Compute-in-memory Circuits and Architectures for Efficient Acceleration of AI and Data Centric Workloads

by

Amitesh Sridharan

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

Approved June 2025 by the
Graduate Supervisory Committee:

Deliang Fan, Chair
Jae-sun Seo
Yu Cao
Jeff Zhang

ARIZONA STATE UNIVERSITY

August 2025

ABSTRACT

Modern computing is increasingly driven by the explosive growth of data from applications such as Artificial Intelligence (AI), Machine Learning (ML), and Genomics. These workloads are inherently data-intensive, requiring fast and efficient processing of large datasets. Although scaling input data in AI applications continues to boost performance, traditional computing architectures have struggled to keep pace, creating a widening gap between data generation and processing capabilities.

This disparity stresses the three fundamental pillars of computing—storage, communication, and computation—impacting performance, energy efficiency, and cost. Conventional Von Neumann architectures, designed to maximize computational throughput, now face the "memory and power wall," where compute units cannot fetch or process data fast enough to meet demand. As data movement becomes the dominant bottleneck, there is a clear need to pivot from compute-centric to memory-centric design approaches.

In-Memory Computing (IMC), or Compute-in-Memory (CIM), addresses these challenges by performing computation directly within memory, minimizing data movement and mitigating the memory wall.

This dissertation introduces a series of digital CIM circuits and architectures that significantly improve power, performance, and area (PPA) metrics for data-intensive workloads. It begins with a programmable CIM design that balances the flexibility of Central-Processing-Units(CPUs)/Graphics Processing Units(GPUs) with the efficiency of ASICs, enabling a broad class of applications. A prototype 28nm CMOS chip is then presented to accelerate general matrix-matrix multiplications (GEMMs) across various fixed-point precisions.

The focus then shifts to sparse GEMM acceleration. The first design demonstrates how CIM tailored for channel decoders leverages both fixed and unstructured sparsity to outperform conventional designs. The second design, fabricated in 28nm CMOS, supports diverse unstructured sparse formats and integer precisions, efficiently targeting highly sparse deep neural networks (DNNs). The final design achieves state-of-the-art efficiency in compressed sparse GEMMs, supporting both integer and floating-point data types using shared hardware. It also integrates a RISC-V CPU to manage computation across diverse matrix sizes and model types.

Together, these contributions advance CIM as a scalable and efficient platform for future AI and data-centric systems.

ACKNOWLEDGMENTS

I owe my deepest gratitude to my wife, Vaishali Sridhar, my parents, Meera Natarajan and Sridharan Kanakarajan, and my brother, Visvesh Sridharan, for their unwavering love, encouragement, and support throughout every stage of my Ph.D. journey. This dissertation is dedicated to them.

I am profoundly grateful to my advisor and committee chair, Dr. Deliang Fan, for his exceptional mentorship, insightful guidance, and constant encouragement during my doctoral studies. His support has been instrumental in shaping my research direction, refining my writing, and developing essential skills in time management and critical thinking. I am especially thankful for his patience, his belief in my potential, and his dedication to guiding me through each phase of my Ph.D. It has truly been an honor and privilege to be his student.

I would also like to express my heartfelt appreciation to Dr. Jae-sun Seo, a member of my Ph.D. advisory committee, for his outstanding mentorship and continued support. From the very beginning of my Ph.D., he has been a constant source of inspiration. I am sincerely grateful for the opportunities he provided, his thoughtful feedback, and, most importantly, for treating me as one of his own students. His influence has had a lasting impact on both my academic and personal growth, and I deeply aspire to follow the example he sets.

My sincere thanks go to Dr. Yu (Kevin) Cao and Dr. Jeff Zhang for their invaluable guidance throughout my research projects and for their generous service as members of my Ph.D. committee. I am also thankful to Dr. Bo Yuan, Dr. Shan X. Wang, Dr. Wilman Tsai, and Dr. Yiran Chen for their collaboration and insightful contributions, which have greatly enriched the scope and impact of my work. Additionally, I would like to thank Dr. Harsono Simka and Dr. Ming He from Samsung Semiconductor Inc., and Dr. Kerem Akarvardar and Dr. Xiaoyu Sun from TSMC Corporate Research, for their valuable feedback and technical mentorship during my summer internships. Their guidance has significantly shaped my research direction and professional development.

I am deeply indebted to my colleagues and collaborators Dr. Shaahin Angizi, Dr. Fan Zhang, Asmer Hamid Ali, Dr. Sai Kiran Cherupally, Dr. Jyotishman Saikia, Anupreetham, Md. Habibur Rahman, Dr. Adnan Siraj Rakin, Dr. Li Yang, Yongjae Lee, Yuan Liao, Dr. Shreyas Venkataramanaiah, Jingxing Li, Jingkai Guo, Yaotian Liu, Dr. Injune Yeo, Haotian Su, Boyang Cheng, Yucheng Wang, and Jiawei Hu—for their collaborative spirit, insightful discussions, and steadfast support throughout my Ph.D. Their friendship and contributions have been instrumental to both the progress and enjoyment of my graduate experience.

I also wish to thank my graduate advisors, Lynn Pratt and Ian James, for their continuous support with administrative matters and logistics. Their guidance ensured a smooth and productive academic journey.

Finally, I would like to thank my friend Magesh Sridhar for his encouragement and support. His positivity and friendship have meant a great deal to me and have played an important role in the successful completion of this dissertation.

This work is supported in part by the National Science Foundation under Grant No.2314591, No.2505326, No.2528723, No.2528767, and ASU.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Today's Computing Limitations

Over the past several decades, compute performance was primarily limited by hardware capabilities, but this was consistently overcome by advances in transistor scaling, as described by Moore's Law (doubling the number of transistors per chip) and Dennard scaling (allowing for higher clock speeds and lower power per transistor as they shrank) However, as we approach the physical and thermodynamic limits of miniaturization, both Moore's Law and Dennard scaling have slowed significantly, making further gains increasingly difficult and expensive Das (2015).

At the same time, the traditional Von-Neumann computing paradigm faces inherent bottlenecks that are now more pronounced. The most significant is the Von-Neumann bottleneck, which arises from the separation of memory and compute units. In this architecture, data and instructions must travel back and forth across a shared bus, limiting throughput and causing the processor to idle while waiting for data transfers. As processor speeds have far outpaced improvements in memory bandwidth, this bottleneck has only worsened, especially for data-intensive applications like AI and large-scale data analytics. Additionally, the serial, step-by-step nature of Von-Neumann processors restricts their ability to efficiently handle highly parallel workloads, further limiting performance gains even as more cores are added. Attempts to mitigate these issues—such as using larger caches, branch prediction, and multi-core designs—have provided incremental improvements but have not solved the fundamental architectural limitations.

The memory system which spans from fast SRAM caches to DRAM main memory and even to slower storage like SSDs is responsible for a disproportionate share of the system's energy consumption, performance bottlenecks, reliability issues, and cost. As applications continue to evolve particularly in domains such as machine learning, genomics, graph processing, and large-scale data analytics they are becoming far more data intensive, exacerbating the pressure on memory systems. The result is a bottleneck between communication data path between the compute and memory. This bottleneck, known as the memory wall problem and results in substantial data movement overhead. A key observation from Mutlu (2018) highlights that, in a traditional CPU, the data movement required for a simple ADD operation can cost up to 115× more than the computation itself.

1.2 Memory Centric Compute Paradigm

A promising solution to the memory wall problem is the Processing-in-Memory (PIM)—also referred to as Compute-in-Memory (CIM) or In-Memory Computing (IMC)—paradigm (Hereafter, the terms Processing-in-Memory (PIM), Compute-in-Memory (CIM), and In-Memory Computing (IMC) will be used interchangeably). This approach rethinks the traditional separation between computation and storage by placing computing elements as close to memory as possible, and in many cases, embedding them directly within the memory arrays. By doing so, it virtually eliminates the performance and energy bottlenecks caused by excessive data movement, addressing the issue from both circuit-level and architectural perspectives. Over the past decade, numerous studies have demonstrated the effectiveness of this paradigm across various memory types including SRAMs Zhang *et al.* (2017); Yin *et al.* (2020); Jiang *et al.* (2020); Hu *et al.* (2025), DRAMs Zhang *et al.* (2021a); Li *et al.* (2017); Ahn *et al.* (2015); Seshadri *et al.* (2017), and even emerging non-volatile memory technologies like MRAMs Sridharan *et al.* (2022b); Zhang *et al.* (2024b) and RRAMs Zhang *et al.* (2024a); Wang *et al.* (2023). The benefits of applying a memory-centric computing model vary depending on the memory type as well as the underlying compute methodology whether analog or digital. The analog computing model as in Zhang *et al.* (2017); Jiang *et al.* (2020), map discrete numbers to the current domain and uses current summing to perform arithmetic instead of traditional digital gates. This has huge potential when it comes to reduced computation resources but it requires the use of ADCs/DACs which tradeoffs compute accuracy with power, performance, area as well as robstness and variability Seo *et al.* (2022). On the other hand we have digital compute paradigm which is similar to digital ASICs but here the memory centric paradigm is adopted by closely interleaving compute and memory by directly driving compute logic with static components thereby effectively moving a part if not most of the compute within the bit-cell or into the memory array. This eliminates the need for memory reads and writes for compute effectively matching the memory bandwidth with the memory size. This has much more practical viability and can achieve energy efficiency close to analog compute. This dissertation specifically focuses on exploring digital compute-in-memory circuits and how they can be effectively used to accelerate data-intensive applications.

