairs. The results indicate that percentage of the wasted cycles are more than theoretical results since LUT tables cover only the part of all cases. According to the figure, the unnecessary cycles range from 11.43% to 77.79% with a mean of 36.45%.

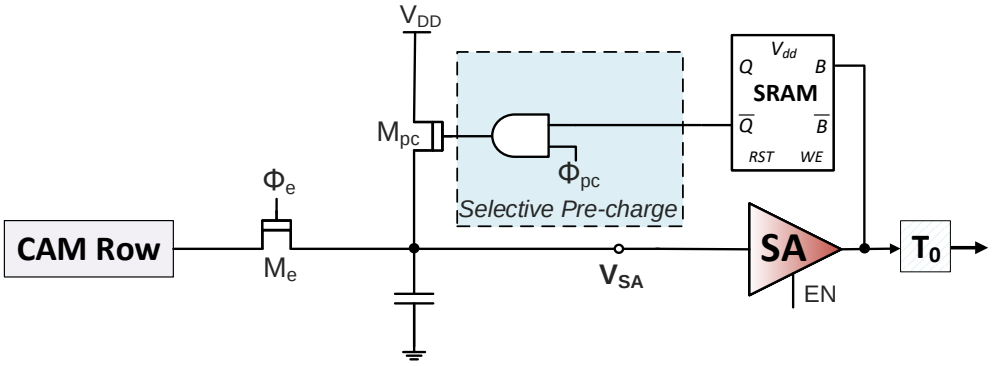
In addition to them, some operations spent unnecessary cycles not within a LUT pass but also within a group of LUT passes (i.e., in a longer period). For example, the absolute value operation in the AP performs unnecessary pre-charge and evaluate cycles even though the input number is positive. Similarly, the multiplication operation in the AP wastes these cycles even if the multiplicand is 0. All these cases in the AP cause the high switching activity on the rows and lead to more energy consumption consequently.

5.1.2 Low-Power Methodologies

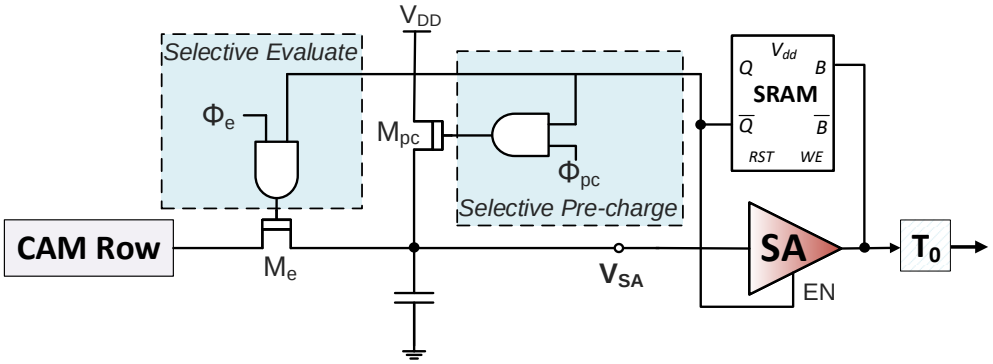
Selective Pre-charge

As discussed in Section 5.1.1, a considerable portion of the compare cycles are spent unnecessarily. This is because the row capacitors are pre-charged even though it is known as a priori that it is going to discharge definitely during the rest of the LUT pass. If a mechanism prevents these unnecessary cycles from occuring, the wasted cycles shown in Figure 5.2 can be eliminated for energy saving. In order to accomplish this mechanism, these rows must be differentiated from the others by tagging them so that in the next cycle the pre-charge operation can be done selectively only on the untagged rows. These tagging operations can be done by using a single-bit memory cell.

Figure 5.3a shows a single row of an AP modified to facilitate this mechanism as named *selective pre-charge (SPC)*. For tagging purpose, a single SRAM is added to the row. Initially, each SRAM stores the logic-0 value which means that these rows are the candidate for a possible match. Within a LUT pass, if a row is matched, the value of SRAM is changed as logic-1. A logic-1 value stored in the SRAM indicates that this row is matched in any of the previous cycles within a LUT pass and it is not possible that it will match again within the current LUT pass. At the end of each LUT pass, the SRAMs in the rows are reset to convert the all values to logic-0. In the figure, it is shown that the circuit also has modifications in the input of the pre-charge switch transistor (M_{pc}). Instead of directly controlling the switch by Q_{pc} signal coming from the controller, the output of the SRAM is connected to an AND gate together with the Q_{pc} signal so that this transistor is controlled by both signals. Even though controller pulls the Q_{pc} signal up to logic-1 in the beginning of a compare cycle, the capacitor cannot be charged if SRAM holds a logic-1. In this way, wasted pre-charge cycles are avoided.



(a) Selective pre-charge mechanism.



(b) Selective evaluate mechanism.

Figure 5.3: Selective pre-charge (a) and evaluate (b) mechanisms for low-power AP.

Figure 5.4a shows the waveform of the operation previously shown in Figure 5.1 when selective pre-charge mechanism is enabled. In the figure, the voltage across the row capacitance of the row 0 (V_{SAR0}) is matched in the first cycle and its SRAM register is set as logic-1. Since this register disables the pre-charging in the following cycles within a LUT pass, this capacitor cannot be charged in the following three pre-charge phases. In this way, the upcoming tree pre-charge cycles are avoided and the energy is saved. The same situation applies to the second row (row 1) after the third pre-charge.

Selective Evaluate

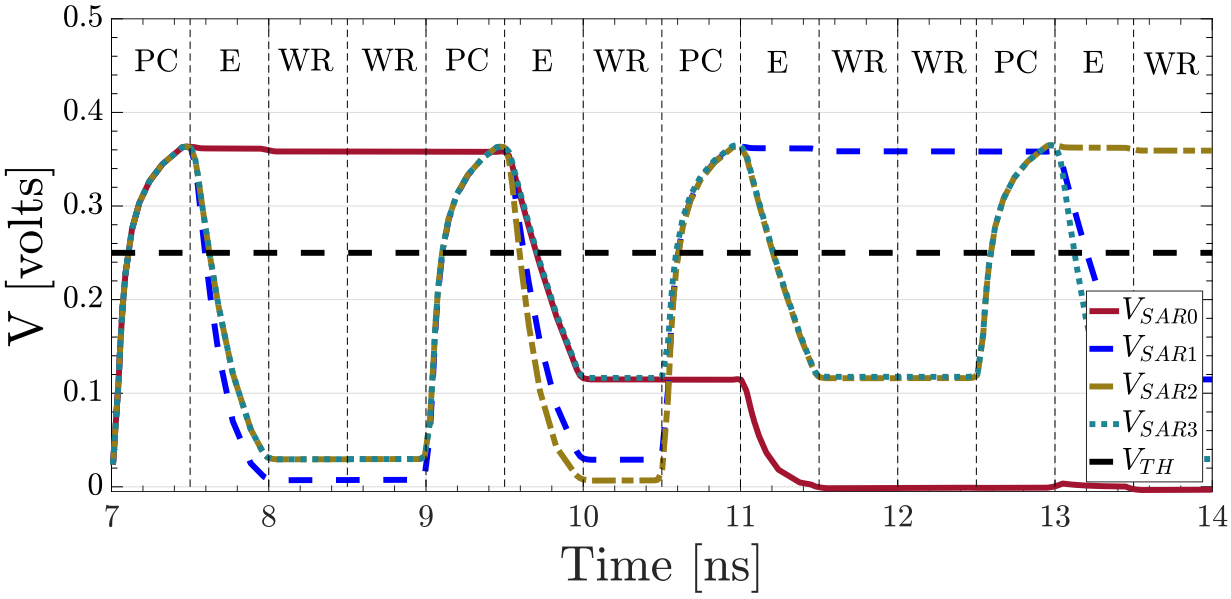
Selective pre-charge is an effective method in avoiding unnecessary pre-charge operations. On the other hand, the charge across the capacitor leaks in the following evaluate cycles

unnecessarily without preceding pre-charges. One improvement in addition to selective pre-charge can be *selective evaluate (SE)* where evaluate cycles can be performed selectively as well so that the charge across the capacitor cannot be lost and is locked in the capacitor for the next LUT pass. The same SRAM can be used for this purpose, however, the circuit needs another AND gate at the input of the evaluate transistor. Figure 5.3b shows the modified architecture to enable selective evaluate together with selective pre-charge. The combination of these two methods is called as *selective compare (SC)* since a compare operation consists of pre-charge and evaluate phases.

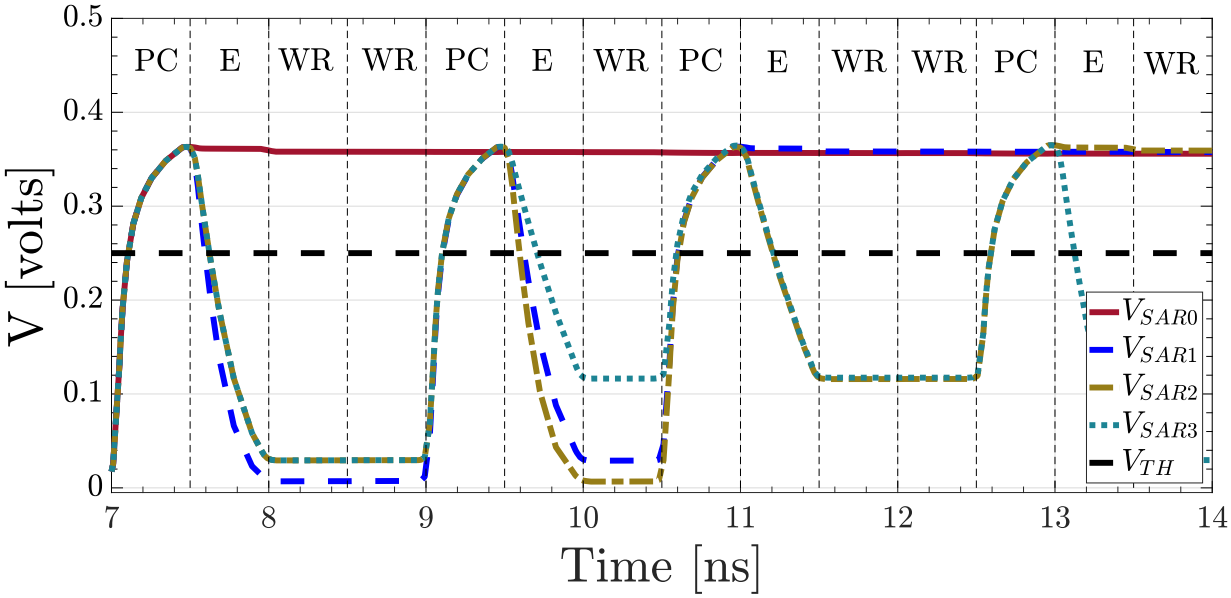
Figure 5.4b shows the waveform of the same operation in Figure 5.1 where both selective pre-charge and evaluate mechanism enabled. The figure demonstrates that after a match, the following pre-charge cycles are disabled as well as the charge does not leak during the following evaluate cycles. On the other hand, in the upcoming evaluate cycles, the preserved charge across the capacitor is interpreted as logic-1 in the sense amplifier. In order to avoid this situation, the sense amplifier is also disabled if SRAM stores a logic-1 by connecting its output to the sense amplifier enable input (EN) (see Figure 5.3b). In this way, the total energy consumption of the sense amplifiers can be decreased as well since unnecessary sensing operations are avoided like pre-charge and evaluate phases.

Modified LUTs

The methodology of selective compare requires an SRAM cell that is able to keep the one-moment history of the previous status. Instead of exploiting this opportunity in a LUT pass (i.e., short-term), this history can be used for a longer period for some AP operations. In this section, the same SRAM cell is used to lower the energy consumption further in the multiplication and absolute value operations by modifying their LUTs. This method is called as modified LUTs (ML).



(a) Selective pre-charge



(b) Selective pre-charge & evaluate

Figure 5.4: Waveform of single-bit addition which corresponds to the second row of Figure 2.10 when selective pre-charge (a) and evaluate (b) mechanisms enabled.

Multiplication: For the unsigned multiplication in the AP ($R = A \times B$), the LUT is applied to all bits of B for each bit of A. Indeed, this table performs the addition operation between B and R if the A's bit is logic-1. On the other hand, if A's bit is logic-0, the partial addition operation is still done even though it has no effect on the results ($R = R+0$) and all

the cycles are wasted. In here instead of enabling selective compare mechanism in fine-grain, it can be employed in the coarse grain to get more advantage from it.

Table 5.1: Modified LUT for the multiplication

Cr	*R*	*B*	*A*	*Cr*	*R*	*Comment*
X	X	X	0	-	-	Once
0	0	1	1	0	1	2nd Pass
0	1	1	1	1	0	1st Pass
1	0	0	1	0	1	3rd Pass
1	1	0	1	1	0	4th Pass

Table 5.1 shows the modified LUT for the low-power multiplication. The table adds an additional LUT entry which looks for a logic-0 in the column of A. This compare operation is performed once at the beginning of each partial addition (i.e., at the beginning of the inner loop). To illustrate, it is executed m times for the multiplication of two m-bit numbers where the total number of compare cycles is $4m^2$. If the A's bit is logic-0, this row is simply excluded from the following partial addition. However, during this operation selective compare cannot be performed as well since there is a single SRAM and it keeps the history whether A is logic-1 or 0 (not the history through the LUT pass). Since more rows are excluded by just checking single bit (average of 50% of the rows), it is expected to get more energy savings when compared to selective compare.

Table 5.2: LUT for Absolute Value

(a) sign = 0

A	*R*	*Comment*
1	1	-

(b) sign = 1

Flag	*R*	*Flag*	*R*	*Comment*
0	1	1	1	3rd Pass
1	0	1	1	1st Pass
1	1	1	0	2nd Pass

Absolute Value: In the APs, the absolute value operation is simply performed by performing 2's complement on negative numbers and copying positive numbers directly. However, while performing 2's complement, the compare cycles spent on the rows having positive

numbers are unnecessarily wasted. For this reason, the positive and negative numbers can be discriminated and the separate LUTs can be defined for each of them. In this way, while performing the operation on positive numbers (i.e., copy operation), the negative numbers are excluded and on negative numbers (i.e., the 2's complement operation), the positive numbers are excluded. Table 5.2 shows the modified LUT table. During the operation, first a logic-0 is searched on the sign bit of the numbers to exclude the negative numbers and then, copy operation (Table 5.2a) is performed on the positive numbers. After that, a logic-1 is searched on the sign bit to exclude the positive numbers and 2's complement operation is performed on the negative numbers.

Since the architecture supports only single-bit history, either SC or ML methods can be exploited in these operations, but not both. In order to decide on the better method, the percentages of the covered cycles by each method are presented in 5.3. The table shows that the ML method can cover more compare cycles that the selective compare method. On the other hand, the modified LUT is adding a small number of extra compare cycles while excluding many rows from the computation. The detailed analysis of both methods is revealed in Section 5.1.3.

Table 5.3: Percentage of covered compare cycles in SC and ML

	Selective Compare	**Modified LUTs**
Absolute Value	33.59%	46.86%
Multiplication	10.26%	48.45%

5.1.3 Experimentation

Simulation Framework

The proposed power reduction techniques are applied to SAP architectures. For the transistor model, the Predictive Technology Models (PTM) [159] is preferred to simulate high-

density memories with 16 nm feature sizes [151]. Performance metrics and statistics are obtained by cross-checking the output of both Matlab and HSpice simulations (see Section 2.5). For the sense amplifier, a low-power, sub-ns amplifier design in [145] is employed in the circuit.

Table 5.4: Average energy and power results of the SAP

	Energy	**Time**	**Power**
Compare (per row)	5.425 fJ	1 ns	5.425 µW
Write (per cell)	0.242 fJ	0.5 ns	0.484 µW
Static Energy (per cell)	0.002 fJ	0.5 ns	0.005 µW

Arithmetic Operations

Figure 5.5 shows the normalized energy consumption of the fundamental arithmetic operations on the low-power SAP. The results are obtained by performing the corresponding operations on 16-bit 1 M (2^{20}) numbers (in 2's complement and absolute value) or number pairs (in others). The figure shows energy results of the operations where SC and ML methods are applied. The reported results are normalized with respect to the energy consumption of each operation without any low-power methodology to fit them into the common scale. For absolute value and multiplication, the normalized energy consumptions of ML method are shown as well. All low-power SAP results include the energy overhead of the additional components such as AND gates, SRAMs, extra cycles due to modified LUTs, etc.

According to the figure, the energy reductions obtained from the selective compare methods ranging from 6.95% in multiplication to 38.92% in 2's complement with a mean of 21.58%. On the other hand, the modified LUTs method provides a reduction of 42.59% and 41.74% in absolute value and multiplication even though selective compare results in 29.67% and 6.95% savings respectively. It is proven that exploiting the single-bit history in a longer-term provides more energy saving in these operations. On the other hand, this method affects

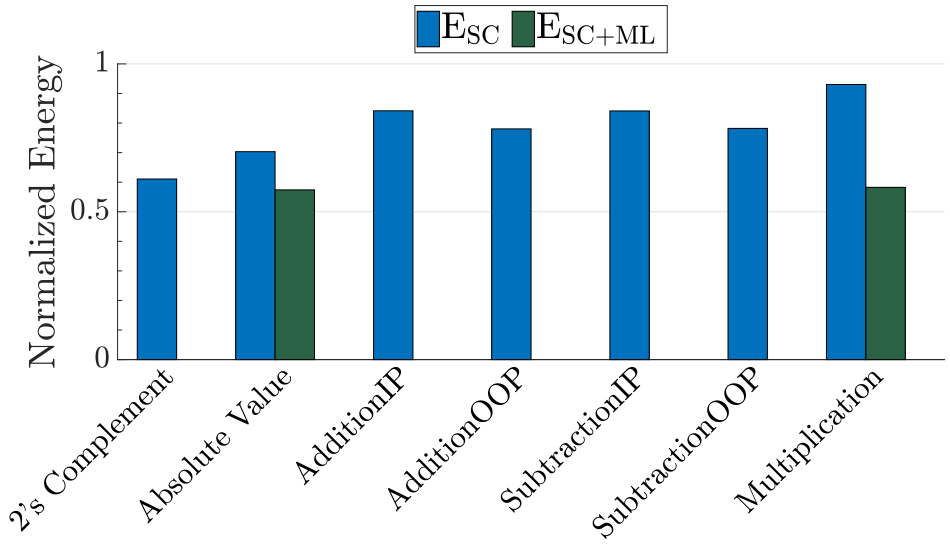


Figure 5.5: Energy reduction in arithmetic operations when selective compare and modified LUTs are enabled.

the performance since it inserts additional compare cycles during the execution. Figure 5.6 shows this overhead percentage when the bit-width of the operands ranges from 2 to 32. The performance overhead decreases as the bit-width of the operands increases. For a traditional bit-width of 16, the performance overheads are less than 2%. When compared with energy reduction, this performance overhead is negligible and can be ignored for the sake of energy savings.

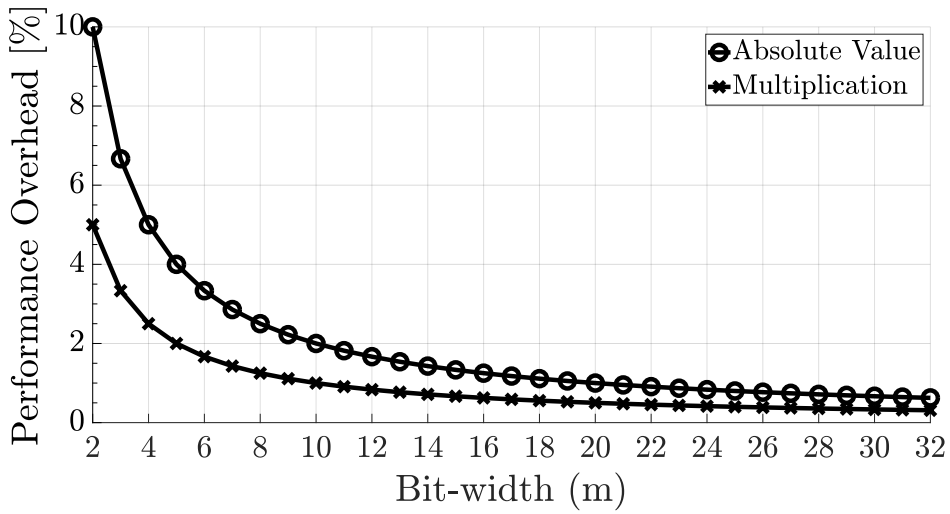


Figure 5.6: The performance overhead in 2's complement and multiplication due to the modified LUTs.

Table 5.5: The evaluated benchmarks and their input sizes

Benchmark	Domain	Parameters & Input
Sobel Filter	Image Processing	512x512 gray image
FIR	Signal Processing	8-tap, 512 x 8-bit integers
FFT	Signal Processing	1K 16-bit complex numbers
Image Binarization	Image Processing	512x512 gray image
RGB2Gray	Image Processing	384x512 color image
FastWalsh	Signal Processing	256x256 gray image
Mean Filter	Machine Vision	512x512 gray image

Benchmarks

For the evaluation of proposed low-power AP for the real cases, seven benchmarks from different domains are implemented on the AP. Table 5.5 shows these benchmarks, their inputs, and parameters during the evaluation. Both selective compare and modified LUTs methods are evaluated on these benchmarks. During the runs, AP is configured according to the corresponding benchmark, so in some cases, more than one AP is pipelined to each other. For example, a 1K FFT requires 10 AP stages. Since the circuit simulation time takes days for the successive iterations of the computationally intensive benchmarks (e.g., 10s thousand cycles for FFT), first, the accurate execution profile (dynamic & static energy, time, match & mismatch statistics, etc.) are obtained from the simulator (Figure 2.15), and later these numbers are used in the Matlab simulator to get the precise results.

Figure 5.7 shows the normalized energy consumption of each benchmark for both methods. If the benchmark includes either multiplication or absolute value, it becomes possible to reduce the energy consumption further by modified LUTs. For example, even though the normalized energy of the FFT is reduced by 4.16% when selective compare method is used, a 47.77% improvement is possible when modified LUT is enabled for absolute value and multiplication. The figure shows that exploiting modified LUTs can provide up to 45.51% more savings on top of selective compare. The one reason for this huge saving is that at the first stages of the FFT, the twiddle factors includes lots of zeros so the modified LUT provides

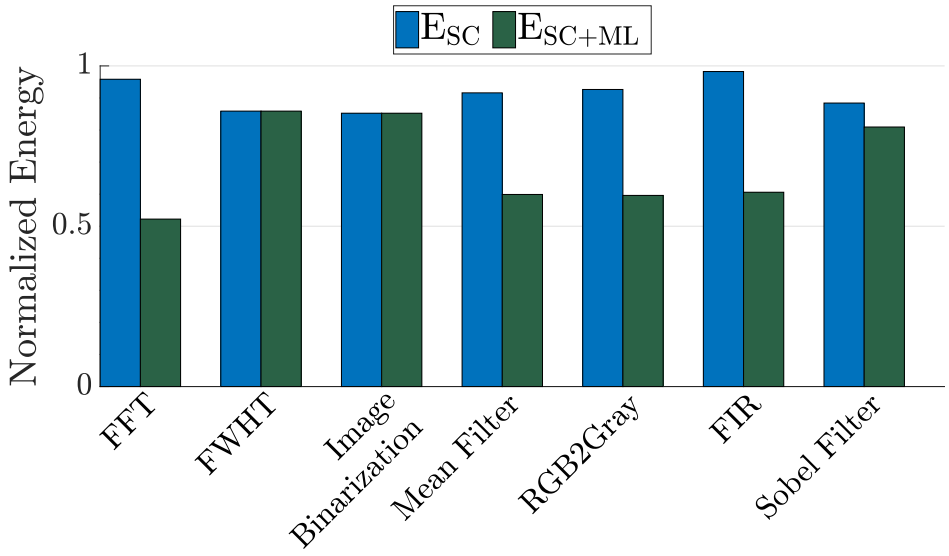


Figure 5.7: Normalized energy consumption of the benchmarks when selective compare and modified LUTs are enabled.

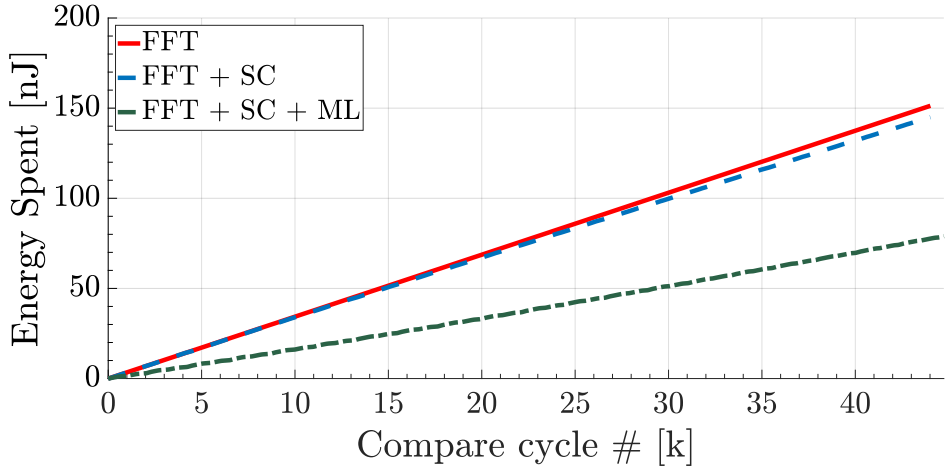


Figure 5.8: Energy consumption during the FFT benchmark runs of all three cases.

an excessive saving on these multiplications. In other benchmarks, the energy savings range from 14% to 40%. Figure 5.8 shows the energy trace comparison between the normal AP and the low-power alternatives. The figure shows the trend of the FFT benchmark since it includes nearly all instructions in Table 3.1. The reported energy is sensed at every compare cycle. The figure shows that the ML follows a much lower energy consumption trend.

As stated in Section 5.1.2, the proposed low-power methodologies cause area overhead in the circuit by inserting AND gates and SRAM cells. The figure 5.9 shows these overheads for

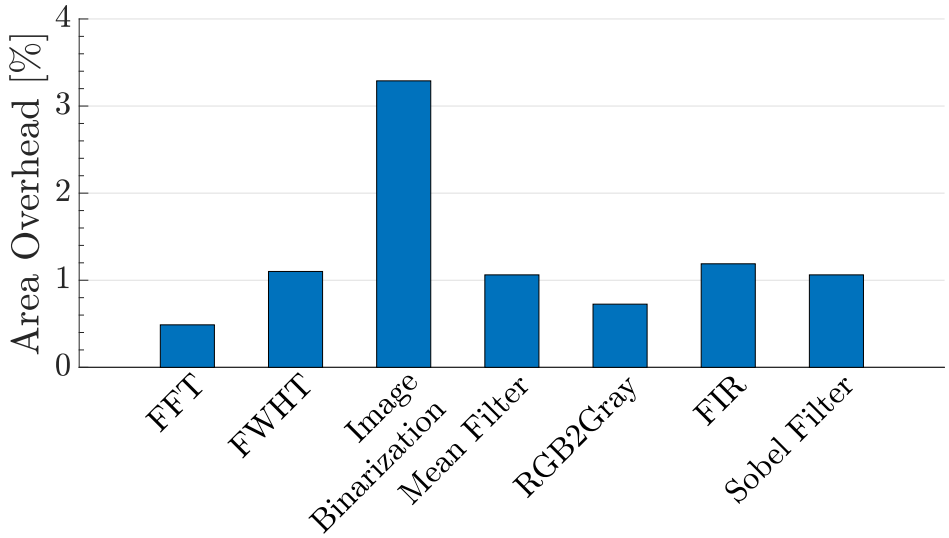


Figure 5.9: Area overhead of the benchmarks when selective compare and modified LUTs are enabled.

Table 5.6: Comparison with other sub-65nm ASIC implementations of FFT

	Technology	**Size** (*points*)	**Word Width** (*bits*)	**Area** (mm^2)	**Effective Throughput** (MS/s)	**Normalized Area Efficiency** ($GS/s/mm^2$)	**Normalized Power Efficiency** ($GS/s/W$)	**Normalized FoM** ($GS/s/W/mm^2$)
AP	16 nm	2048	16	0.096	268	3.74	7.84	81.35
AP+SC+ML	16 nm	2048	16	0.097	268	3.68	15.66	161.58
AP+SC+ML (VoS)	16 nm	2048	16	0.097	268	3.68	25.08	258.83
[146]	65 nm	1024	16	8.29	240	0.16	129.14	116.86
[10]	45 nm	2048	32	0.97	0.22	0.002	0.25	3.99
[170]	65 nm	2048	12	1.38	20	0.06	8.97	80.79

each benchmark when the AP architecture is configured specifically for these benchmarks. According to the results, the area overhead is between 0.49% in FFT and 3.29% in image binarization with an average of 1.27%. The FFT processor requires a 121x512 cell array together with additional matching circuits in each row, so adding two AND gates and a single SRAM cell does not cause a significant area overhead. The overhead of the image binarization is the most because it requires relatively smaller CAM (i.e., 17 cells/row). On the other hand, it is explicit that the energy reduction is much more than all these numbers.

Figure of Merit

The comparison of AP-based implementation of FFT processors with the traditional sub-65 nm approaches is shown in Table 5.6. The table includes three versions of AP implementation of 2K 16-bit FFT where the first one corresponds to the original implementation, the second one exploits the SC and ML together. The third one corresponds to the case where voltage over scaling is applied to the CAM cells (i.e., core voltage is decreased 10%) and noise margin of the compare operations is narrowed as a result. The table shows the normalized area, power efficiency numbers and a figure of merit in terms of throughput over power density. For a fair comparison, the respective numbers are normalized according to the equations 5.2, 5.3, and 5.4 where N corresponds to the FFT size.

$$\underset{Area}{Normalized} = \frac{Area}{\left(\frac{Tech}{16\ nm}\right)^2 \cdot \left(\frac{Wordlength}{16}\right) \cdot \left(\frac{N \cdot log_2 N}{11 \times 2^{11}}\right)} \tag{5.2}$$

$$\underset{Power}{Normalized} = \frac{Power}{\left(\frac{Tech}{16\ nm}\right) \cdot \left(\frac{Wordlength}{16}\right) \cdot \left(\frac{V_{DD}}{0.7\ V}\right)^2 \cdot \left(\frac{N \cdot log_2 N}{11 \times 2^{11}}\right)} \tag{5.3}$$

$$\underset{Throughput}{Normalized} = \frac{Throughput}{\left(\frac{16\ nm}{Tech}\right) \cdot \left(\frac{16}{Wordlength}\right) \cdot \left(\frac{N}{2^{11}}\right)} \tag{5.4}$$

As shown in the table, AP provides tremendous gain in terms of area efficiency. The gains are shown in both the absolute area numbers as well as the normalized area efficiency as measured in $GS/s/mm^2$. In terms of normalized power efficiency measured in $GS/s/W$, the

low-power AP architecture increases the efficiency by 2x making it the second best where it had been third. Finally, in terms of the Figure of Merit (FoM) chosen as the normalized power efficiency per unit area $GS/s/W/mm^2$ where the low-power AP outperforms all other architectures whereas the normal one is not.