The key contributions of this thesis are:

- We propose a programmable Processing-in-Memory (PIM) architecture that unifies PIM paradigm with flexible programmability. This design allows a single memory-centric compute substrate to efficiently support and accelerate a wide range of data-intensive applications, rather than being limited to fixed-function accelerators.

- To demonstrate its practical viability, we implement and fabricate the programmable PIM design as a silicon prototype using 65nm CMOS technology. We then validate the architecture's effectiveness by mapping representative applications directly onto the chip. Through real-time execution and benchmarking, we demonstrate significant performance and energy benefits, highlighting the practicality and generalizability of our programmable PIM compute approach.
- Given the dominance of AI as a data-intensive workload, we explore how different characteristics of AI/ML models—including model size, numerical precision, data types, and sparsity—interact and trade off with one another in the context of in-memory computing. We propose a suite of CIM architectures tailored to effectively handle these diverse requirements and to deliver real-time compute benefits.
- To enable scalable precision, we present PS-IMC (Precision-Scalable In-Memory Compute), a novel CIM architecture that supports bit-parallel multiply-accumulate (MAC) operations across multiple integer precisions. PS-IMC achieves high compute efficiency by dynamically adapting to different precision levels through intelligent resource sharing within the compute pipeline. This flexibility allows it to deliver one of the highest compute throughputs reported to date for a wide range of integer formats. The design is fabricated in 28nm CMOS technology and is benchmarked against state-of-the-art CIM macros, demonstrating its superior performance and precision scalability.
- To efficiently handle sparsity, we propose two CIM architectures: DSPIMM

and SP-IMC. These are among the first in-memory compute designs to support unstructured sparsity and can directly operate on compressed weight representations stored within the memory. This is made possible through a time-interleaved sparse decode methodology, which achieves the benefits of memory compression while significantly reducing decoding complexity and overhead.

- We validate the DSPIMM architecture by mapping a sparse communication-driven deep neural network (DNN) algorithm—specifically, the Neural Belief Propagation (Neural BP) algorithm—onto the hardware. We compare its performance against traditional ASIC implementations to demonstrate the advantages of in-memory execution for sparse models.
- In SP-IMC we further explore how the time shared sparse decode methodology can be configured to support multiple sparse compression formats and a wide range of sparsity ratios, all while preserving bit-parallel MAC support and offering robust compute efficiency across use cases. A prototype of the proposed architecture, fabricated in 28nm CMOS technology, demonstrates more than a 4-50× reduction in compute resources required to run popular models such as ResNet, compared to baseline digital designs, highlighting the substantial benefits of CIM for real-world inference tasks.
- To address flexible data type support, we design a fused multiply-add (FMA) pipeline within the CIM fabric that can process both integer and floating-point formats across various precision levels. This allows hardware reuse across both INT8 and FP8 formats.
- We also explore an alternative semi-structured N:M sparsity methodology in CIMs, which offers a more hardware-friendly approach to sparse compute while

preserving high compression efficiency with better accuracy compared to a fully structured sparsity.

- To address the scalability of matrix operations across varying shapes and dimensions, we co-design a lightweight, custom RISC-V processor with our CIM macros. This hybrid architecture serves as a flexible controller to orchestrate data movement and computation across matrix tiles, enabling scalable and programmable matrix multiplication.
- All proposed features are integrated into a complete 28nm silicon prototype, and we perform comprehensive comparisons with state-of-the-art CIM designs to show that our system achieves best-in-class performance, power, and area efficiency.

1.3 Thesis Organization

The outline of this thesis is as follows:

- **Chapter 2** presents a programmable and generic SRAM (PSRAM) based Compute-in-memory macro prototyped in 65nm CMOS. This chapter goes over the memory-cell circuit design, macro architecture, variability analysis, chip measurement results, and case studies for power/performance characterization of prototype chip taped-out in commercial 65nm CMOS.
- **Chapter 3** presents a precision scalable Compute-in-memory macro prototyped in 28nm CMOS that proposes a hardware scaling methodology in CIM based design to accelerate variable fixed precision multiply-and-accumulate workloads primarily focused on accelerating matrix-vector multiplications. The CIM circuit

design, macro architecture, chip measurement results, workload characterizations and comparsion with prior SoTA CIM desings are included in this chapter.

- **Chapter 4** presents a system level CIM based acceleration hardware for an AI based channel decoder primarily used in communication applications. This chapter elaborates on channel codes, AL based channel decoding algorithms, a sparse matrix multiplier design based on CIM, algorithm and hardware performance characterization and comparison with prior works.
- **Chapter 5** presents a prototype chip fabricated in 28nm CMOS that accelerates generic unstructured sparse matrix-vector multiplications using a digital Compute-in-Memory (CIM) architecture. This design explores the integration of multiple sparsity formats and compression ratios within a single, monolithic CIM framework, enabling broad applicability across diverse AI workloads. By directly computing on compressed representations we show how the proposed architecture achieves substantial reductions in compute resource usage for large-scale AI models.
- **Chapter 6** presents a digital CIM-based fused multiply-add (FMA) matrix-vector accelerator prototype chip that supports semi-structured sparsity and integrates a custom RISC-V CPU for efficient data movement. We demonstrate how the FMA pipeline is designed to handle both integer and floating-point workloads using shared hardware. We then evaluate the accelerator on various CNN and LLM models with different semi-structured sparsity formats, all supported natively in hardware. The chapter also explores the idea of how a RISC-V core can serve as a programmable controller to manage data flow across diverse AI workloads.
- **Chapter 7** concludes the dissertation.

Chapter 2

PROGRAMMABLE AND GENERIC PROCESSING-IN-SRAM ACCELERATOR

This chapter introduces a generic and programmable Processing-in-SRAM (PSRAM) accelerator chip that enables acceleration of a variety of data-intensive workloads. Built on an 8T-SRAM array, the proposed design is the first to support the full set of Boolean logic operations—including 2 and 3-input NOR, NAND, XOR—along with majority and full-adder operations, all executed in a single cycle reduced to just memory reads. This level of functionality and flexibility within a single PSRAM platform enables a broad range of compute-in-memory applications, including parallel vector processing, neural network acceleration, and data encryption. To demonstrate its capabilities, the PSRAM accelerator is implemented within a 16 Kb SRAM macro and fabricated in a 65nm CMOS process. The prototype chip operates at a frequency of 1.23 GHz, making it one of the fastest programmable in-memory computing systems reported to date. The chip achieves a peak system-level throughput of 1.2 TOPS and delivers an energy efficiency of 34.98 TOPS/W at 1.2V, across the complete set of supported 2- and 3-input Boolean logic functions.

2.1 Introduction

Traditional Von Neumann computing architectures—such as central processing units (CPUs) and graphics processing units (GPUs)—have long been the backbone of general-purpose and high-performance computing. These platforms are valued for their programmability, flexibility, and ability to efficiently execute a broad spectrum of bitwise logic and arithmetic operations. However, as data-intensive applications continue to grow in both scale and complexity, these architectures increasingly face fundamental bottlenecks in-memory bandwidth and energy efficiency. The separation of memory and computation in Von Neumann systems results in frequent and energy-expensive data transfers between processing and storage units, which limits performance scalability and contributes significantly to power consumption. In response to these limitations, Processing-in-Memory (PIM) architectures have emerged as a promising alternative. By integrating computation capabilities directly within or near memory arrays, PIM designs aim to reduce data movement, lower energy consumption, and improve overall throughput. However, application-specific PIM architectures, while efficient for targeted tasks, often suffer from limited generality. These designs are typically optimized for a specific algorithm or application domain—such as convolutional neural networks or encryption—which makes them inflexible and unable to adapt to the fast-paced evolution of software workloads and algorithmic paradigms. As such, their utility diminishes when applied outside their narrowly defined operational scope, making them unsuitable for general-purpose use Biswas *et al.* (2018).