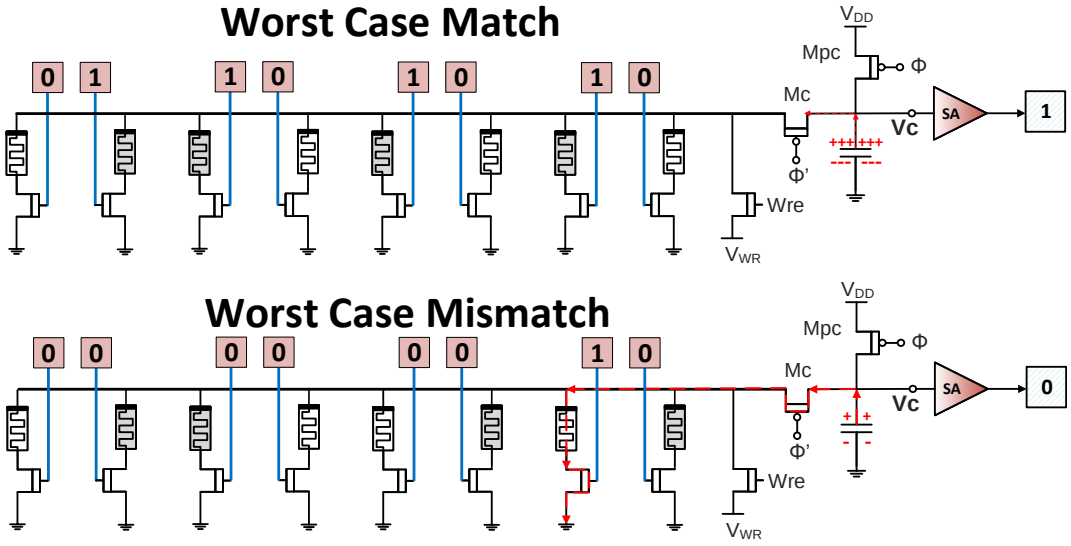
5.2 Multi-compare for RAPs

Even though high switching activity poses a threat on the power efficiency of SAPs, the main problem of RAP architectures is the high writing energy of memristors.

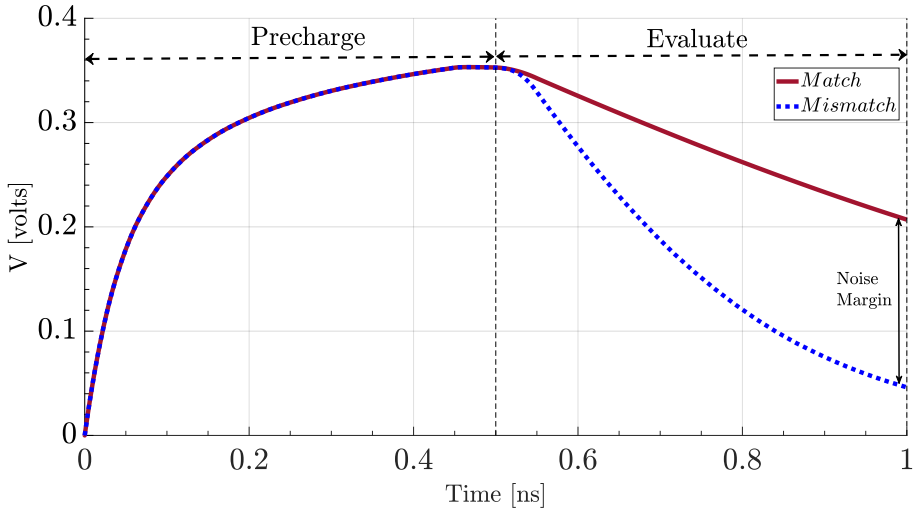
5.2.1 Motivation

In associative processing, the compare operations can be done on up to four columns (i.e., not like the traditional CAM operations where sometimes a search operation is performed on all columns). This is because the maximum number of searched columns is 4 for which the four columns are searched together in the multiplication operations. On the other hand, the minimum number of columns required in a search operation is one which is performed during the copy operations. Depending on these informations, Figure 5.10 shows the two ultimate compare cases for match and mismatch situations in an traditional AP. Figure 5.10aa shows the worst case match operation where the search operations are performed on the all four columns where some leakage occurs from the capacitor, but still sense amplifier outputs logic-1 since this leakage is not enough to fall it below V_{th}. On the other hand, Figure 5.10ab shows the worst case mismatch operations where the only one cell leaks the charge while others are in the off state and as a result, the voltage across the capacitor falls below the V_{th} and leads to a logic-0 at the output of the sense amplifier. Figure 5.10b shows the waveform corresponding to the worst match and mismatch cases during a compare operation

together with the noise margin between them. To guarantee a sufficient noise margin between these two cases, the memristors must be switched between a sufficient range of *on* and *off* values (i.e., R_{off}-R_{on}). Otherwise, the remaining charges across the the capacitor cannot be distinguishable for the worst match and the worst mismatch cases (and even for the others) unless the memristor is written properly.



(a) The worst match and mismatch cases at the AP rows



(b) The waveform and the noise margin corresponding to the worst match and mismatch cases

Figure 5.10: The cases for the worst case match/mismatch and their corresponding waveform

The main contribution to the energy consumption in RAPs is the memristor write energy. Even though a single compare cycle per row consumes an order of femto joules, switching a

cell within full range (R_{off}-R_{on}) which consists of two memristors takes order of pico joules. This consumption is mainly caused due to the huge leakage current occurred at the low resistance states of the memristor and the longer switching period when the memristance becomes closer to the border values (see Figure 4.5). The solution of this problem can be avoiding the low resistance states while switching the memristor within an acceptable margin. In this way the memristor switches between R'_{off}-R'_{on} where $R'_{on} > R_{on}$ and $R'_{off} < R_{off}$. However as stated in the previous section (Section 4), this situation causes inaccuracy in the results since some match and mismatch operations mislead to the wrong results. Table 5.7 shows the three cases for memristance scaling where the Case 0 corresponds to the full range and the others show the scaled two cases. As stated in the table, the scaling the memristor leads to a considerable decrease in the write energy. On the other hand, it deteriorates the noise margin negatively. Figure 5.11 shows the difference between the worst case matched and mismatched rows (i.e., noise margin) for the all three cases in Table 5.7.

As the summary, driving the ReRAMs within a narrower range provides a considerable energy savings as well as a performance improvement since the required write pulse becomes narrower as well. On the other hand, it forces the system to run in an inaccurate region where it is not acceptable for the exact (non-approximate) computation. As a result, if one can get benefit from the ReRAM scaling together with the acceptable noise margin, the low power RAPs can be obtained without sacrificing the reliability of the processors. The following section details the methodology for this purpose.

e-15

Table 5.7: Cases for ReRAM scaling and their corresponding energy and timing results

Case #	$\mathbf{R_{on}}$ (Ω)	$\mathbf{R_{off}}$ (Ω)	$\mathbf{T_{write}}$	$\mathbf{E_{write}}$	$\mathbf{E_{compare}}$	**NoiseMargin**
0	100	100 k	2 ns	21.7 pJ	2.92 fJ	178 mV
1	1.7 k	79 k	1 ns	349.6 fJ	2.56 fJ	119 mV
2	3.2 k	40 k	0.5 ns	121.8 fJ	2.29 fJ	36 mV

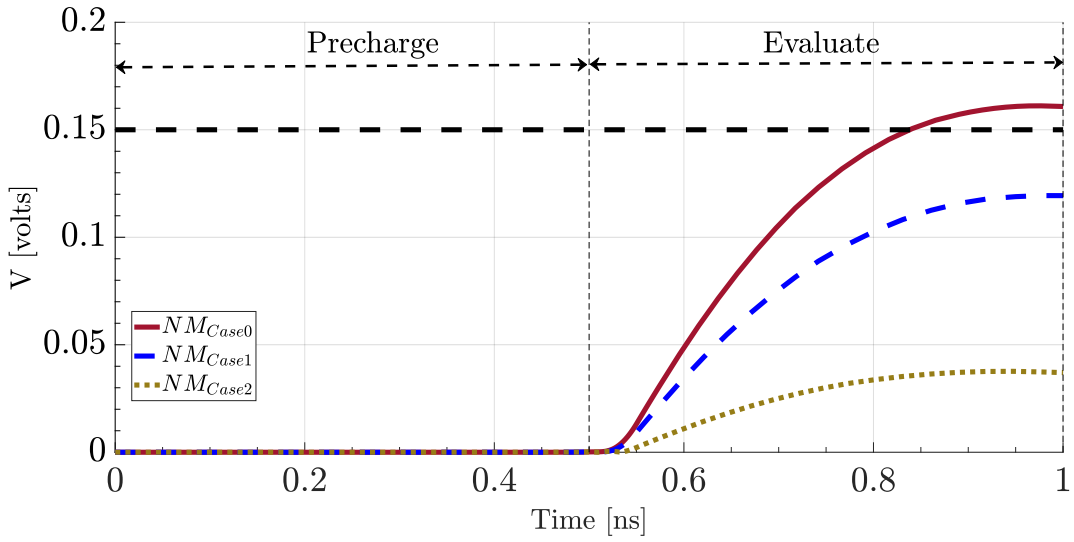


Figure 5.11: The waveform for the noise margin of the three cases during a compare cycles

5.2.2 Methodology

In order to obtain the low energy consumption proposed by memristance scaling as well as satisfactory noise margin without harming the accuracy of the computation, the proposed method of multi-compare for resistive associative processing are evaluated. In multi-compare, the single cycle compare operation (i.e., consists of one precharge and evaluate phases) is divided into multiple compare cycles by decreasing the number of compared columns (i.e., $<$ 4) during an operation. By this way, the noise margin between the worst match and mismatch cases can still satisfy the reliability requirements of the processors. As an example, a compare operation during the multiplication is separated into two consecutive compare operations when the maximum number of compared columns is set as 2. After each compare, the result of the sense amplifier is ANDed with the result of the previous cycle if they are the part of a single compare operation. Even though increasing the number of required compare cycles negatively affects the performance, the decrease in the write time as a result of scaling (see Table 5.7) can compensate the performance decrease in the compare operations by increasing the performance of the write operations.

Figure 5.12 shows the noise margins for each proposed cases and number of maximum com-

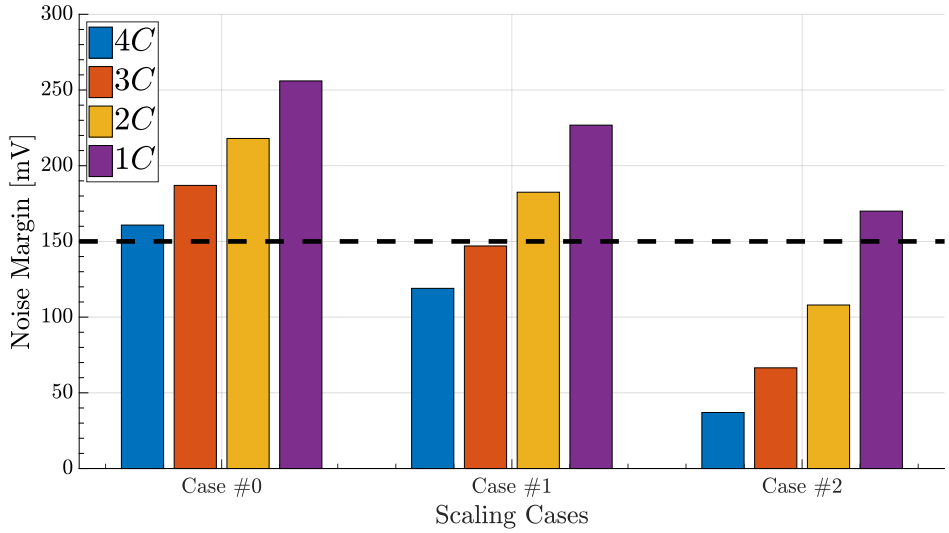


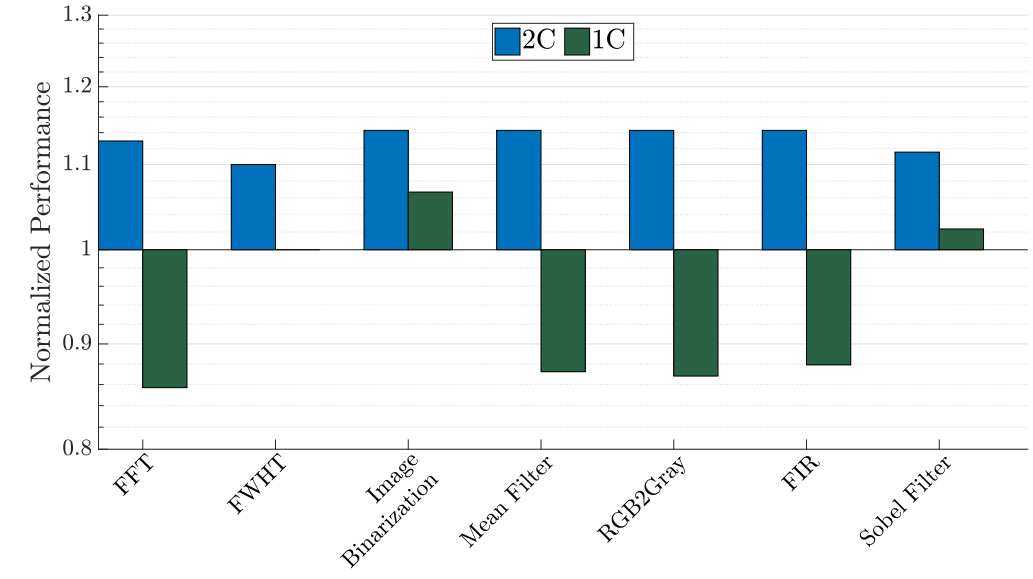
Figure 5.12: Noise margins for each case and the maximum number of compared columns

pared columns. As an acceptable threshold, we set a limit of 150 mV to ensure the satisfactory gap between the two worst cases which is about 21% of the nominal V_{dd} of 0.7v. In some cases, more than one maximum number of compared columns satisfy this requirement, however, in this case, keeping the column number as big as possible provides the maximum performance. As an example, for case #1, 1-column and 2-column compare cases satisfy the noise margin. On the other hand, setting it to 2-column compare provides better performance than 1-column compare.

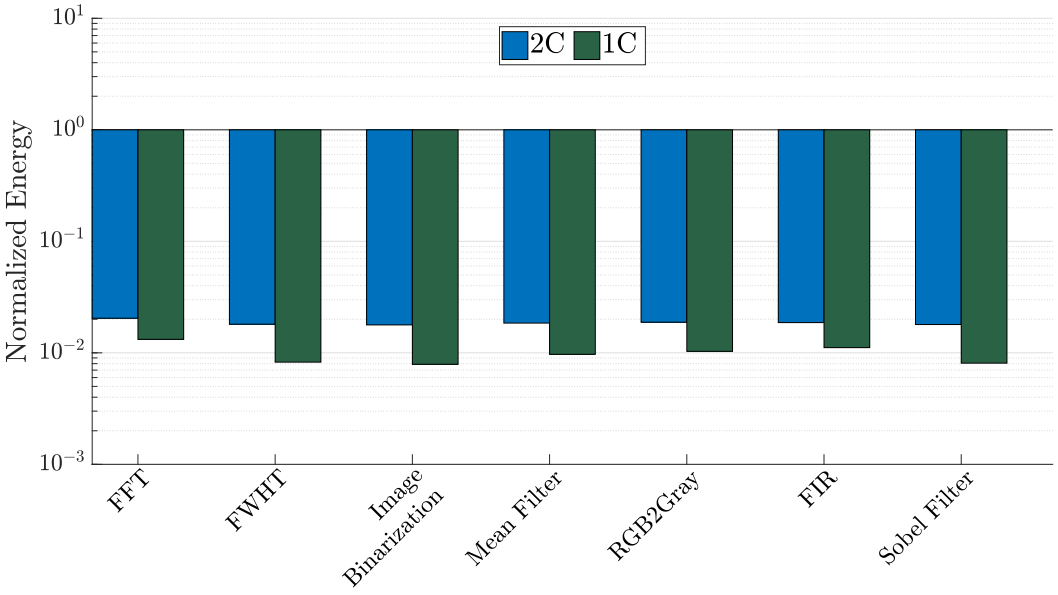
5.2.3 Evaluation

For the evaluation, a set of seven benchmarks from different domains are simulated on the simulator described in Section 2.5. Figure 5.13 shows the normalized performance and energy results which are obtained for each benchmark for the 1-column compare and 2-column compare cases. For 1-column compare, the scaling case #2 is referenced and the case #1 is referenced for 2-column case. Even though multi-compare provides a considerable energy savings for both compare cases, 1-column compare case sometimes provides performance decrease in some benchmarks. On the other hand, applying 2-colum compare to a RAP

provides a considerable speedup and energy gain.



(a) Normalized performance



(b) Normalized energy

Figure 5.13: The normalized performance and energy results of the benchmarks for 2-column and 1-column compare cases

5.3 Conclusion

In this chapter, low-power APs implementations by proposing architectural and instructional improvements are proposed. In SAPs, a selective pre-charge and evaluate mechanism are facilitated and the LUTs of multiplication and absolute value operations are modified to get more benefit from architectural changes. For RAPs, the multi-cycle compare operation are proposed to decrease the write energy consumption of ReRAMs without harming the reliability of the system.

Chapter 6

Two-dimensional AP

In this chapter, a new and novel associative processing architecture is proposed to cope with the current limitations of the traditional 1D AP architecture. The designed architecture is implemented by both SRAM-based and ReRAM-based CAMs and can be employed as an accelerator.

6.1 Introduction

These studies on the traditional AP architectures prove that APs can be a good candidate for solving the problems of current computing systems when used as an accelerator near the main processor [176, 60, 69]. In these, and other designs [118, 177], CAMs operate on 1D vectors, and CAM-based APs are therefore limited to operate within a CAM row. On the other hand, any 2D reduction operations across columns of data (e.g. such as summation, product, population count, etc) must rely on circuits external to the CAM and typically implemented in CMOS. These circuits end up being too expensive in area, power and/or performance eliminating much of the benefits of the resistive RAM. These structures

significantly increase area and/or delay and power of the composite computing system and cannot fulfill the deficiencies of the traditional architecture. In [40], these tree structures take up over 50% of the overall area and increase the delay by over 20x. Moreover, these architectures cannot parallelize all the steps needed to perform fundamental vector and matrix operations in the CAM array and need an adjuvant structure, often implemented in CMOS to perform the other operations.

6.1.1 Motivation

Even though AP facilitates efficient parallel in-memory computation, one of the shortcomings of the conventional AP architecture presented in [176] (and of most similar one in [69]) is that only row operations can be performed in parallel. This limits the scope of the architecture to 1-D vector-based operations. Additionally, any possible inter-operations between the rows requires additional data interconnection mechanisms. For example, consider a dot product

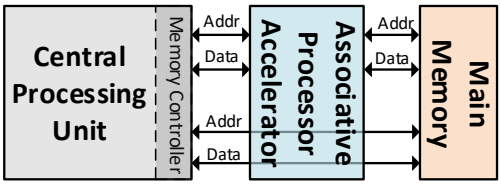


Figure 6.1: Overall system architecture with AP accelerator

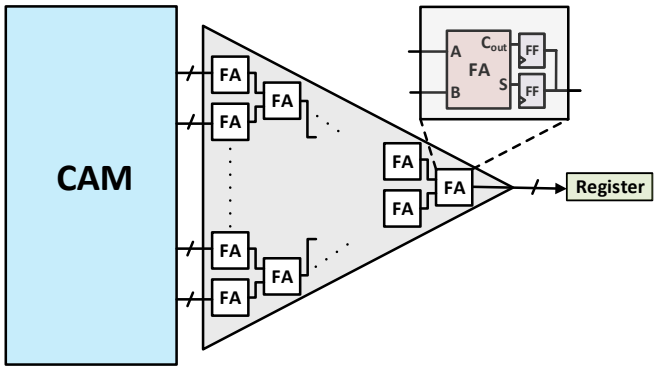


Figure 6.2: 1D AP with Adder Tree

operation:

$$A \cdot B = \sum_{i=0}^{n-1} A_i B_i = A_0 B_0 + A_1 B_1 + \cdots + A_{n-1} B_{n-1} \tag{6.1}$$

The traditional AP architecture (1D AP) can compute all the individual $A_i \cdot B_i$ in parallel in $\mathcal{O}(m^2)$, but summing the individual product terms cannot be done on the CAM because it is not capable of performing operations across different rows. Thus, one must either move the individual product terms to the main processor or juxtapose a reduction tree (or adder tree) to the 1D architecture as proposed in [176, 42, 40], allowing for a limited set of functions (e.g., tree addition of n numbers). Figure 6.2 demonstrates the AP architecture with a reduction tree (i.e., adder tree). In here, the reduction tree is used to perform the vectorial operations that produce a scalar value (e.g., the sum of products) at the end. In addition to them, this reduction tree can be used to count the number of matches or mismatches. As architecture, the adder tree consists of the stages of the full adders (FA). The first stage of adders performs the pairwise addition of two CAM rows (A and B) so that every two rows feeds to one adder. The second stage sums the partial results of the first stages and so on. After each stage, the partial sum of the results is stored in the registers (flip-flops between the FAs) so that the reduction tree is fully pipelined. At the end, the last full adder outputs the sum of the values in the CAM rows and writes the result to the output register.

Even though the reduction tree provides an additional functionality to the 1D AP, these functionality is limited to the addition of all rows. However, some applications require the local addition inside a group of CAM rows (e.g., FIR filter). Furthermore, some applications that need parallel computing capability requires vector processing in both horizontal and vertical dimensions (2D) (e.g., image filtering applications, 2D transformations) and this processing can include not only addition but also other operations such as multiplication, subtraction, etc. When the traditional 1D AP is employed for these applications, the AP

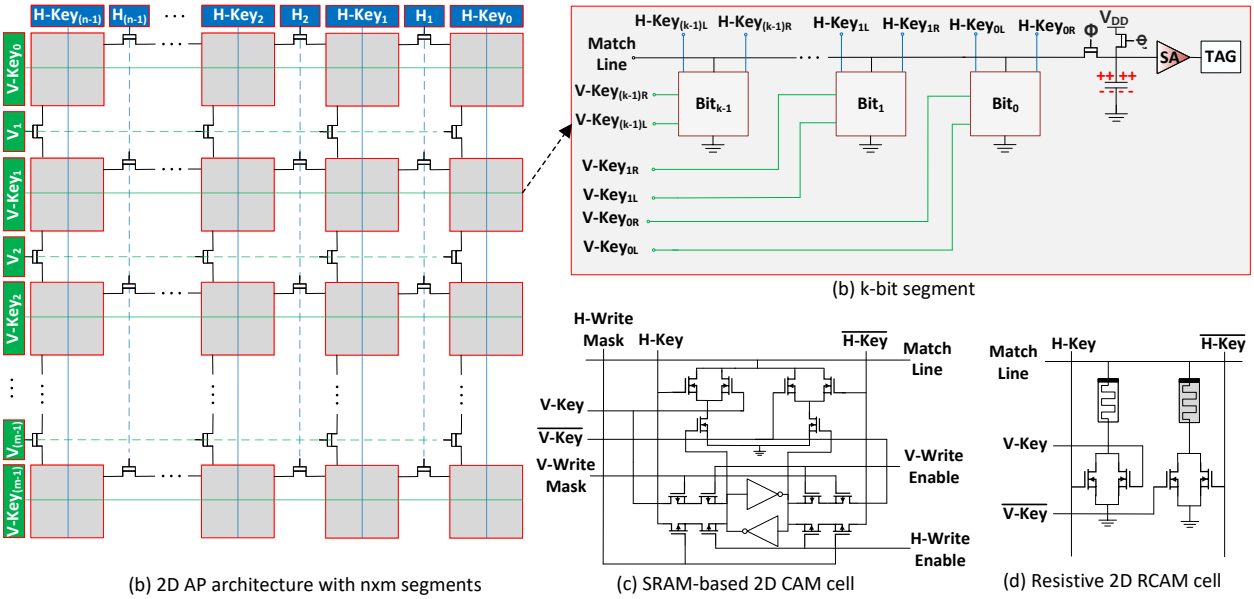


Figure 6.3: Proposed 2D associative processor (AP) architecture

itself cannot demand the requirements of the applications and requires to handle them sequentially. Therefore, this issue affects the power density, speed, and flexibility of these architectures, and limits the scope of tasks that can be efficiently executed.

6.2 Proposed Architecture (2D AP)

To compensate the aforementioned deficiencies of 1D AP, a *two-dimensional (2D) AP* architecture based on 2D CAM is proposed as illustrated in Figure 6.3. The figure shows the architecture of 2D AP hierarchically from the overall architecture (a) to the segment circuit (b), and down to the circuit implementations of CAM cells in detail (c, d). In a 1D CAM, a key is broadcast across the horizontal key (H-Key) broadcast lines (blue in Figure 6.3a) such that each row in the CAM is matched against the key. Matching rows will assert their respective match lines (solid black lines). In order to accommodate the 2D connectivity, the 1D CAM cell from Figure 2.2 is modified as shown in Figure 6.3c and Figure 6.3d. Over the 1D AP, the 2D CAM adds another set of broadcast and match lines, and another set of key

registers vertically (as shown in green). Therefore, in addition to the horizontal match operations described above, the architecture can now perform vertical match operations whereby a key from the set of vertical key registers (V-Key, in green) is matched against columns of matched lines. Thus, when a column matches the key contents, its match line (dashed blue) is asserted. In Figure 6.3a, the dashed blue lines correspond to the horizontal select lines, and the dashed green lines correspond to the vertical select lines. These select lines control the transistors that connect the match lines of the segments to each other. For example, H_1 select line combines the segments corresponding to $H - Key_0$ and $H - Key_1$ to each other and operations can be performed on these two segments.

Figure 6.3b illustrates the architecture of a k-bit segment. In the same manner done for 1D CAM, an SRAM-based and ReRAM-based CAM cells can be employed in the 2D architecture. For the 2D SRAM-based cell, the write and bit lines are duplicated to support the vertical search as well as horizontal search. The 2D SRAM cell (shown in Figure 6.3c) requires 18 transistors instead of 12 transistor correspondence for 1D SRAM cell. Only the circuit used for the writing and matching are duplicated to support the operations in the second dimension. The SRAM-cell (i.e., a coupled inverter) has no change and the cell still stores the single bit, so the static power consumption is not affected. The increase in the area can be compensated by using more area efficient SRAM structures such as 4T SRAMs [14]. Similarly, the two-transistor two-ReRAM cell circuit of 1D RAP is scaled to accommodate 2D access using four-transistors and two-ReRAM per cell in 2D RAP. In the same manner, a single bit is stored in complementary mode inside the cell. The complementary ReRAM can be "sensed" from either the horizontal or the vertical key registers. For both cell types, a mismatch from either direction will cause the cells' output to discharge, causing either the horizontal or vertical match lines to discharge, depending on which mode is enabled. For ReRAM-based 2D AP (i.e., 2D RAP), the increase in the array size does not affect the overall energy consumption since memristor is a nonvolatile element. Contrary to the SRAM-based CAM implementations which require static energy to keep the data even though the circuit is

not activated. This deficiency can be overcome by using the design in [28] where an SRAM-based CAM cell is modified as non-volatile by adding two more memristors. For any CAM cell implementations as either ReRAM, or SRAM, or any other technology, the proposed 2D architecture is independent of the cell implementation. To illustrate, if one-day ReRAM replaces the post CMOS technology, the architecture can be accommodated seamlessly.

In addition to vertical matching support, the 2D architecture also supports segmentation in order to improve parallelism. Each segment can be considered as an individual tiny AP with its own pre-charging, matching, and writing circuitries. A segment stores the smallest meaningful data in the CAM. Depending on the AP mode, either horizontal or vertical select lines are activated. It is also possible to activate a subset of them as well, thus, there can be different modes in which this AP can operate. As shown in the figure, each key broadcast either horizontally or vertically has an extra control bit (H or V). The 2D AP works as before in horizontal mode when (H, V) = (1, 0), and vertical mode when (H, V) = (0, 1). When (H, V) = (0, 0) the word is disconnected from match lines in both directions. Since each word has its own write circuitry (essentially operating as a 1D, single word AP), operations can be performed in parallel across different segments in a row or column. Using the segmented architecture, it is possible to segment a row or column into the independent groups, each performing the same or different operations across rows or columns simultaneously. The segmentation also provides energy efficient architecture since it allows those segments which are not needed to be disabled in a dynamic manner.

The choice of *segment size (k)* is a design space parameter. For a fixed size of 2D AP, a small value of k allows for more parallelism across a single row and also enables finer grain power management. Operations across wider bit widths can be accomplished by concatenating adjacent segments, therefore making the array very flexible when handling operations of varying bit widths. However, the overhead of the peripheral circuitry (such as the sense amplifier, tag, precharge transistors, and write circuits) results in an increase in power/energy

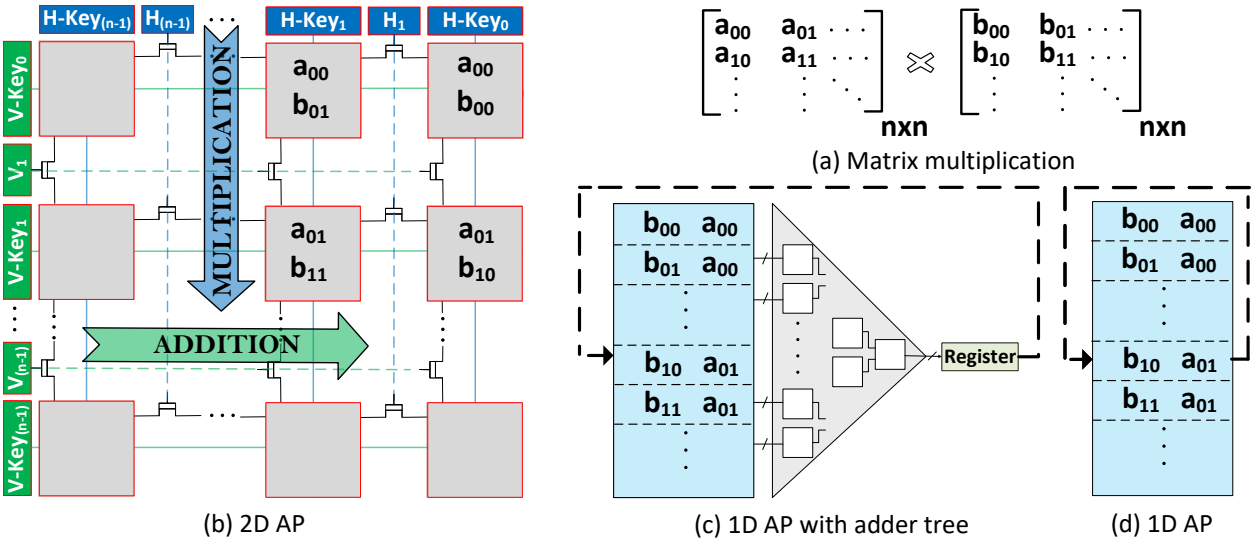


Figure 6.4: n×n matrix multiplication on 2D AP, 1D AP w/o adder tree

and area, and eventually degrade the overall area and energy efficiency (and potentially the performance) of the whole system.