To overcome these limitations, recent research has focused on developing generic and programmable PIM architectures capable of supporting a wider range of applications. State-of-the-art designs, such as those presented in Wang *et al.* (2020), explore alternatives to conventional bit-parallel processing models by adopting bit-serial computing techniques. These approaches enable more compact and reconfigurable PIM implementations that support basic arithmetic and logic functions using a limited set of in-memory operations. Bit-serial methods decompose arithmetic computations into a sequence of Boolean operations executed over multiple cycles, which allows for flexible algorithm mapping without requiring extensive hardware specialization.

However, this added flexibility comes at the cost of increased latency and greater intermediate data movement. Since bit-serial implementations often require multiple compute cycles even for basic operations, they involve frequent read and write-back steps between memory cells and peripheral logic. This can negate some of the energy and performance benefits that PIM architectures are designed to deliver, especially in scenarios where high-throughput and low-latency computation are essential Biswas *et al.* (2018); Wang *et al.* (2020); Valavi *et al.* (2019); Zhang *et al.* (2018); Yue *et al.* (2020).

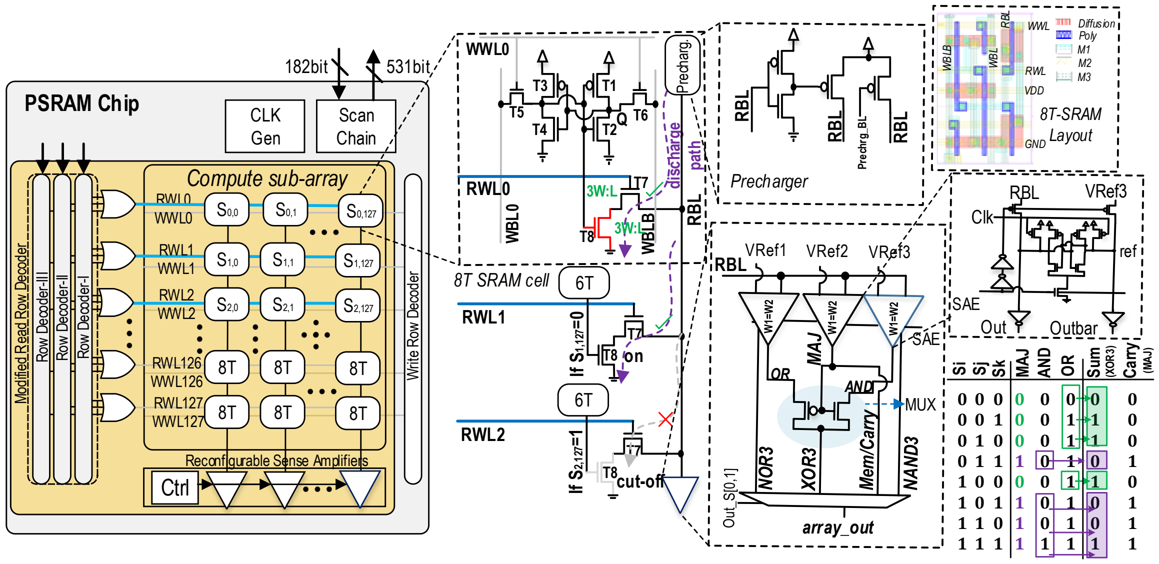
To address the limitations of existing application-specific and bit-serial programmable PIM designs, this work presents a programmable Processing-in-SRAM (PSRAM) accelerator that combines the high computational efficiency of in-memory processing with the flexibility of general-purpose programmability. Unlike prior works that are either limited to a narrow set of operations or require multiple compute cycles to implement basic logic functions, the proposed PSRAM supports a complete set of Boolean operations—including both 2-input and 3-input boolean logic operations (e.g., NOR, NAND, XOR), majority logic, and full adder functionality all executed within a single memory cycle. To the best of our knowledge, this is the first in-memory computing macro that realizes the complete set of logic functions in a single-cycle execution model.

This enables one of the fastest programmable in-memory computing systems reported to date, with the prototype design achieving a clock frequency of 1.23 GHz. The one-cycle logic execution model not only improves raw throughput but also addresses one of the major inefficiencies of previous programmable in-memory logic designs: redundant intermediate data write-back. In traditional multi-cycle PIM systems, more complex logic functions such as 3-input operations and full adders are typically realized through a sequence of basic Boolean steps, requiring multiple cycles. Each cycle may involve reading and writing intermediate results to and from memory, incurring both latency and energy overheads. By integrating support for complex Boolean operations within a single memory cycle, PSRAM eliminates this overhead, substantially reducing both latency and dynamic energy consumption Biswas *et al.* (2018); Wang *et al.* (2020); Valavi *et al.* (2019); Zhang *et al.* (2018); Yue *et al.* (2020).

PSRAM accelerator is demonstrated across three distinct application domains to showcase its versatility and general-purpose programmability. First, it supports bulk bitwise vector operations, which are commonly used in data-intensive workloads such as search, database filtering, and bitmap indexing. Second, the architecture enables efficient low-precision deep learning acceleration, leveraging its Boolean and arithmetic capabilities to execute quantized neural network inference directly within memory. Finally, PSRAM is applied to cryptographic computation by supporting the Advanced Encryption Standard (AES) algorithm, illustrating its capacity to handle complex bit-manipulation and control flow patterns required in security applications.

2.2 PSRAM Chip Circuit and Architecture

PSRAM leverages the charge-sharing feature of the 8T-SRAM cell on Read Bit-Line (RBL) and elevates it to implement 2-input and 3-input Boolean logic between two or three selected rows in a single memory read cycle. The key idea comes from the observation that certain discharge rate on the precharged RBL is determined by the data value stored in the simultaneously selected memory cells attached to the same bit-line. For instance, by activating three memory rows via Read Word-Lines (RWL), e.g., RWL0-RWL2 (Fig. 1), if $S_{0,0}$, $S_{1,0}$, and $S_{2,0}$ memory cells all store ‘1’s, then



Si	Sj	Sk	MAJ	AND	OR	Sum (XOR3)	Carry (MAJ)
0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	0
0	1	0	0	0	1	1	0
0	1	1	1	0	1	0	1
1	0	0	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	1	0	1	0	1
1	1	1	1	1	1	1	1

Figure 1. PSRAM Chip with 8t SRAM Cell as the Operand Memory and the Proposed Single-cycle Logic-SA Design (Adapted from Sridharan *et al.* (2022a), with Permission).

the read access transistors (T8) remain OFF, and the RBL precharged voltage does not discharge. On the other side, if all cells store ‘0’s, the RBL voltage will rapidly discharge through T8s. Similarly, based on different combinations of the values stored in those memory cells, the discharged voltage value will be different if sampled at a preset frequency, which could be sensed by our follow up ‘logic-SA’ design to implement different logic functions through selecting different voltage references. Theoretically, there will be four different voltage levels based on all possible combinations of three memory cell data in the same bit-line. In our design, to yield a sufficiently large sense margin, as shown in Monte Carlo simulations (Fig. 5), the read path transistor (T7 and T8) size is designed to be 3× as shown in Fig. 1. To implement a programmable logic function, a new re-configurable logic-SA is designed as in Fig. 1. It consists of three sub-SAs with voltage references (i.e., $V_{Ref1}<V_{Ref2}<V_{Ref3}$), each dedicated to distinct logic functions. In this way, by activating three memory rows (i.e., input operand vectors) at the same time, each sub-SA performs a neat voltage comparison between

the reference voltage and the discharged RBL voltages (w.r.t. different discharge rate corresponding to stored memory cell data), which respectively generates (N)OR3, (MAJ)MIN, and (N)AND3 logic output (complementary SA), and more importantly, at the same time. A novel single-cycle in-SRAM XOR3 (full adder's Sum) logic is developed through an interesting observation as shown in the bottom-right truth table of Fig. 1. When the majority function (MAJ) output (green box in the truth table) is '0', the corresponding XOR3's output is the same as the OR3's output. When the majority function output is '1', XOR3's output can be achieved through AND3 as highlighted by the purple box. Based on our last paragraph description, our logic-SA could simultaneously get the OR3, MAJ and AND3 logic outputs, then we propose to design the XOR3 logic through a two-transistor 2:1 multiplexer (with MAJ output as the selector) circuit highlighted in the proposed reconfigurable logic-SA. The Boolean logic of in-memory XOR3 can be given as $XOR3 = MAJ(S_i, S_j, S_k).AND(S_i, S_j, S_k)+ MIN(S_i, S_j, S_k).OR(S_i, S_j, S_k)$. In this way, assuming three vector operands are pre-stored in the memory, parallel in-memory full adder logic can be implemented for the first time in a single memory cycle, where MAJ and XOR3 outputs generate the carry-out and Sum signals, respectively. The two-input bit-wise operations will be readily implemented by initializing one row to '0'/'1'. All in-memory logic simulations are first shown in Fig. 2, showing corresponding functionality.