Going back to the dot product example in Section 6.1.1 (Equation 6.1), the product of a vector on the 2D AP can be performed as follows:

i. First the array is set to the horizontal mode (blue lines) and the $A_i \cdot B_i$ product terms are computed in parallel and stored in a column, such as, the A column, in $\mathcal{O}(m^2)$ time.

ii. Following that, the array is switched to the V mode and the keys are now broadcast vertically, allowing the addition of the A_i and B_i product terms. Adding n numbers can be done in $\mathcal{O}(m \log n)$ time and the result can be placed in a word, for example, the top left word.

For the complexity, since m (i.e., bit width) is fixed for a given application, the time complexity of the dot product operation becomes $\mathcal{O}(\log_2 n)$ in overall.

Table 6.1: Theoretical complexity of various kernels where complexity order between the cells is green < blue < red.

Kernel	Sequential	1D RAP [40] [69]	1D RAP * [176]	2D RAP
Vector Add, Multiply	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Vector Dot Product	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(\log_2 n)$
Vector-Matrix Multiply	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log_2 n)$	$\mathcal{O}(\log_2 n)$
Matrix-Matrix Multiply	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2 \log_2 n)$	$\mathcal{O}(n \log_2 n)$
Histogram (k bins and n values)	$\mathcal{O}(kn)$	$\mathcal{O}(k)$	$\mathcal{O}(k)$	$\mathcal{O}(k)$
Frequency (k values in n values)	$\mathcal{O}(kn)$	$\mathcal{O}(k)$	$\mathcal{O}(k)$	$\mathcal{O}(k)$
Set membership	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Set intersection or union	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
1D FFT, IFFT, FWHT, Conv. (n)	$\mathcal{O}(n \log_2 n)$	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(\log_2 n)$	$\mathcal{O}(\log_4 n)$
1D Filter (n data, m filter [1])	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
2D FFT (nxn)	$\mathcal{O}(n^2 \log_2 n)$	$\mathcal{O}(n \log_2 n)$	$\mathcal{O}(n \log_2 n)$	$\mathcal{O}(n \log_4 n)$
2D Stencil (nxn, 5 point)	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$
2D Filter (nxn data, mxm filter [1])	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$

* includes an adder tree.
[1] m is a constant.

6.3 Evaluation

Capability of processing the data on the second dimension together with the segmentation feature provides performance improvement theoretically by decreasing the operational complexities. For the evaluation of 2D AP, the theoretical performance improvements due to two-dimensional architecture are investigated in terms of big-oh notation. As shown in Table 6.1, a wide range of applications and kernels is considered to evaluate the performance of the 2D AP over a single 1D AP (with and without an adder tree) when they are used as an accelerator and a sequential processor. It is assumed that the data is stored in the accelerator since it is an in-memory accelerator. It is important to note that the table summarizes the theoretical bounds on time complexity of various kernels where bit width (m) is constant, and the actual performance may depend on the specific capabilities of the architectures, IO, data size, and etc. Note that in some cases, the 2D architecture offers the same complexity as 1D, while in others, a significant speedup can be attained. For example in FFT, 2D

architecture facilitates the radix-4 FFT instead of the radix-2 of 1D, and this can provide considerable improvement in time complexity. For 2D filters, 2D AP can provide concurrent computation on the data if data is ready inside the CAM array (i.e., that is what in-memory accelerator does). Additionally, in 2D AP, no additional hardware is needed to perform population count or reduction, compared to [176, 42] and [40]. Similarly, the speedup in 2D Stencil comes from the segmented computing capability performed on 2D data as parallel and data movement savings which requires additional interconnections in 1D AP. Furthermore, even though 1D AP with addition tree facilitates these features, 2D AP provides more functionality apart from the addition and population count such as multiplication, subtraction, logical operations, and etc. which are very useful especially in filters. In the 2D AP, it is also possible to perform multiple dot product operations in parallel as well, such as the case in matrix multiplication that brings a considerable speedup. While the adder tree juxtaposition improves computational complexity only for vector dot product, vector-matrix, and matrix-matrix multiplication kernels (compared to 1D), the 2D architecture provides a significant advantage in computational complexity over the 1D with adder tree for both Vector-Matrix and Matrix-Matrix multiplication kernels and similar complexity for Vector dot product.

Figure 6.4 shows an example case of n×n matrix multiplication (Figure 6.4a) on 2D AP, 1D AP with and without adder tree. In the matrix multiplication, one row of the matrix A is multiplied by the matrix B at a time. In this case, all implementations have a complexity term of $\mathcal{O}(n)$ since this computation has to be done. Figure 6.4b shows the operation on 2D AP. In 2D AP, the multiplication operation is completed in the horizontal mode within a constant time. After that, the processor switches to the vertical mode and the summation operation (reduction) are performed for each column in a constant time. This reduction takes $\mathcal{O}(\log_2 n)$ complexity, so total complexity becomes $\mathcal{O}(n \log_2 n)$. For 1D AP with adder tree (Figure 6.4b) The same data is placed as horizontally to the CAM array and the multiplication is performed in constant time. After the multiplication, the summation

operation is performed within each group to form a coefficient in the result matrix. Even though adder tree supports $\mathcal{O}(\log_2 n)$ complexity in the addition as like in 2D AP vertical mode, in here, the reduction operation is performed n times. In total, the complexity for the 1D AP for adder tree becomes $\mathcal{O}(n^2 \log_2 n)$. In 1D AP without adder tree, the reduction operation is also performed in linear time, so complexity becomes $\mathcal{O}(n^3)$.

6.4 Experimentation

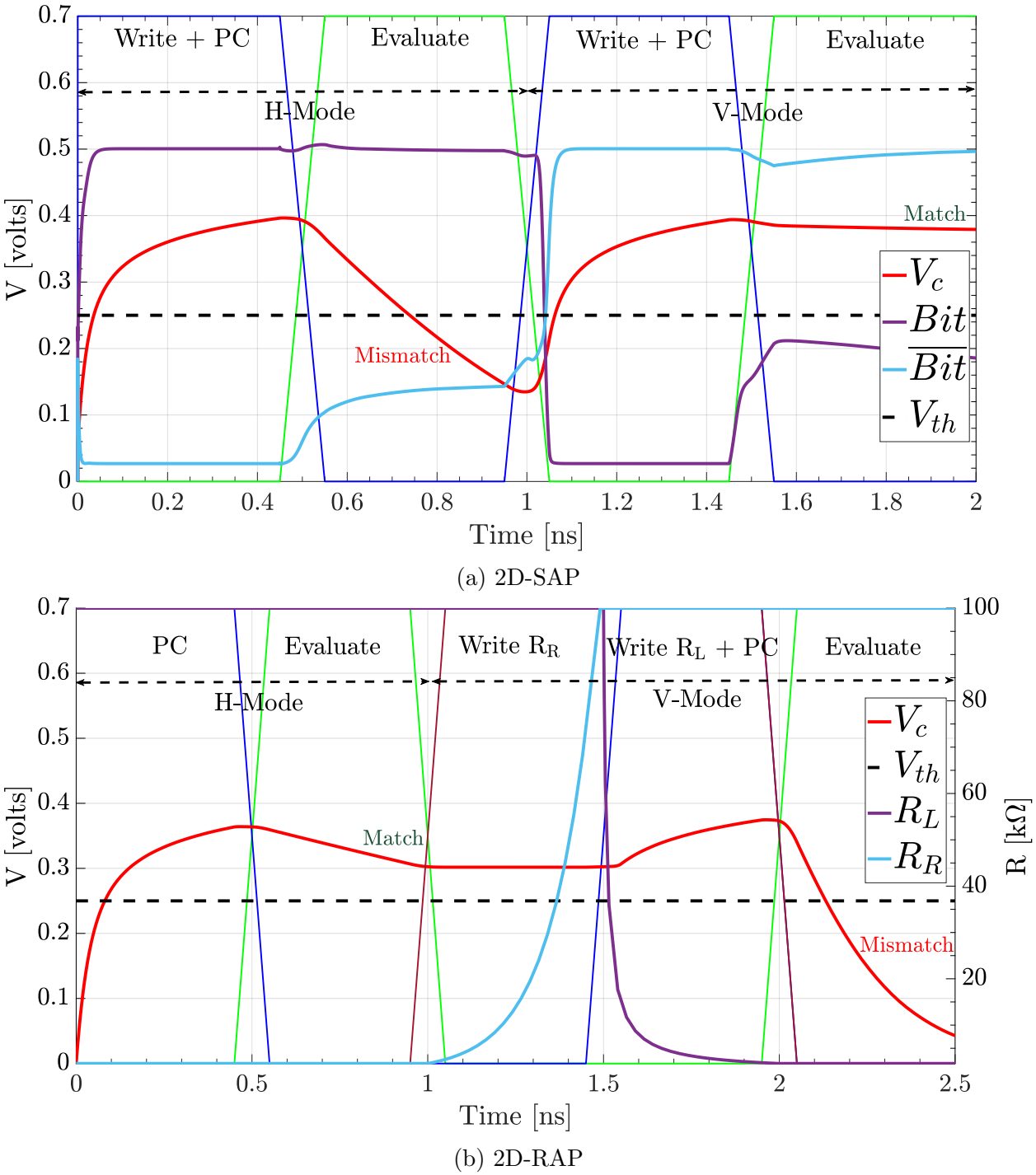
6.4.1 Simulation Framework

In order to assess the efficiency of 2D AP implementation experimentally rather than theoretical bounds, we utilize a SPICE-based in-house simulator from Section 2.5. For the transistors, Predictive Technology Models (PTM) from [159] are used to simulate high-density CAM arrays with sub 20nm feature sizes [151]. For the memristor element, the nanosecond switching time device model presented in [114] and its corresponding SPICE model in [168] are adopted since it exhibits the fastest memristor device between many fabricated memristor models [107] [121]. For the sense amplifier, a low-power, sub-ns amplifier design in [145] is employed in the circuit.

In the associative processing, the number of the searched columns are limited to the number of compare bits in the LUTs and this number is maximum four for the multiplication and minimum one for the copy operations [32]. For this reason, we optimize each architecture accordingly (e.g., setting the V_{th}) and make them resistive against the variations in the voltage sources [79] and memristor elements [88].

Figure 6.5 shows voltage changes during two compare cycles of 2D AP architectures in H and V modes consecutively and proves the functional correctness of the architecture where

the figures 6.5a and 6.5b corresponds to 2D SAP and 2D RAP correspondingly. During the operation, first H mode is selected by asserting the H line as logic-1 and the V line as logic-0



(a) 2D-SAP

(b) 2D-RAP

Figure 6.5: Spice simulation of two consecutive write and compare cycles in H and V mode respectively in 2D AP

and then the compare operations are performed as row-wise. After that, the configuration is switched to V-mode. Despite the previously published AP architectures, we manage to perform the precharge and write steps simultaneously to get a benefit of throughput since these two operations are independent of each other as inferred from the Figure 2.3, Figure 2.5, and Figure 6.3b, so they can be intertwined. Figure 6.5a shows the waveform of this architecture where write phase is combined with the precharge cycle in the 2D SAP. This situation also provides energy reduction in 2D SAP (and in 1D SAP if applied) since the energy consumption due to the static power decreases as the total time decreases. In 2D RAP, the precharge step can be performed during the last write cycle since single cell write operations consists of two consecutive write cycles, but still provides a considerable speedup overall. Lastly, the compare operations are performed as column-wise in the V-mode.

Table 6.2: Average energy and power results of the 2D AP (ReRAM & SRAM) where each segment is 32-bit

	2D ReRAM		**2D SRAM**	
Operation	**Energy**	**Power**	**Energy**	**Power**
Compare (per row)	4.908 fJ	4.8908 µW	5.425 fJ	5.425 µW
Write (per cell)	347.208 fJ	347.208 µW	0.242 fJ	0.484 µW

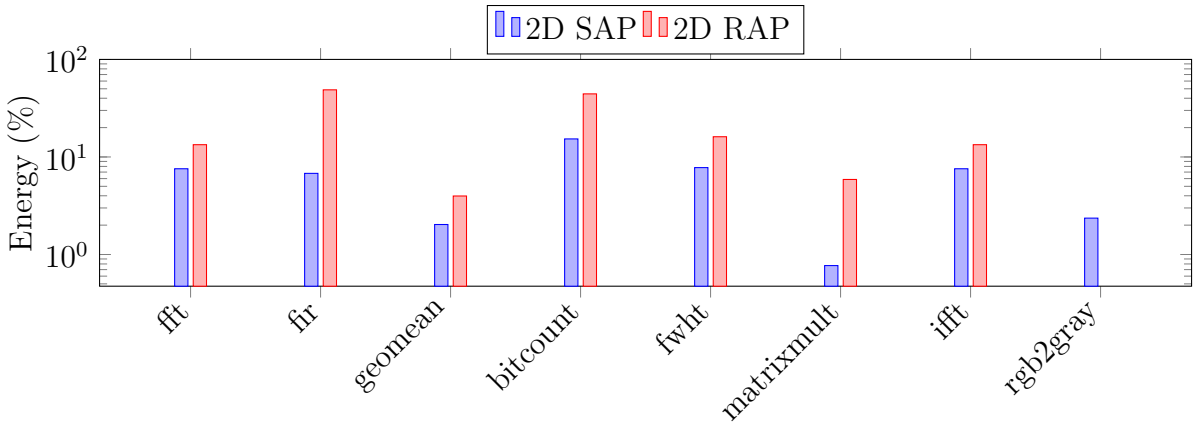
Table 6.2 shows the average energy consumption and power dissipation results of 2D AP for SRAM and ReRAM based variants for 32-bit segment size. In the table, all operations except write in 2D RAP take 0.5 ns. In RAP, write operations takes 1 ns as the combination of two write cycles each is 0.5 ns. Even though, SRAM-based segment outperforms the ReRAM-based segment overwhelmingly in terms of power and energy consumptions, ReRAM-based cell provides excellent scaling in terms of area (i.e., 4T2M vs. 18T). It is also hopeful with the advent of new ultra-low power and high-speed memristors (e.g., a possible candidate in [18]) and low power ReRAM-based CAM techniques [58], [60], [173], this technology can be viable as a low power alternative.