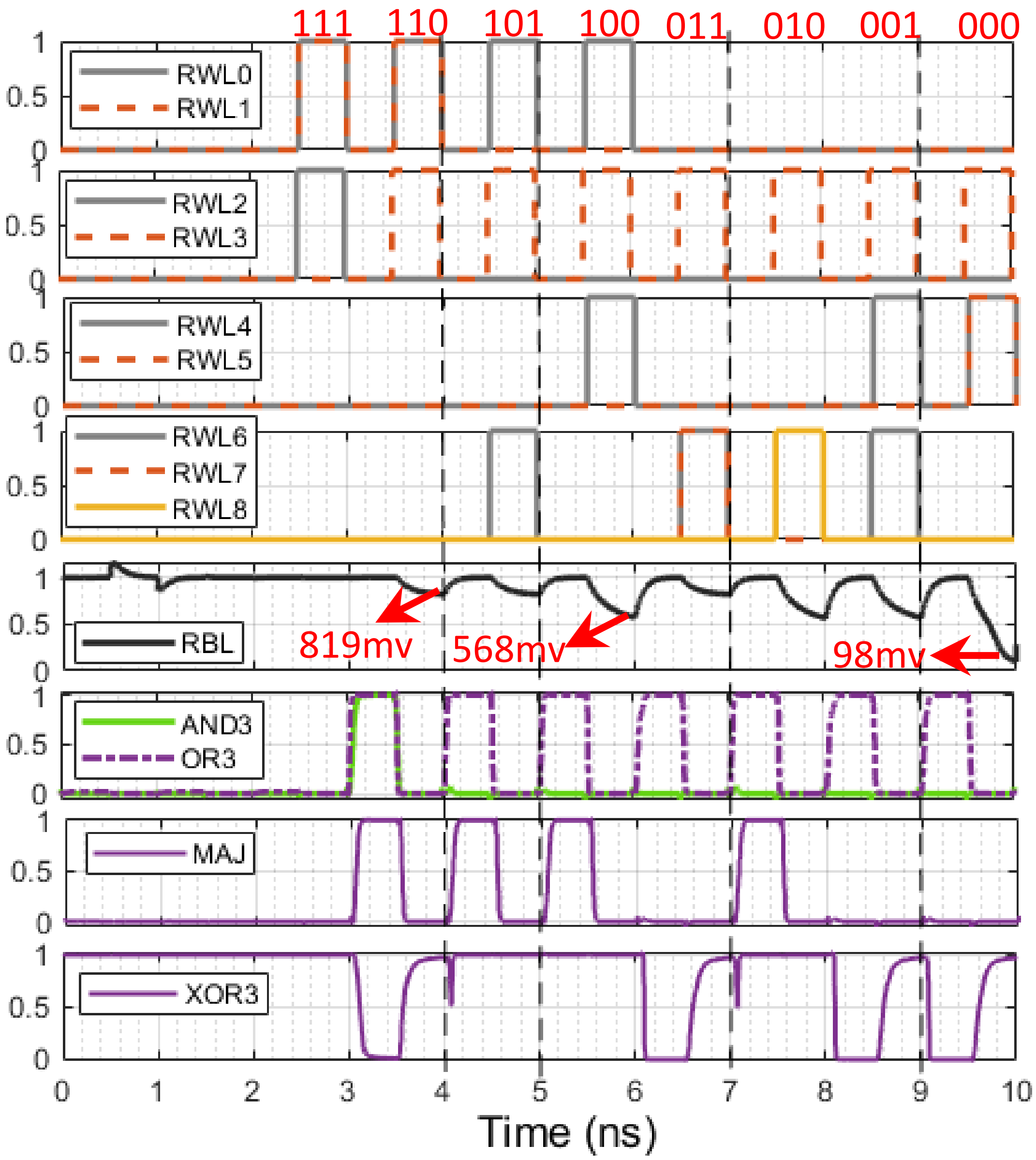


Figure 2. In-memory Logic Simulation Waveforms.

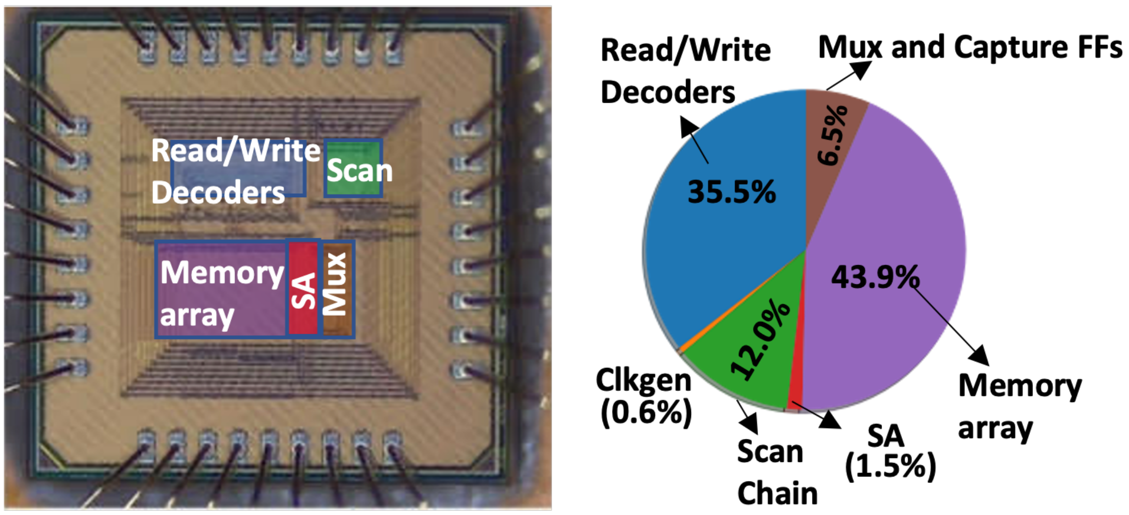


Figure 3. PSRAM Chip Micrograph and Area Breakdown

2.3 Chip Measurement Results

2.3.1 Performance Measurements

We prototyped the PSRAM macro (128×128) in TSMC 65nm CMOS (Fig. 3). The macro has a 2-KB/16Kb capacity and occupies 0.17 mm^2 (with decoder) in the chip floorplan. The bit-cell has an area of 4.56 μm^2 (1080 F^2 when scaled according to feature size), which is designed using logic rules. For efficient integration, the SAs are pitch matched w.r.t. the column and occupy 3.4% of the array size (0.082 mm^2). The complete core area breakdown is shown in Fig. 3. The PSRAM macro consumes 36 pJ (includes power consumed by all components on the die) and takes 813 ps to generate 512 outputs of the complete 3 input logic set (AND3, XOR3, OR3, MAJ). This represents a peak throughput of $2 \times 128 \times 4/813\text{ps} = 1259.52$ GOPs at 1.2V supply and a compute density of 583.12 GOPS/mm^2. PSRAM achieves a significant speedup of 4-157× when compared to state-of-the-art in-memory computing works Biswas

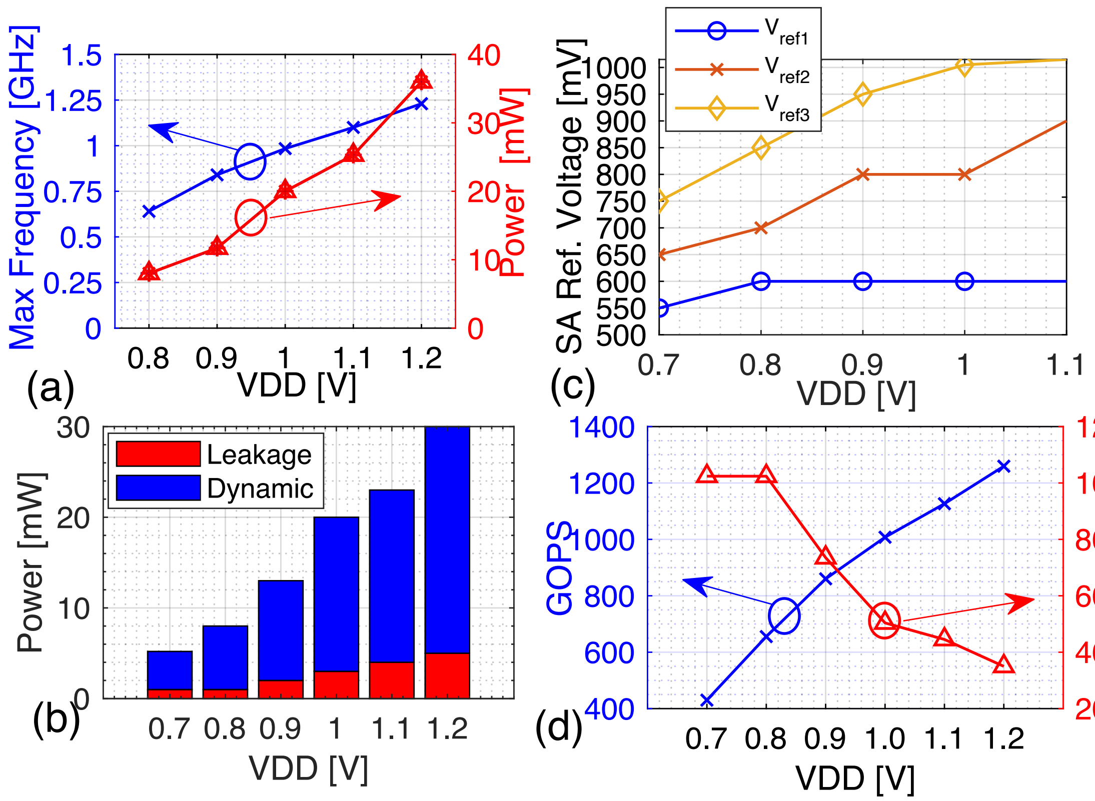


Figure 4. Frequency Scaling over Different Vdds (A), Static and Dynamic Power Consumption (B), Vref Scaling over Different Vdds (C), and Throughput Scaling over Different Vdds (D).

et al. (2018); Wang *et al.* (2020); Valavi *et al.* (2019); Zhang *et al.* (2018). We report the maximum frequency, power consumption, throughput w.r.t. different VDDs in Fig. 4.

2.3.2 Variability Analysis

The RBL sense margins are first tested through post-layout Monte Carlo simulations in Cadence Spectre for the four possible sensing voltages, as shown in Fig. 5, where

Table 1. Measured Reference Voltage Ranges

VDD/V_{Ref}	V_{Ref1} (V)	V_{Ref2} (V)	V_{Ref3} (V)
0.7V @ 0.42GHz	509m-546m	603m-647m	658m-693m
0.8V @ 0.64 GHz	452m-616m	620m-733m	745m-780m
0.9V @ 0.84 GHz	414m-661m	669m-750m	829m-889m
1.0V @ 0.98GHz	503m-711m	735m-902m	908m-995m
1.1V @ 1.1GHz	550m-754m	760m-994m	999m-1.083
1.2V @ 1.23GHz	554m-790m	815m-1.08	1.09-1.16

the sensing margin is reported considering both process (inter-die) and mismatch variations (intra-die) for core VDD (1.0 V) at 1 GHz.

During the chip measurements, off-chip voltage references are provided (V_{Ref}) to the SAs. To conduct the V_{Ref} variation analysis on chip, we test all 128 bit-lines, 100 times, for all possible bit value combinations in memory. 10 chips are tested and we report all the reference voltage ranges at different VDDs and the corresponding maximum frequencies with zero logic errors in Table 1. It is found that at lower voltages the maximum operating frequency is limited by the shrink of V_{Ref} ranges. A higher VDD also yields a larger sensing margin.

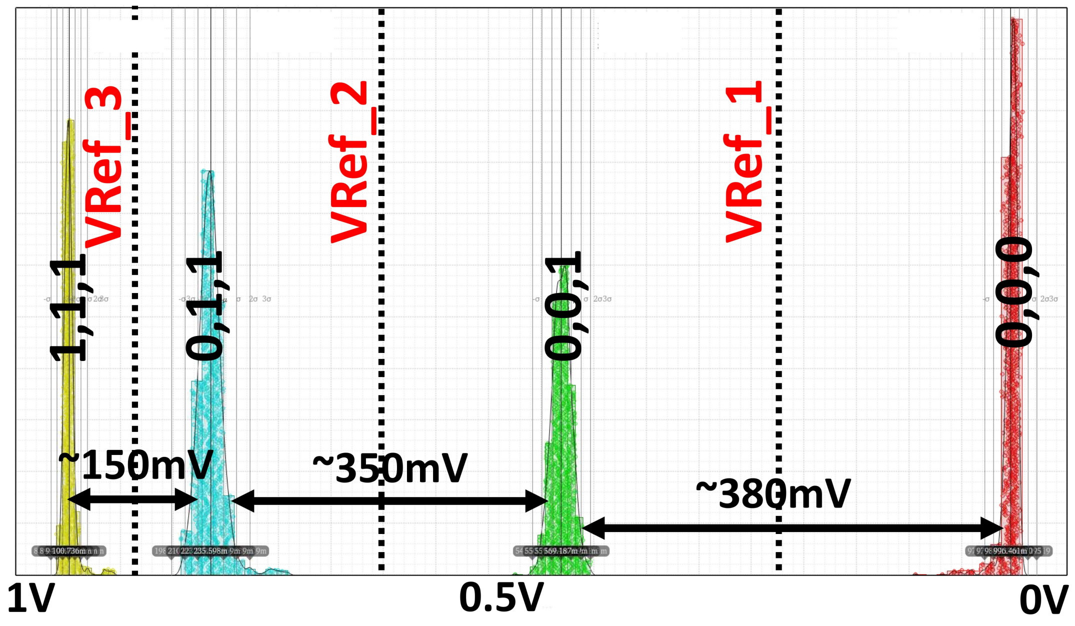


Figure 5. Monte-carlo Simulations for Vref Seperation Analysis

2.4 Application Evaluation and Comparison

2.4.1 Case Study 1: Bulk Bitwise Boolean Operations

The PSRAM could be leveraged to implement bulk bitwise Boolean logic operations efficiently between vectors stored in the same memory sub-array. This can lead to efficient re-use of the internal memory bandwidth. Table 2 compares the latency for a set of vector operations of interest, implemented by three generic PIM designs. We achieve the best performance of each design, where input vectors $\texttt{A}(a_0a_1...)$ $\texttt{B}(b_0b_1...)$ and $\texttt{C}(c_0c_1...)$ are stored in separate rows of the memory. We draw two conclusions from Table 2. First, our PSRAM is the only design that supports a full-set of Boolean logic (both 2-input and 3-input) and integer operations. Second, due to the complexity

Table 2. Latency Comparison of Vector Boolean Logic Operations Supported by PSRAM and Prior Accelerators

Parameters	Y. Zhang et al. (2018)	J. Wang et al. (2020)	PSRAM
Capacity (KB)	8	16	2
Technology (nm)	40	28	65
Frequency (GHz)	0.029	0.475	1.23
NOT (ns / # of Cycle)	34.72 / 1	2.1 / 1	**0.81 / 1**
NAND2 (ns / # of Cycle)	34.72 / 1	2.1 / 1	**0.81 / 1**
NAND3 (ns / # of Cycle)	69.44 / 2	4.2 / 2	**0.81 / 1**
NOR2 (ns / # of Cycle)	34.72 / 1	2.1 / 1	**0.81 / 1**
NOR3 (ns / # of Cycle)	69.44 / 2	4.2 / 2	**0.81 / 1**
X(N)OR2 (ns / # of Cycle)	34.72 / 1	2.1 / 1	**0.81 / 1**
XOR3 (ns / # of Cycle)	69.44 / 2	4.2 / 2	**0.81 / 1**
Majority (ns / # of Cycle)	n/a	n/a	**0.81 / 1**
FULL-ADD (ns / # of Cycle)	69.44 / 2	4.2 / 2	**0.81 / 1**
FULL-SUB (ns / # of Cycle)	69.44 / 2	4.2 / 2	**1.62 / 2**
ADD-RCA (4-bit) (ns # of Cycle)	n/a	n/a	**3.24 / 4**
ADD-CSA (4-bit) (ns # of Cycle)	n/a	n/a	**4.05 / 5**
ADD-Serial* (4-bit) (ns)	173.6	10.5	**4.05**
SUB-Serial† (4-bit) (ns)	312.48	18.9	**7.29**
MULT-Serial‡ (4-bit) (ns)	1180.48	71.4	**27.54**
MULT-Serial (8-bit) (ns)	3541.44	214.2	**82.62**

of some operations (e.g., ADD/SUB/MULT), they cannot be implemented in a time-efficient manner by the prior designs Zhang *et al.* (2018); Wang *et al.* (2020), while PSRAM outperforms all prior works in latency.