Table 6.3: The evaluated benchmarks, their features, and the provided input

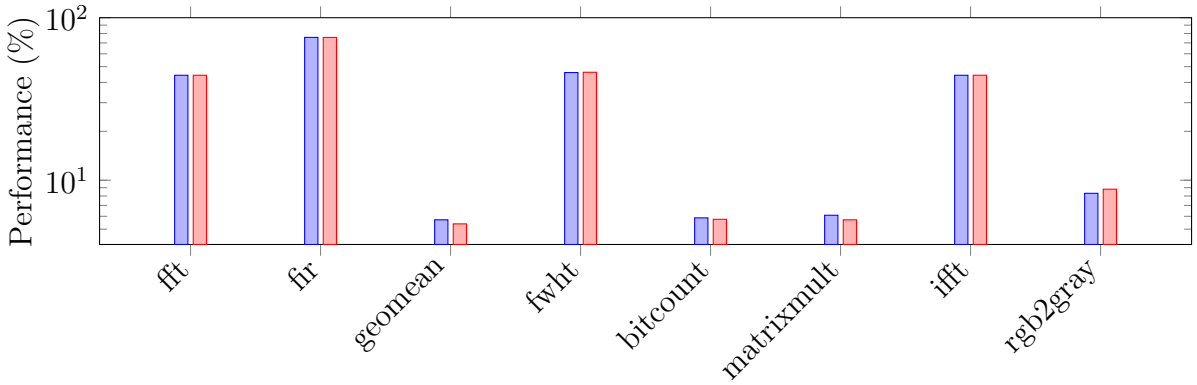
Benchmark	Domain	Description	Input
FIR	Signal Processing	8-tap filter	1K 8-bit data
FFT	Signal Processing	12-bit	1K 12-bit complex number
FastWalsh	Signal Processing	18-bit	256x256 gray image
IFFT	Signal Processing	12-bit	1K 12-bit complex number
RGB2Gray	Image Processing	8-bit	256x256 color image

6.4.2 Energy & Performance

For the evaluation of 2D AP, a set of benchmarks from different domains [179], [44] are run on the simulation framework. During the evaluation, we compare the proposed 2D architecture with its 1D correspondence. We assume that the both APs are used as an accelerator, the outer processor has the equal fixed bandwidth, and the capacity of the architectures are equal. The duty of the processor is just initializing the task and getting the results back, therefore the processor does not interfere during the operation inside the APs. In porting the applications, we used the segmentation feature of the 2D AP for FFT and inverse FFT where multiplications of the complex twiddle factors and inputs (i.e., butterfly operation) are performed separately. Fast Walsh Hadamard transform (FWHT) has a similar computation flow to FFT, but less computational complexity. The segmentation feature enables parallel execution for this benchmark as well. FIR operation is a kind of vector-matrix multiplication where the multiplications with the filter coefficients are performed in the horizontal direction and the summation in the vertical direction. In this benchmark, operation on the second dimension enables the computing without moving the data or need for an adder tree. For Energy and performance improvements for both SRAM-based and ReRAM-based 2D AP over 1D correspondences are presented in Figure 6.6. The energy improvement mainly comes from the data locality advantage of 2D AP together with the static power reduction because of the faster computation. The speedup improvement is mainly due to the faster computation that 2D AP facilitates as showed in Table 6.1.



(a) Energy savings



(b) Performance improvement

Figure 6.6: 2D SAP and 2D RAP performance improvement and energy savings vs. 1D SAP and 1D RAP, respectively

Figure 6.6 shows the improvement in both speedup (performance improvement) and energy (energy saving) as the percentage when we used an SRAM-based AP and ReRAM-based AP for both 1D and 2D architectures. As Figure shows, the 2D architecture provides a performance improvement between 8% and 75% for all benchmarks. This performance improvement comes from parallel execution because of the segmented architecture. As an example for this case, RGB2Gray benchmark (i.e., RGB image to gray image conversion) requires a computation in which all three-pixel values are multiplied with different coefficients, and then summed to form the gray correspondence [73]. In 2D AP, these multiplication operations can be done as parallel inside a row due to the segmentation. After, the summation operation can be performed by horizontally combining the segments. In a similar manner,

other benchmarks get benefit from both parallelism and data locality features of 2D AP.

On the other hand, even though there is an energy improvement, it is limited up to 7% in the 2D SAP. This is mainly due to that in SRAM-based AP, write energy is comparable with compare energy and two-dimensionality provides mainly data movement savings and static energy savings because of the performance improvement. On the other hand, for ReRAM-based AP, the energy performance is more than SRAM-based one ranging up to 48% since write energy in memristor is higher than SRAM. It gives the almost same performance improvement as SRAM correspondence. In here it is worth to point that even though these benchmarks include some reduction operation through their running flow (e.g., in FIR, the summation of products, in RGB2Gray, the summation of coefficient-pixel products), these operations are local reductions where only a subset of the rows needs to be accumulated. For this reason, the 1D architecture without adder tree is referenced since it outperforms. For the simpler benchmarks such as vector-matrix or matrix-matrix multiplies, using the 1D architecture with adder three would be the more reasonable.

From the observed behavior and the results obtained in Figure 6.6 and stated operational complexities, one can infer that in order to obtain a benefit from the 2D architecture, a benchmark has a SIMD like computation pattern together with some degree of parallelism inside a row. If there is a local reduction operation as in FIR, 2D AP can perform the reduction operations in the second dimension without moving the data. On the other hand, the reduction tree requires an extra movement in 1D AP. In addition, it can support only one reduction at a time even though 2D AP can support multiple because of the segmentation. As a result, 1D AP with adder tree causes a decrease in the performance and increase in the energy consumption (see Figure 6.4. Due to both segmentation and horizontal processing features of 2D AP, the architecture provides an inherent advantage for the 2D signal processing algorithms as well.

The key point in application mapping to the 2D AP is the effect of the architectural pa-

rameters of k, n, and m. The performance of an application is tightly dependent on these parameters. For a given area budget of N-bit CAM, increasing k (i.e., the bit size of the segments) decreases the degree of parallelism since it decreases n. If k is decreased, the AP can support more parallel operations leads to increase in performance with the expense of an increase in the power density. On the other hand, k must be big enough to perform the unit operations inside a segment. Otherwise, two horizontal segments must be connected to each other to create enough place for a computation which in turn causes an indirect decrease on n, leading to decrease in horizontal parallelism. To illustrate, we can inspect the effect of these parameters on some benchmarks. For example, if we set n=1 for an FIR benchmark, we cannot get the benefit from parallelism that the 2D AP provides. Similarly, setting it to more than the order of the filters is not necessary and does not provide any speedup since maximum parallelism can be obtained by setting it to the number of taps. If the application requires vector parallelism, the parameter m becomes important as well. For instance, a radix-2 FFT requires parallelism to apply the butterfly on all word-pairs as well as the parallelism within a butterfly computation. If it is set to less than the amount of data or maximum degree of parallelism, the overall execution must be separated in chunks. As an example case, if it is set as 512 for a FFT on 1K data pairs, a single butterfly stage on 512 data pairs must be run two times sequentially.

In addition to those, there are some limitations posed by the circuit parameters. For example, if k is too wide, the noise margin after a compare operation can be very small and cause the inaccuracies in the operation. According to the our simulations on 2D-RAPs, the noise margin between the worst case match (i.e., all 4 cells are mismatched during the multiplication) and the worst case mismatch (i.e., only one cell is mismatched) drops below than the 5% of the precharge voltage (V_{pc}) after the value of k=2048. The 2D SAPs show better noise margin since a transistor in the off state exhibits much higher resistance than the ReRAM. If such a limitation exists, a compare operation can be performed separately in two or more segments and corresponding tags can be ANDed as mentioned in [42].

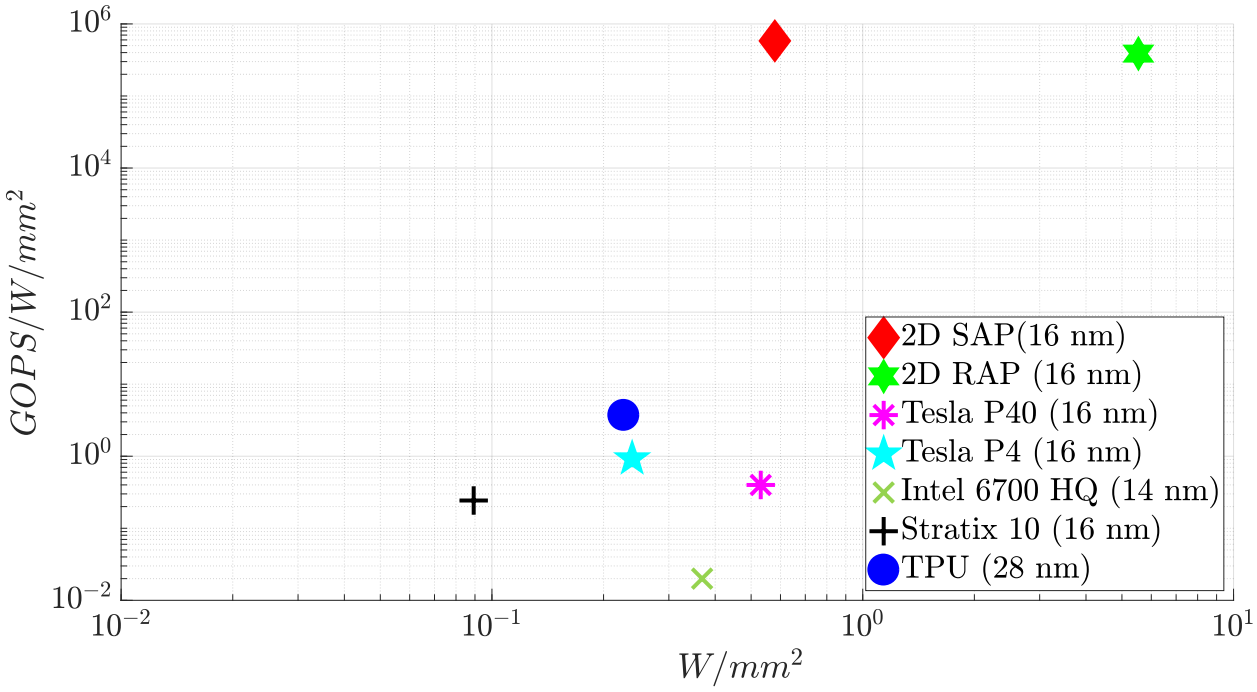


Figure 6.7: Comparing FOMs for different architectures

6.4.3 Figure of Merit

Figure 6.7 shows the $GOPS/W/mm^2$ and W/mm^2 Figures of Merit (FOM) extracted from different accelerator realizations; Google TPU [76] (28 nm), Intel Altera Stratix 10 (16 nm), NVidia P40 and P4 (16 nm) and a state of the art high-performance Intel processor [63] (from Sandra arithmetic benchmarks [152]). We contrast that with the projected FOM for the proposed architecture. The high density of the proposed architecture coupled with the simple design and elimination of data movements results in good $GOPS/W/mm^2$ that is 1-2 orders of magnitude higher than existing approaches. While high $GOPS/W/mm^2$ is desirable, the power density (W/mm^2) in ReRAM-based 2D AP is is over 10x more than some existing architectures which is a challenge that must be addressed in the future research. This can be addressed in a variety of ways. Increasing area is an obvious choice but has implications for performance and energy. As another choice without sacrificing area, building memristive devices with low write energy is an active research problem [45]. System and algorithmic approaches also have a good potential for reducing the overall system power

density. Over the past two decades, the research community has developed a substantial repertoire of power-optimization techniques, many of which can be transplanted to these architectures, eventually leading to power densities allowing this architecture to operate in a reliable manner. On the other hand, SRAM-based 2D AP realization provides an acceptable power density together with the best $GOPS/W/mm^2$.

At 16 nm technology, the ternary SRAM-based CAM allows very small area ($0.1126\,\mu m^2$/bit in [83]) and since all rows behave as a different processor, AP processor proves excellent parallel operations per power per area ($GOPS/W/mm^2$). On the other hand, this dense area causes a power density problem.

6.5 Conclusion

In this section, the novel idea of 2D AP is introduced and implemented by both SRAM and ReRAM CAM cells. The proposed architecture effectively eliminates the need for data movements, saves energy, and provides flexibility for a variety of benchmarks. The efficiency of the proposed architecture is proven by comparing it with the state of the art processors in terms of performance, energy, power, and area.

Chapter 7

Conclusion & Future Work

In this dissertation, we explore efficient in-memory computation architecture through the associative processing for data intensive applications. The perspective of the design space, system architecture, software and micro-architecture are inspected. The primary contributions of this study can be summarized as follows:

- **Architecture and Trade-offs of Associative Processors:** We described the understanding of the associative processing and processors in Chapter 2 in detail. The chapter can be handled as a fundamental source for the researchers who wants to conduct research on APs. The chapter also includes the newly defined operations and system-wide configuration schemes which has not been existing in the current literature. The proposed system architectures allow reprogrammability in memory-based computation and they are uniquely suited for vector based operations, while fully benefiting from the extreme parallelism. The architectural innovations are proposed that reduce or eliminate the need for any supporting logic, thus addressing the two main barriers to adoption and making AP based on CAMs an excellent candidate for the development of in-memory accelerators. The statements are supported by the trade-off

analysis in terms of energy, performance and reliability.

- **Approximate In-memory Computing:** Approximate computing for the associative processors are brought in the literature for the first time. The proposed approximate computing methodologies include the bit trimming, the CAM-cell scaling (either ReRAM or SRAM), and combination of them. Furthermore, it is proven that these methodologies naturally supported by AP architectures as tunable and dynamic so that during the execution degree of approximation can be tuned with respect to the performed task. A design flow to optimize the approximate in-memory computing is proposed to obtain the highest efficiency with the least quality degradation.

- **Low-power Associative Processor:** A low-power SRAM-based AP implementation is suggested by proposing novel architectural improvements to decrease the switching activity. Furthermore, some traditional operations are modified to allow better energy efficiency. For ReRAM-based APs, a considerable energy reduction is provided by multi-compare architectures where ReRAM switching range is scaled without sacrificing the reliability constraints.

- **Software Framework for Associative Processors:** We developed a cycle accurate simulator for Associative Processors together with the broad range of benchmarks from domains of machine learning, image processing, statistics, etc. The simulator is highly configurable with more than 50 parameters, supports for ReRAM-based and SRAM-based architectures. It can facilitate circuit-level simulation as well as system-level simulations. The simulator works as fully automated in which the cooperation between the system-level simulator (Matlab) and circuit-level simulator (HSpice) is coordinated seamlessly.

- **Two-dimensional Associative Processor:** A novel two-dimensional AP architecture are proposed to solve the deficiencies of the traditional in-memory processor architectures. The proposed architecture provide new improvements over the existing one

such as flexibility and sequential execution. For this reason, a novel two-dimensional in-memory computing architecture is proposed.

For the future work, we have been planning to extend our research on the following topics:

- **Acceleration of deep learning applications:** Deep learning applications performs well on parallelized processor architectures such as GPUs. From this perspective, an efficient AP implementations can be projected. For this purpose, we aim to port some CNN architectures (such as AlexNet [94], VGG-16 [150]) onto APs to perform in-memory deep learning. Furthermore, since deep learning applications can be classified as error resillient, the approximate computing methodologies can be applied in this domain.

- **Open-source simulator:** To attract more research to this area, the simulator will be released as an open-source software framework. Additionally, we are planning to extend the capability of the current simulator by adding the feature of machine learning based predictor to obtain faster profile information for the energy consumption. For this purpose, a simple model for compare and write operations can be generated by training the outcomes of the circuit simulations.

Bibliography

[1] A. Afifi, A. Ayatollahi, and F. Raissi. Implementation of biologically plausible spiking neural network models on the memristor crossbar-based cmos/nano circuits. In *2009 European Conference on Circuit Theory and Design*, pages 563–566, Aug 2009.

[2] N. Ahmed, T. Natarajan, and K. Rao. Discrete cosine transform. *Computers, IEEE Transactions on*, C-23(1):90–93, Jan 1974.

[3] A. Akerib and R. Adar. Associative approach to real time color, motion and stereo vision. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 3291–3294 vol.5, May 1995.