2.4.2 Case study 2: Binary Weight Neural Networks

We also implement the binary-weight neural network (BWNN) with various weight configurations for AlexNet and report the energy, latency and other performance in Table 3 and Fig. 7. The general HW/SW framework developed for BWNN consists of image and kernel banks, and PSRAM sub-arrays. Weights and activation are

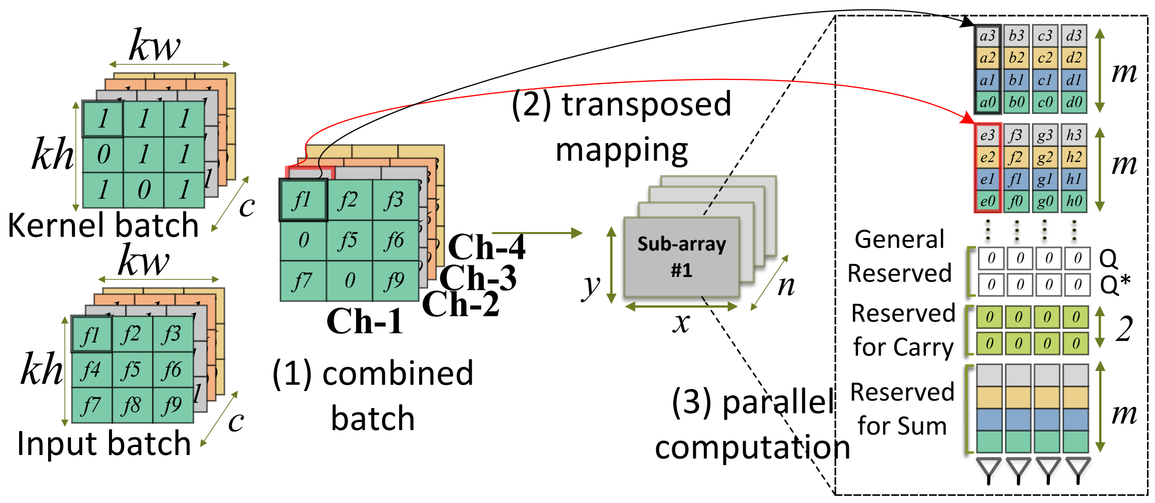


Figure 6. BWNN Hardware Mapping

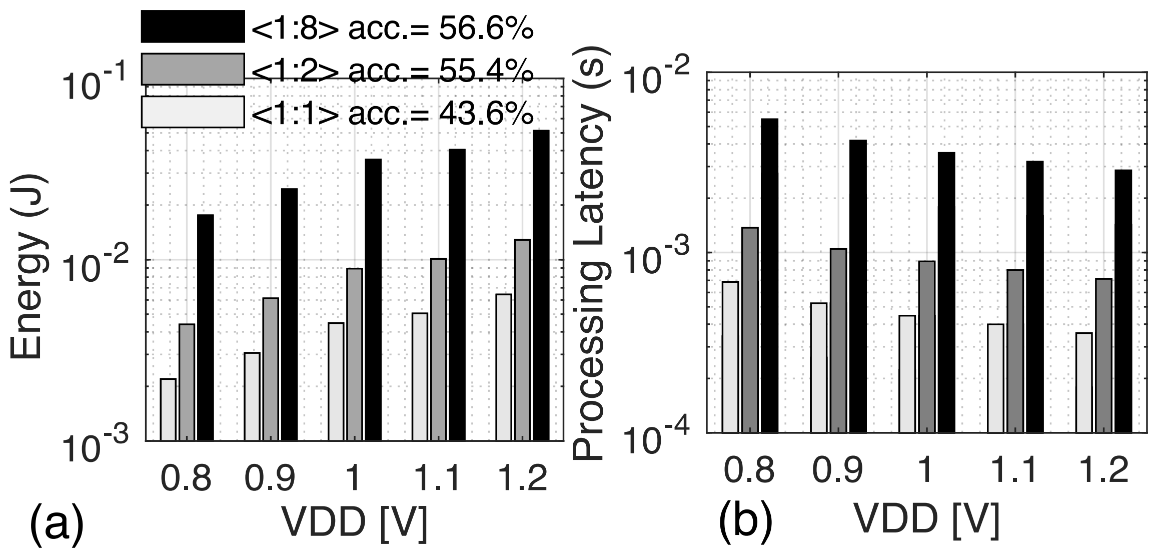


Figure 7. (A) PSRAM Energy Consumption and (B) Processing Time for Running the Alexnet (Imagenet Dataset).

Table 3. Comparison with State-of-the-art SRAM Based PIM Accelerators.

		BWNN Accelerators		Generic Accelerators	
Reference	PSRAM	A. Biswas et.al. (2018)	H. Valavi et al. (2019)	J. Yue et al. (2020)	Y. Zhang et al. (2018)
Technology	65nm	65nm	65nm	28nm	40nm
Bit cell Density	8T	10T	8T	8T Transposable	10T
Supply Voltage	0.8-1.2V	0.8-1.2V	0.68-1.2V	0.6 – 1.1V	0.5-0.9V
Max Frequency	1230MHz (1.2V)	5MHz	100MHz	475MHz (1.1V)	28.8MHz (0.7V)
SRAM Macro Size	2KB	2KB	4.8KB	16 KB	8KB
Performance (GOPS)	1259.52	8	295	32.7	14.7
Performance per unit area (GOPS/mm2)	583.12	126	23.4	27.3	70
Energy-Efficiency (TOPS/W)	34.98	40.3	20.6	5.27 (add) 0.55 (mult.)	31.28
Reconfigurable	Programmable	N/A	N/A	Programmable	N/A

constantly quantized to 1-bit and q-bit using the same method as Faraone *et al.* (2018), respectively, and then mapped to the parallel PSRAM sub-arrays. The top-1 accuracy after quantization on ImageNet dataset is reported in Fig. 7. For hardware mapping, considering n-activated PSRAM chips with the size of 128$\times$128 (Fig. 6), each sub-array can handle the parallel ADD/SUB (multiply-and-accumulate operations are converted to ADD/SUB in BWNNs) of up to 128 elements of m-bit ($2m \leq 128$) and so accelerator could process $n\times128$ elements simultaneously within computational sub-arrays to maximize the throughput. The memory sub-array data mapping for PSRAM is depicted in Fig. 6. We reserve four rows for Carry results initialized by zero and up to 32 rows for Sum results. Every pair of corresponding elements to be added together is aligned in the same bit-line. Herein, channel 1 (Ch1) and Ch2 should be aligned in the same sub-array. With m=32-bit, Ch1 elements occupy the first 32 rows of the sub-array followed by Ch2 in the next 32 rows. The addition algorithm starts bit-by-bit from the LSBs of the two words and continues towards MSBs. For evaluation, a 7-layer BWNN is adopted with distinct weight configurations of <W:I>: <1:1>, <1:2>, <1:8>. Our evaluation result reported in Fig. 7 shows that PSRAM

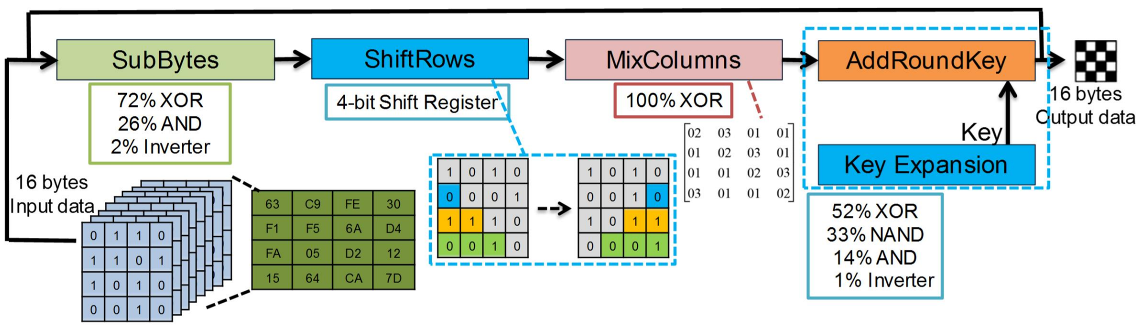


Figure 8. AES Block Diagram with the Gate Utilization

Table 4. 128-bit AES Performance.

Platforms	#Cycles	Freq. (MHz)	Time (μS) (Norm.)	Energy (nJ) (Norm.)
Y. Zhang et al.(2018)	6358	24	265 (1x)	64.2 (1x)
M. Hutter et al. (2011)	5429	0.847	6410 (24x)	10259 (160x)
PSRAM	718	1230	0.58 (0.002x)	19.21(0.3x)

can process AlexNet on average with 35 mJ energy per inference and ∼0.5 ms latency. The process energy and latency include the amount required by multiple PSRAM chips working as a whole entity. More detailed performance comparison with other recent SRAM based PIM designs are reported in Table 3.

2.4.3 Case study 3: Data Encryption

We further take the Advanced Encryption Standard (AES) data encryption algorithm as the third case-study. To facilitate working with input data (with a standard input length of 128 bits), each input byte data is distributed into 8-bit such that eight PSRAM sub-arrays are filled by 4×4 bit-matrices Mathew *et al.* (2010). After mapping, PSRAM supports the required AES bulk bit-wise operations to accelerate each transformations inside the memory. As shown in Fig. 8, all AES transformations are mainly based on (N)AND and XOR operations that are fully supported in PSRAM. In SubBytes, MixColumns, and AddRoundKey stages, parallel in-memory XOR2 and (N)AND2 operations contribute to more than 90% of the operations. In ShiftRows stage, state matrix will undergo a cyclical shift operation by a certain offset. We use the 128-bit AES software implementation as the baseline from Zhang *et al.* (2018), a 350nm ASIC Hutter *et al.* (2011), and a 40nm ASIC Zhang *et al.* (2018) designs for comparison. Table 4 shows that PSRAM achieves the highest speed-up over baseline. This mainly comes from the massively-parallel and high throughput XOR operation supported in PSRAM.

2.5 Conclusion

This chapter introduces a programmable PSRAM chip design in TSMC 65nm CMOS technology. For the first time, the PSRAM could execute a complete set of Boolean logic vector operations (i.e., NOR/NAND/XOR, both 2- and 3-input), majority, and full adder, all in a single memory cycle. The design is taped-out in 65nm CMOS and we demonstrate three case studies leveraging our PSRAM design on the prototype chip, including parallel vector operation, neural networks and data encryption.

Chapter 3

PRECISION SCALABLE IN-MEMORY COMPUTING MACRO WITH BIT-PARALLEL INPUTS AND DECOMPOSABLE WEIGHTS

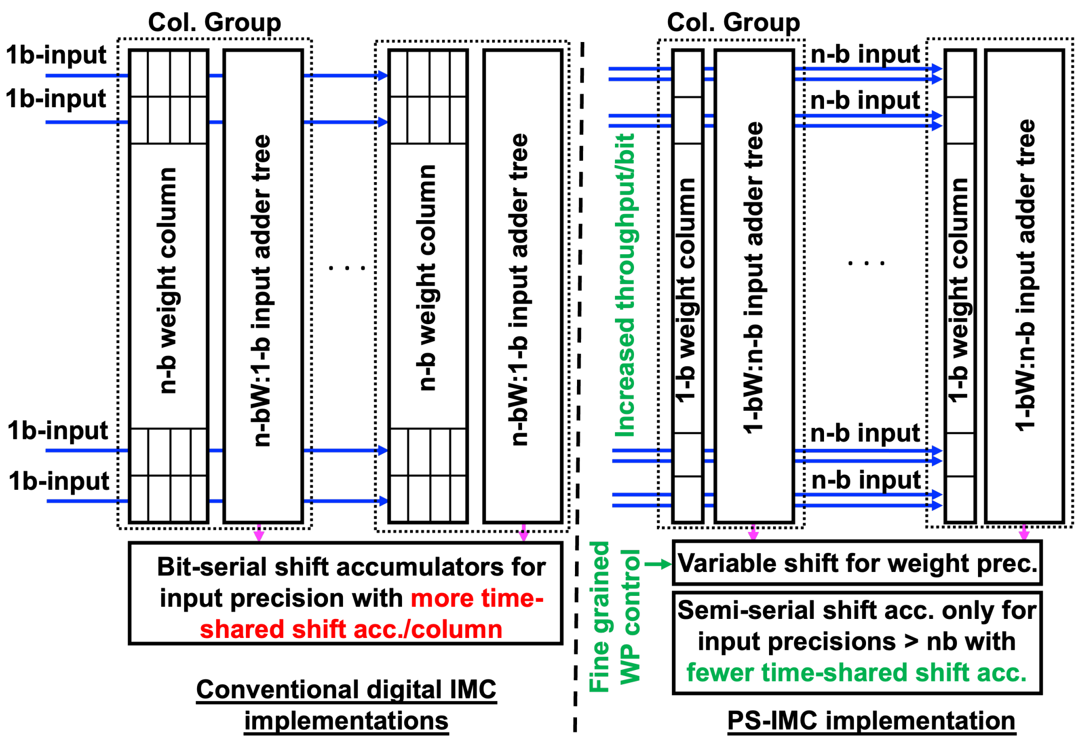
AI is currently the most dominant class of data-intensive applications, with large-dimensional matrix multiplications forming the core of AI computation. As AI models evolve rapidly in size, architecture, and numerical precision, there is a growing demand for adaptable hardware accelerators capable of efficiently handling this diversity. While generic compute approaches—such as those discussed in the previous chapter—offer flexibility, they often suffer from high overhead when executing matrix-matrix operations. This is primarily due to their reliance on multiple bit-serial instructions to complete a single matrix multiplication, resulting in significant latency overhead. A more efficient alternative is the use of SIMD-style instructions, which allow entire matrix multiplications to be executed with minimal cycle count and reduced instruction complexity, making them better suited for high-throughput AI workloads Dally (2023). To address this we present a fully digital multiply and accumulate (MAC) in-memory computing (IMC) macro demonstrating one of the fastest flexible precision integer based MACs to date. The design boasts a new bit-parallel architecture enabled by a 10T bit-cell capable of four AND operations and a decomposed precision data-flow that decreases the number of shift-accumulate operations, bringing down the overall adder hardware cost by 1.57x whilst maintaining 100% utilization for all supported precision. It also employs a carry save adder tree that saves 21% of adder hardware. The 28nm prototype chip achieves a speed-up of

2.6×, 10.8×, 2.42×, and 3.22× over prior SoTA in 1bW:1bI, 1bW:4bI, 4bW:4bI, and 8bW:8bI MACs respectively.

3.1 Introduction

In-memory computing (IMC) has been widely investigated as a promising approach to accelerate AI workloads. There are mainly two prevalent IMC design paradigms, i.e., compute in analog or digital domain. Analog IMC has gained attention due to the large number of operations it can perform per watt as well as per unit area. But it faces significant drawback from computing accuracy standpoint. On the other hand, digital IMCs Fujiwara *et al.* (2022b); Lee *et al.* (2022); Chih *et al.* (2021); Mori *et al.* (2023b); Lin *et al.* (2023) are more akin to digital ASICs closely interleaving memory and logic units. Many recent digital IMC works demonstrate high throughput and energy efficiency compared to their analog counterparts, without any accuracy drop due to robust rail-to-rail logic operations. Convolutions are at the heart of deep learning algorithms and are the most compute intensive operations. They follow a multi-dimensional compute pattern, where the weights and inputs have five dimensions (2D kernel, input channel, output channel, and bit-width).

Typical IMC designs follow a weight stationary approach by storing flattened 5D weights (W) in 2D-space. The 2D weight matrix is constructed by assigning output channels and bit-width to IMC rows (enabling parallel multiplications), the kernel dimensions and input channels to IMC columns (enabling parallel accumulations). The 5D-input feature maps (input/IP) are streamed from outside the memory onto the word-lines (WL) performing multiplications within the bit-cell. WLs being a limited resource (1-bit/cycle per WL in most cases), the 5D-input stream-in is time-



Col. Group
1b-input
1b-input
1b-input
1b-input
n-b weight column
n-bW:1-b input adder tree
n-b weight column
n-bW:1-b input adder tree
Bit-serial shift accumulators for input precision with more time-shared shift acc./column
Conventional digital IMC implementations
Increased throughput/bit
Fine grained WP control
Col. Group
n-b input
n-b input
n-b input
n-b input
1-b weight column
1-bW:n-b input adder tree
1-b weight column
1-bW:n-b input adder tree
Variable shift for weight prec.
Semi-serial shift acc. only for input precisions > nb with fewer time-shared shift acc.
PS-IMC implementation

Figure 9. Comparison of Conventional Digital IMC Implementations Versus Proposed Throughput Oriented PS-IMC Implementation

multiplexed to just 1D-input/unit time. In this setup, the input bit-width is also unrolled in the time domain, hence a larger number of time-shared shift accumulations circuits are present. The overall system incurs a large latency overhead due to the bit-wise stream-in of inputs (latency overhead = (input precision) × (# of inputs)). To maintain high throughput, large weight precision is typically addressed spatially by grouping several memory columns together. This in-turn reduces the flexibility to tune weight precision during inference. With the purpose of improving storage density, recent works Fujiwara *et al.* (2022b); Mori *et al.* (2023b); Oh *et al.* (2023b) attempted to time-share the compute hardware (adders, multipliers, etc.) with more memory cells at the cost of throughput. Considering that convolutions are compute bound, our approach is to maximize the throughput akin to Lee *et al.* (2022). However, Lee *et al.* (2022) requires weight and/or input replication to achieve full utilization for different precision MACs. Fig. 9 illustrates the PS-IMC design and its advantages over prior works. Implemented in 28nm CMOS, PS-IMC achieves the highest throughput for all supported MAC precision (1/4/8-b Weight (W):1-8b Input (I)), the highest energy efficiency for 1bW:1bI and 1bW:4bI MACs and the highest normalized compute density (TOPS/mm^2) for 8bW:8bI MACs.