[4] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, A. Driskill-Smith, and M. Krounbi. Spin-transfer torque magnetic random access memory (stt-mram). *J. Emerg. Technol. Comput. Syst.*, 9(2):13:1–13:35, May 2013.

[5] A. Ascoli, F. Corinto, V. Senger, and R. Tetzlaff. Memristor model comparison. *IEEE Circuits and Systems Magazine*, 13(2):89–105, Secondquarter 2013.

[6] S. Balam and D. Schonfeld. Associative processors for video coding applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(2):241–250, Feb 2006.

[7] B. Barney et al. Introduction to parallel computing.

[8] D. Batas and H. Fiedler. A memristor spice implementation and a new approach for magnetic flux-controlled memristor modeling. *IEEE Transactions on Nanotechnology*, 10(2):250–255, March 2011.

[9] K. E. Batcher. Staran parallel processor system hardware. In *Proceedings of the May 6-10, 1974, National Computer Conference and Exposition*, AFIPS '74, pages 405–410, New York, NY, USA, 1974. ACM.

[10] A. S. Beulet Paul, S. Raju, and R. Janakiraman. Low power reconfigurable fp-fft core with an array of folded da butterflies. *EURASIP Journal on Advances in Signal Processing*, 2014(1):144, Sep 2014.

[11] M. Bhujade. *Parallel Computing*. New Age International (P) Limited, 1995.

[12] Z. Biolek, D. Biolek, and V. Biolkov. Spice model of memristor with nonlinear dopant drift. *Radioengineering*, pages 210–214.

[13] S. Borkar. Exascale computing – a fact or a fiction? In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IPDPS '13, pages 3–, Washington, DC, USA, 2013. IEEE Computer Society.

[14] R. Boumchedda, J. P. Noel, B. Giraud, K. C. Akyel, M. Brocard, D. Turgis, and E. Beigne. High-density 4t sram bitcell in 14-nm 3-d coolcube technology exploiting assist techniques. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(8):2296–2306, Aug 2017.

[15] J. Chang, Y. H. Chen, W. M. Chan, S. P. Singh, H. Cheng, H. Fujiwara, J. Y. Lin, K. C. Lin, J. Hung, R. Lee, H. J. Liao, J. J. Liaw, Q. Li, C. Y. Lin, M. C. Chiang, and S. Y. Wu. 12.1 a 7nm 256mb sram in high-k metal-gate finfet technology with write-assist circuitry for low-vmin applications. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 206–207, Feb 2017.

[16] M. F. Chang, C.-H. Chuang, M.-P. Chen, L.-F. Chen, H. Yamauchi, P. F. Chiu, and S. S. Sheu. Endurance-aware circuit designs of nonvolatile logic and nonvolatile sram using resistive memory (memristor) device. In *17th Asia and South Pacific Design Automation Conference*, pages 329–334, Jan 2012.

[17] T. Chang, S.-H. Jo, and W. Lu. Short-term memory to long-term memory transition in a nanoscale memristor. *ACS Nano*, 5(9):7669–7676, 2011. PMID: 21861506.

[18] B. J. Choi, A. C. Torrezan, J. P. Strachan, P. G. Kotula, A. J. Lohn, M. J. Marinella, Z. Li, R. S. Williams, and J. J. Yang. High-speed and low-energy nitride memristors. *Advanced Functional Materials*, 26(29):5290–5296, 2016.

[19] L. Chua. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, Sep 1971.

[20] L. O. Chua and S. M. Kang. Memristive devices and systems. *Proceedings of the IEEE*, 64(2):209–223, Feb 1976.

[21] J. Cong and B. Xiao. mrfpga: A novel fpga architecture with memristor-based reconfiguration. In *2011 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 1–8, June 2011.

[22] F. Corinto and A. Ascoli. A boundary condition-based approach to the modeling of memristor nanostructures. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(11):2713–2726, Nov 2012.

[23] F. Corinto and A. Ascoli. Memristive diode bridge with lcr filter. *Electronics Letters*, 48(14):824–825, July 2012.

[24] F. Corinto, A. Ascoli, and M. Gilli. Nonlinear dynamics of memristor oscillators. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(6):1323–1336, June 2011.

[25] Y. V. P. D. Biolek, M. Di Ventra. Reliable spice simulations of memristors, memcapacitors and meminductors. *Radioengineering*, 22(4):945–968, Dec. 2013.

[26] I. E. Ebong and P. Mazumder. Self-controlled writing and erasing in a memristor crossbar memory. *IEEE Transactions on Nanotechnology*, 10(6):1454–1463, Nov 2011.

[27] A. H. Edwards, H. J. Barnaby, K. A. Campbell, M. N. Kozicki, W. Liu, and M. J. Marinella. Reconfigurable memristive device technologies. *Proceedings of the IEEE*, 103(7):1004–1033, July 2015.

[28] K. Eshraghian, K. R. Cho, O. Kavehei, S. K. Kang, D. Abbott, and S. M. S. Kang. Memristor mos content addressable memory (mcam): Hybrid architecture for future high performance search engines. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(8):1407–1417, Aug 2011.

[29] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 449–460, Washington, DC, USA, 2012. IEEE Computer Society.

[30] U. Farooq, Z. Marrakchi, and H. Mehrez. *FPGA Architectures: An Overview*, pages 7–48. Springer New York, New York, NY, 2012.

[31] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, Sept 1972.

[32] C. C. Foster. *Content Addressable Parallel Processors*. John Wiley & Sons, Inc., New York, NY, USA, 1976.

[33] Fujitsu Semiconductor Limited. 4m (512 k 8) bit spi memory reram datasheet, 12 2016.

[34] E. Gale, B. de Lacy Costello, and A. Adamatzky. Boolean logic gates from A single memristor via low-level sequential logic. *CoRR*, abs/1402.4046, 2014.

[35] L. Gao, F. Alibart, and D. B. Strukov. Programmable cmos/memristor threshold logic. *IEEE Transactions on Nanotechnology*, 12(2):115–119, March 2013.

[36] D. Garbin, E. Vianello, Q. Rafhay, M. Azzaz, P. Candelier, B. DeSalvo, G. Ghibaudo, and L. Perniola. Resistive memory variability: A simplified trap-assisted tunneling model. *Solid-State Electronics*, 115, Part B:126 – 132, 2016. Selected papers from the EUROSOI-ULIS conference.

[37] N. Gergel-Hackett, J. L. Tedesco, and C. A. Richter. Memristors with flexible electronic applications. *Proceedings of the IEEE*, 100(6):1971–1978, June 2012.

[38] GSI Technology. In-place associative computing, 2017.

[39] W. Guan, M. Liu, S. Long, Q. Liu, and W. Wang. On the resistive switching mechanisms of cu/zro2:cu/pt. *Applied Physics Letters*, 93(22), 2008.

[40] Q. Guo, X. Guo, Y. Bai, and E. İpek. A resistive tcam accelerator for data-intensive computing. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 339–350, New York, NY, USA, 2011. ACM.

[41] Q. Guo, X. Guo, Y. Bai, R. Patel, E. Ipek, and E. G. Friedman. Resistive ternary content addressable memory systems for data-intensive computing. *IEEE Micro*, 35(5):62–71, Sept 2015.

[42] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman. Ac-dimm: Associative computing with stt-mram. *SIGARCH Comput. Archit. News*, 41(3):189–200, June 2013.

[43] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman. Ac-dimm: Associative computing with stt-mram. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 189–200, New York, NY, USA, 2013. ACM.

[44] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, pages 3–14, Dec 2001.

[45] Y. Halawani, B. Mohammad, D. Homouz, M. Al-Qutayri, and H. Saleh. Modeling and optimization of memristor and stt-ram-based memory for low-power applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(3):1003–1014, March 2016.

[46] S. Hamdioui, L. Xie, H. A. D. Nguyen, M. Taouil, K. Bertels, H. Corporaal, H. Jiao, F. Catthoor, D. Wouters, L. Eike, and J. van Lunteren. Memristor based computation-in-memory architecture for data-intensive applications. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1718–1725, March 2015.

[47] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[48] Y. Ho, G. M. Huang, and P. Li. Dynamical properties and design analysis for non-volatile memristor memories. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(4):724–736, April 2011.

[49] HP. Beyond dram and flash, August 2014.

[50] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, Sept 1952.

[51] M. Imani, S. Gupta, and T. Rosing. Ultra-efficient processing in-memory for data intensive applications. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, pages 6:1–6:6, New York, NY, USA, 2017. ACM.

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

[53] M. Imani, P. Mercati, and T. Rosing. Remam: Low energy resistive multi-stage associative memory for energy efficient computing. In *2016 17th International Symposium on Quality Electronic Design (ISQED)*, pages 101–106, March 2016.

[54] M. Imani, S. Patil, and T. Rosing. Approximate computing using multiple-access single-charge associative memory. *IEEE Transactions on Emerging Topics in Computing*, PP(99):1–1, 2017.

[55] M. Imani, S. Patil, and T. S. Rosing. Masc: Ultra-low energy multiple-access single-charge tcam for approximate computing. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 373–378, March 2016.

[56] M. Imani, D. Peroni, A. Rahimi, and T. Rosing. Resistive cam acceleration for tunable approximate computing. *IEEE Transactions on Emerging Topics in Computing*, PP(99):1–1, 2017.

[57] M. Imani, D. Peroni, and T. Rosing. Nvalt: Non-volatile approximate lookup table for gpu acceleration. *IEEE Embedded Systems Letters*, PP(99):1–1, 2017.

[58] M. Imani, A. Rahimi, P. Mercati, and T. Rosing. Multi-stage tunable approximate search in resistive associative memory. *IEEE Transactions on Multi-Scale Computing Systems*, PP(99):1–1, 2017.

[59] M. Imani, A. Rahimi, and T. S. Rosing. Resistive configurable associative memory for approximate computing. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1327–1332, March 2016.

[60] M. Imani and T. Rosing. Cap: Configurable resistive associative processor for near-data computing. In *2017 18th International Symposium on Quality Electronic Design (ISQED)*, pages 346–352, March 2017.

[61] C. Inc. 1t1r product: Picoe - embedded resistive ram, 2016.

[62] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Prodromakis. Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology*, 24(38):384010, 2013.

[63] Intel. Intel product specifications.

[64] Intel. Intel xpoint. News Release, 2015.

[65] Intel Altera. Fpga architecture white paper, July 2006.

[66] International Roadmap for Devices and Systems. Emerging research devices. Technical report, 2011.

[67] International Roadmap for Devices and Systems. Beyond cmos (emerging research devices), white paper, 2016.

[68] International Roadmap for Devices and Systems. More moore, white paper. Technical report, IEEE, 2016.

[69] E. Ipek, Q. Guo, X. Guo, and Y. Bai. *Resistive Memories in Associative Computing*, pages 201–229. Springer New York, New York, NY, 2014.

[70] H. Ishiwara, M. Okuyama, and Y. Arimoto. *Ferroelectric random access memories: fundamentals and applications*, volume 93. Springer Science & Business Media, 2004.

[71] M. ITOH and L. O. CHUA. Memristor oscillators. *International Journal of Bifurcation and Chaos*, 18(11):3183–3206, 2008.

[72] ITRS. ITRS Reports. Technical report, International Technology Roadmap for Semiconductors.

[73] ITU-R. Bt.601 : Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios, Mar. 2011.

[74] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Letters*, 10(4):1297–1301, Apr 2010.

[75] S. H. Jo, K.-H. Kim, and W. Lu. High-density crossbar arrays based on a si memristive system. *Nano Letters*, 9(2):870–874, 2009. PMID: 19206536.

[76] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12, New York, NY, USA, 2017. ACM.

[77] R. Kaplan, L. Yavits, and R. Ginosar. Prins: Processing-in-storage acceleration of machine learning. *IEEE Transactions on Nanotechnology*, pages 1–1, 2018.

[78] G. Karakonstantis, D. Mohapatra, and K. Roy. Logic and memory design based on unequal error protection for voltage-scalable, robust and adaptive dsp systems. *J. Signal Process. Syst.*, 68(3):415–431, Sept. 2012.

[79] U. R. Karpuzcu, N. S. Kim, and J. Torrellas. Coping with parametric variation at near-threshold voltages. *IEEE Micro*, 33(4):6–14, July 2013.

[80] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas. Varius-ntv: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. In *Proceedings of the 2012 42Nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, DSN '12, pages 1–11, Washington, DC, USA, 2012. IEEE Computer Society.

[81] O. Kavehei, S. Al-Sarawi, K. R. Cho, N. Iannella, S. J. Kim, K. Eshraghian, and D. Abbott. Memristor-based synaptic networks and logical operations using in-situ computing. In *2011 Seventh International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 137–142, Dec 2011.

[82] O. Kavehei, K. Cho, S. Lee, S. J. Kim, S. Al-Sarawi, D. Abbott, and K. Eshraghian. Fabrication and modeling of ag/tio2/ito memristor. In *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4, Aug 2011.

[83] S. Khasanvis, M. Rahman, and C. A. Moritz. Heterogeneous graphenecmos ternary content addressable memory. *Journal of Parallel and Distributed Computing*, 74(6):2497 – 2503, 2014. Computing with Future Nanotechnology.

[84] D. Kim, K. Lee, S. joong Lee, and H.-J. Yoo. A reconfigurable crossbar switch with adaptive bandwidth control for networks-on-chip. In *2005 IEEE International Symposium on Circuits and Systems*, pages 2369–2372 Vol. 3, May 2005.

[85] H. Kim, M. P. Sah, C. Yang, T. Roska, and L. O. Chua. Neural synaptic weighting with a pulse-based memristor circuit. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(1):148–158, Jan 2012.

[86] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu. A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. *Nano letters*, 12(1):389–395, 2011.

[87] K.-H. Kim, S. Gaba, D. Wheeler, J. M. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu. A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. *Nano Letters*, 12(1):389–395, 2012. PMID: 22141918.

[88] K. M. Kim, J. J. Yang, J. P. Strachan, E. M. Grafals, N. Ge, N. D. Melendez, Z. Li, and R. S. Williams. Voltage divider effect for the improvement of variability and endurance of taox memristor. *Scientific Reports*, 6:20085 EP –, Feb 2016. Article.

[89] S. Kim, H. Y. Jeong, S. K. Kim, S.-Y. Choi, and K. J. Lee. Flexible memristive memory array on plastic substrates. *Nano Letters*, 11(12):5438–5442, 2011. PMID: 22026616.

[90] Y. Kim, M. Imani, and T. Rosing. Orchard: Visual object recognition accelerator based on approximate in-memory processing.

[91] P. N. V. Kiran and N. Saxena. Design and analysis of different types sram cell topologies. In *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*, pages 1060–1065, Feb 2015.

[92] C. E. Kozyrakis and D. A. Patterson. A new direction for computer architecture research. *Computer*, 31(11):24–32, Nov 1998.

[93] C. E. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, and K. Yelick. Scalable processors in the billion-transistor era: Iram. *Computer*, 30(9):75–78, Sep 1997.

[94] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[95] S. Kumar, N. Davila, Z. Wang, X. Huang, J. P. Strachan, D. Vine, A. D. Kilcoyne, Y. Nishi, and R. S. Williams. Spatially uniform resistance switching of low current, high endurance titanium–niobium-oxide memristors. *Nanoscale*, 9(5):1793–1798, 2017.

[96] S. Kunkel, M. Schmidt, J. M. Eppler, H. E. Plesser, G. Masumoto, J. Igarashi, S. Ishii, T. Fukai, A. Morrison, M. Diesmann, and M. Helias. Spiking network simulation code for petascale computers. *Frontiers in Neuroinformatics*, 8:78, 2014.

[97] I. Kuon, R. Tessier, and J. Rose. Fpga architecture: Survey and challenges. *Found. Trends Electron. Des. Autom.*, 2(2):135–253, Feb. 2008.

[98] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser. The desired memristor for circuit designers. *IEEE Circuits and Systems Magazine*, 13(2):17–22, Secondquarter 2013.

[99] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser. Team: Threshold adaptive memristor model. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(1):211–221, Jan 2013.

[100] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo, U.-I. Chung, I.-K. Yoo, and K. Kim. A fast, high-endurance and scalable non-volatile memory device made from asymmetric ta2o5-x/tao2-x bilayer structures. *Nature Materials*, 10:625 EP –, Jul 2011. Article.

[101] E. Lehtonen and M. Laiho. Stateful implication logic with memristors. In *2009 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 33–36, July 2009.

[102] H. Li, Z. Jiang, P. Huang, Y. Wu, H.-Y. Chen, B. Gao, X. Y. Liu, J. F. Kang, and H.-S. P. Wong. Variation-aware, reliability-emphasized design and optimization of rram using spice model. In *Proceedings of the 2015 Design, Automation & Test in Europe*

Conference & Exhibition, DATE '15, pages 1425–1430, San Jose, CA, USA, 2015. EDA Consortium.

[103] J. Li, R. K. Montoye, M. Ishii, and L. Chang. 1 mb 0.41 mm2; 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing. *IEEE Journal of Solid-State Circuits*, 49(4):896–907, April 2014.

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

[105] J. Liu, T. Li, S. Duan, and L. Wang. Energy consumption analysis for the read and write mode of the memristor with voltage threshold in the real-time control system. *Neurocomputing*, 266:477 – 484, 2017.

[106] Z. Liu, W. Wen, L. Jiang, Y. Jin, and G. Quan. A statistical stt-ram retention model for fast memory subsystem designs. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 720–725, Jan 2017.

[107] W. Lu, K. H. Kim, T. Chang, and S. Gaba. Two-terminal resistive switches (memristors) for memory and logic applications. In *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*, pages 217–223, Jan 2011.

[108] P. Lugli, A. Mahmoud, G. Csaba, M. Algasinger, M. Stutzmann, and U. Rhrmair. Physical unclonable functions based on crossbar arrays for cryptographic applications. *International Journal of Circuit Theory and Applications*, 41(6):619–633, 2013.

[109] J. Makhoul. A fast cosine transform in one and two dimensions. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(1):27–34, Feb 1980.

[110] H. Manem, G. S. Rose, X. He, and W. Wang. Design considerations for variation tolerant multilevel cmos/nano memristor memory. In *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI*, GLSVLSI '10, pages 287–292, New York, NY, USA, 2010. ACM.

[111] N. M. Manjunath Shevgoor, Rajeev Balasubramonian. Designing a fast and reliable memory with memristor technology. 6th Annual Non-Volatile Memories Workshop 2015, 2015.

[112] S. Marsland. *Machine Learning: An Algorithmic Perspective*. CRC Press, 2011.

[113] F. Merrikh-Bayat and S. B. Shouraki. Memristor-based circuits for performing basic arithmetic operations. *Procedia Computer Science*, 3:128 – 132, 2011. World Conference on Information Technology.

[114] F. Miao, J. P. Strachan, J. J. Yang, M.-X. Zhang, I. Goldfarb, A. C. Torrezan, P. Eschbach, R. D. Kelley, G. Medeiros-Ribeiro, and R. S. Williams. Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor. *Advanced Materials*, 23(47):5633–5640, 2011.

[115] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. Quality of service profiling. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 25–34, New York, NY, USA, 2010. ACM.

[116] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):62:1–62:33, Mar. 2016.

[117] D. Mohapatra, V. Chippa, A. Raghunathan, and K. Roy. Design of voltage-scalable meta-functions for approximate computing. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011.

[118] A. Morad, L. Yavits, S. Kvatinsky, and R. Ginosar. Resistive gp-simd processing-in-memory. *ACM Trans. Archit. Code Optim.*, 12(4):57:1–57:22, Jan. 2016.

[119] M. A. Neggaz, H. E. Yantr, S. Niar, A. Eltawil, and F. Kurdahi. Rapid in-memory matrix multiplication using associative processor. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 985–990, March 2018.

[120] D. Niu, Y. Chen, C. Xu, and Y. Xie. Impact of process variations on emerging memristor. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 877–882, June 2010.

[121] A. S. Oblea, A. Timilsina, D. Moore, and K. A. Campbell. Silver chalcogenide based memristor devices. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–3, July 2010.

[122] K. Pagiamtzis and A. Sheikholeslami. Content-addressable memory (cam) circuits and architectures: a tutorial and survey. *IEEE Journal of Solid-State Circuits*, 41(3):712–727, March 2006.

[123] Panasonic Corporation. Panasonic mn101l resistive ram embedded 8-bit mcus, 12 2016.

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

[125] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1992.

[126] Y. V. Pershin and M. D. Ventra. Experimental demonstration of associative memory with memristive neural networks. *Neural Networks*, 23(7):881 – 886, 2010.

[127] Y. V. Pershin and M. D. Ventra. Practical approach to programmable analog circuits with memristors. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(8):1857–1864, Aug 2010.

[128] N. Pinckney, M. Fojtik, B. Giridhar, D. Sylvester, and D. Blaauw. Shortstop: An on-chip fast supply boosting technique. In *2013 Symposium on VLSI Circuits*, pages C290–C291, June 2013.

[129] J. L. Potter. *Associative Computing: A Programming Paradigm for Massively Parallel Computers*. Perseus Publishing, 1991.

[130] P. Pouyan, E. Amat, and A. Rubio. Statistical lifetime analysis of memristive crossbar matrix. In *2015 10th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6, April 2015.

[131] T. Prodromakis, C. Toumazou, and L. Chua. Two centuries of memristors. *Nature Materials*, 11:478 EP –, May 2012.

[132] A. G. Radwan, M. A. Zidan, and K. N. Salama. On the mathematical modeling of memristors. In *2010 International Conference on Microelectronics*, pages 284–287, Dec 2010.

[133] A. Rahimi, A. Ghofrani, K. T. Cheng, L. Benini, and R. K. Gupta. Approximate associative memristive memory for energy-efficient gpus. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1497–1502, March 2015.

[134] J. Rajendran, R. Karri, and G. S. Rose. Improving tolerance to variations in memristor-based applications using parallel memristors. *IEEE Transactions on Computers*, 64(3):733–746, March 2015.

[135] M. Renovell, J. M. Portal, J. Figueras, and Y. Zorian. Testing the interconnect of ram-based fpgas. *IEEE Design Test of Computers*, 15(1):45–50, Jan 1998.

[136] W. Robinett, M. Pickett, J. Borghetti, Q. Xia, G. S. Snider, G. Medeiros-Ribeiro, and R. S. Williams. A memristor-based nonvolatile latch circuit. *Nanotechnology*, 21(23):235203, 2010.

[137] K. Roy. Approximate computing for energy-efficient error-resilient multimedia systems. In *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2013 IEEE 16th International Symposium on*, pages 5–6, April 2013.

[138] J. A. Rudolph. A production implementation of an associative array processor: Staran. In *Proceedings of the December 5-7, 1972, Fall Joint Computer Conference, Part I*, AFIPS '72 (Fall, part I), pages 229–241, New York, NY, USA, 1972. ACM.

[139] S. Ruhman and I. Scherson. Associative processor particularly useful for tomographic image reconstruction, Jan. 1 1985. US Patent 4,491,932.

[140] N. Sakimura, T. Sugibayashi, T. Honda, H. Honjo, S. Saito, T. Suzuki, N. Ishiwata, and S. Tahara. Mram cell technology for over 500-mhz soc. *IEEE Journal of Solid-State Circuits*, 42(4):830–838, April 2007.

[141] I. Scherson and S. Ruhman. Multi-operand associative arithmetic. In *1983 IEEE 6th Symposium on Computer Arithmetic (ARITH)*, pages 123–129, June 1983.

[142] I. D. Scherson and S. Ilgen. A reconfigurable fully parallel associative processor. *J. Parallel Distrib. Comput.*, 6(1):69–89, Feb. 1989.

[143] I. D. Scherson, D. A. Kramer, and B. D. Alleyne. Bit-parallel arithmetic in a massively-parallel associative processor. *IEEE Transactions on Computers*, 41(10):1201–1210, Oct 1992.

[144] I. D. Scherson and S. Ruhman. Multi-operand arithmetic in a partitioned associative architecture. *Journal of Parallel and Distributed Computing*, 5(6):655 – 668, 1988.

[145] D. Schinkel, E. Mensink, E. Klumperink, E. van Tuijl, and B. Nauta. A double-tail latch-type voltage sense amplifier with 18ps setup+hold time. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 314–605, Feb 2007.

[146] M. Seok, D. Jeon, C. Chakrabarti, D. Blaauw, and D. Sylvester. A 0.27v 30mhz 17.7nj/transform 1024-pt complex fft core with super-pipelining. In *2011 IEEE International Solid-State Circuits Conference*, pages 342–344, Feb 2011.

[147] Y. Shain, A. Akerib, and R. Adar. Associative architecture for fast dct. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 5, pages 3109–3112 vol.5, May 1998.

[148] S. Shin, K. Kim, and S. M. Kang. Memristor applications for programmable analog ics. *IEEE Transactions on Nanotechnology*, 10(2):266–274, March 2011.

[149] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 124–134, New York, NY, USA, 2011. ACM.

[150] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[151] S. Sinha, G. Yeric, V. Chandra, B. Cline, and Y. Cao. Exploring sub-20nm finfet design with predictive technology models. In *DAC Design Automation Conference 2012*, pages 283–288, June 2012.

[152] SiSoftware. Sandra benchmarking software.

[153] S. Smaili and Y. Massoud. Studying the effect of memristor state variability on the gain of memristor-based tunable amplifiers. In *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 912–915, Aug 2013.

[154] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams. The missing memristor found. *Nature*, 453(7191):80–83, May 2008.

[155] M. M. A. Taha, W. Woods, and C. Teuscher. Approximate in-memory hamming distance calculation with a memristive associative memory. In *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 159–164, July 2016.

[156] A. Talukdar, A. Radwan, and K. Salama. Generalized model for memristor-based wien family oscillators. *Microelectronics Journal*, 42(9):1032 – 1038, 2011.

[157] A. Talukdar, A. Radwan, and K. Salama. Non linear dynamics of memristor based 3rd order oscillatory system. *Microelectronics Journal*, 43(3):169 – 175, 2012.

[158] R. Tetzlaff. *Memristors and memristive systems*. Springer, 2013.

[159] A. S. University. Predictive technology model (ptm), 2012.

[160] J. Valsa, D. Biolek, and Z. Biolek. An analogue model of the memristor. *Int. J. Numer. Model.*, 24(4):400–408, July 2011.

[161] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 796–801, June 2012.

[162] J. von Neumann. First draft of a report on the edvac. *IEEE Ann. Hist. Comput.*, 15(4):27–75, Oct. 1993.

[163] P. O. Vontobel, W. Robinett, P. J. Kuekes, D. R. Stewart, J. Straznicky, and R. S. Williams. Writing to and reading from a nano-scale crossbar memory based on memristors. *Nanotechnology*, 20(42):425204, 2009.

[164] D. Wang, Z. Hu, X. Yu, and J. Yu. A pwl model of memristor and its application example. In *2009 International Conference on Communications, Circuits and Systems*, pages 932–934, July 2009.

[165] K. Wu, F. Ober, S. Hamlin, and D. Li. Early evaluation of intel optane non-volatile memory with hpc i/o workloads. *arXiv preprint arXiv:1708.02199*, 2017.

[166] Q. Xia, W. Robinett, M. W. Cumbie, N. Banerjee, T. J. Cardinali, J. J. Yang, W. Wu, X. Li, W. M. Tong, D. B. Strukov, G. S. Snider, G. Medeiros-Ribeiro, and R. S. Williams. Memristorcmos hybrid integrated circuits for reconfigurable logic. *Nano Letters*, 9(10):3640–3645, 2009. PMID: 19722537.

[167] Q. Xia, J. J. Yang, W. Wu, X. Li, and R. S. Williams. Self-aligned memristor cross-point arrays fabricated with one nanoimprint lithography step. *Nano Letters*, 10(8):2909–2914, Aug 2010.

[168] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino. Generalized memristive device spice model and its application in circuit design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(8):1201–1214, Aug 2013.

[169] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino. Memristor spice model and crossbar simulation based on devices with nanosecond switching time. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, Aug 2013.

[170] C. H. Yang, T. H. Yu, and D. Markovic. Power and area minimization of reconfigurable fft processors: A 3gpp-lte example. *IEEE Journal of Solid-State Circuits*, 47(3):757–768, March 2012.

[171] J. J. Yang, D. B. Strukov, and D. R. Stewart. Memristive devices for computing. *Nature nanotechnology*, 8(1):13, 2013.

[172] J. J. Yang, M.-X. Zhang, J. P. Strachan, F. Miao, M. D. Pickett, R. D. Kelley, G. Medeiros-Ribeiro, and R. S. Williams. High switching endurance in taox memristive devices. *Applied Physics Letters*, 97(23):232102, 2010.

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

[174] H. E. Yantir, A. M. Eltawil, and F. J. Kurdahi. Approximate memristive in-memory computing. *ACM Trans. Embed. Comput. Syst.*, 16(5s):129:1–129:18, Sept. 2017.

[175] H. E. Yantir, M. E. Fouda, A. M. Eltawil, and F. J. Kurdahi. Process variations-aware resistive associative processor design. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 49–55, Oct 2016.

[176] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar. Resistive associative processor. *IEEE Computer Architecture Letters*, 14(2):148–151, July 2015.

[177] L. Yavits, A. Morad, and R. Ginosar. Computer architecture with associative processor replacing last-level cache and simd accelerator. *IEEE Transactions on Computers*, 64(2):368–381, Feb 2015.

[178] L. Yavits, A. Morad, and R. Ginosar. Sparse matrix multiplication on an associative processor. *IEEE Transactions on Parallel and Distributed Systems*, 26(11):3175–3183, Nov 2015.

[179] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran. Axbench: A multiplatform benchmark suite for approximate computing. *IEEE Design Test*, 34(2):60–68, April 2017.

[180] A. Yazdanbakhsh, D. Mahajan, B. Thwaites, J. Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, H. Esmaeilzadeh, and K. Bazargan. Axilog: Language support for approximate hardware design. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 812–817, March 2015.

[181] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh. Neural acceleration for gpu throughput processors. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pages 482–493, New York, NY, USA, 2015. ACM.

[182] S. P. Young, K. Chaudhary, and T. J. Bauer. Fpga repeatable interconnect structure with hierarchical interconnect lines, June 22 1999. US Patent 5,914,616.

[183] H. Zhao, L. Xue, P. Chi, and J. Zhao. Approximate image storage with multi-level cell stt-mram main memory.

[184] M. A. Zidan, A. M. Eltawil, F. Kurdahi, H. A. H. Fahmy, and K. N. Salama. Memristor multiport readout: A closed-form solution for sneak paths. *IEEE Transactions on Nanotechnology*, 13(2):274–282, March 2014.

[185] M. A. Zidan, H. Omran, A. G. Radwan, and K. N. Salama. Memristor-based reactance-less oscillator. *Electronics Letters*, 47(22):1220–1221, Oct 2011.

[186] M. A. Zidan, H. Omran, C. Smith, A. Syed, A. G. Radwan, and K. N. Salama. A family of memristor-based reactance-less oscillators. *International Journal of Circuit Theory and Applications*, 42(11):1103–1122, 2014.