3.2 PS-IMC Macro Architecture

Fig. 10 depicts the architecture diagram of the PS-IMC macro. We design the macro using a semi-bit-parallel architecture, i.e., the design is completely bit-parallel up to 4b-input, and a higher input precision will require time multiplexing (4b-input/unit time). There is a pipeline stage between the adder-tree and shifter to evenly distribute the critical path across two cycles. Hence, it takes two clock cycles to complete a MAC with a 4b-input and 4b-weight, three cycles for a MAC with 8b-input with 4b or 8b-weight, and only one clock cycle for a 4b-input and 1b-weight. We implement two 128×64 PS-IMC macros on the prototype chip and each 128×64 macro has two 64x64 sub-arrays stacked one on top of the other, with the variable shift accumulator (VSA) in the middle. Each column of the PS-IMC sub array has one 64-input 4-bit CSA tree and 64 10T-SRAM bit-cells.

3.2.1 Bit-Cell Design

Fig. 11(A) and Fig. 11(B) show the proposed 10T bit-cell schematic and layout. Each bit-cell occupies 1.38μm^2 and is designed using logic rules. Each of the four

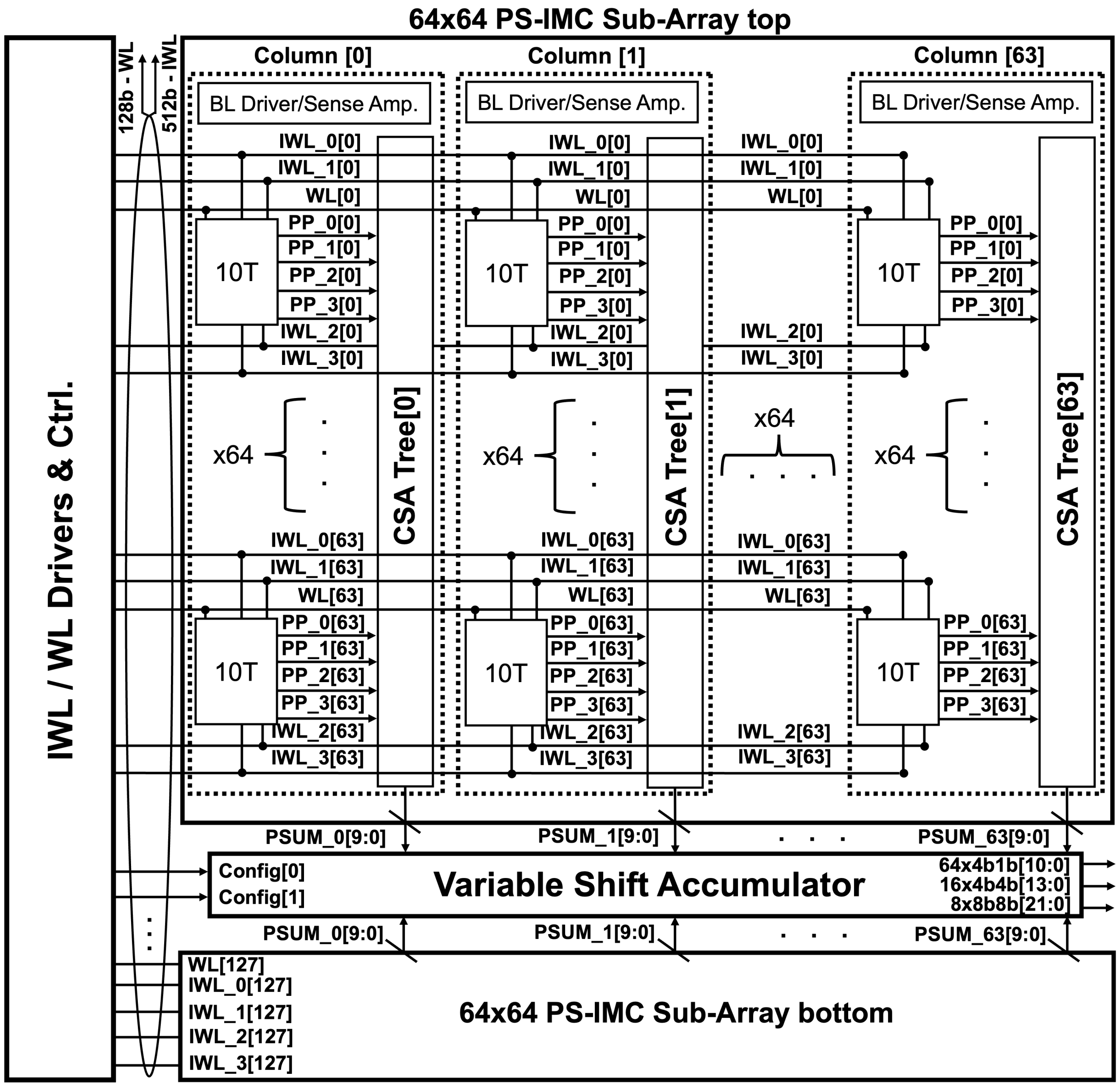
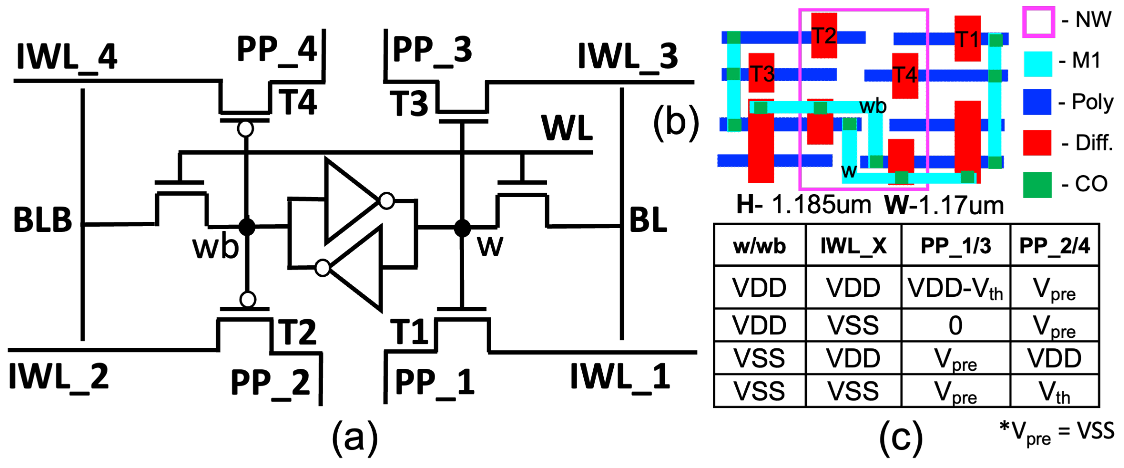


Figure 10. PS-IMC Macro Architecture Design. (Adapted from Sridharan *et al.* (2024a) with Permission)

additional transistors (T1-T4) perform a pass-gate based dot-product between 4b input streamed in through the input/IP word lines (IWLs) and the 1b weight and it's complement stored on either side of the cross-coupled inverters. Each transistor (T1-T4) is allocated to one input bit-significance, and performs a 1bW:1bI dot-product. Thus, each bit-cell as a whole can perform a 1bW:4bI dot-product. Prior to compute, the Partial Product (PP) nets are precharged to VSS and the IWLs are held at VSS



w/wb	IWL_X	PP_1/3	PP_2/4
VDD	VDD	VDD-V_{th}	V_{pre}
VDD	VSS	0	V_{pre}
VSS	VDD	V_{pre}	VDD
VSS	VSS	V_{pre}	V_{th}

*V_{pre} = VSS

Figure 11. (A) Bit-cell Schematic, (B) Layout, and (C) Truth Table.

until all the weight bits are written into the SRAM bit-cells. This avoids any erroneous compute when the stored bit is zero as the pass gates are controlled by the stored weights. Additionally, PP_X requires precharge after a large time interval of 0.5μs. This is dependent on the RC load on the PP_X. We determine this precharge interval based on post layout simulations. The functionality of the pass-gate based AND operation is shown in the truth table in Fig. 11(C).

3.2.2 Carry Save Adder (CSA) Tree

Fig. 12 compares the precision breakdown of all branches in a binary adder tree (BAT) widely used in prior works Fujiwara *et al.* (2022b); Chih *et al.* (2021); Mori *et al.* (2023b) and a CSA tree of similar configuration used in this work. CSAs have been widely adopted in digital designs requiring multi-operand additions because of the significant reduction in the number of full adders and the speed-up they provide. CSAs isolate each operand into partial sums and carries which are accumulated in

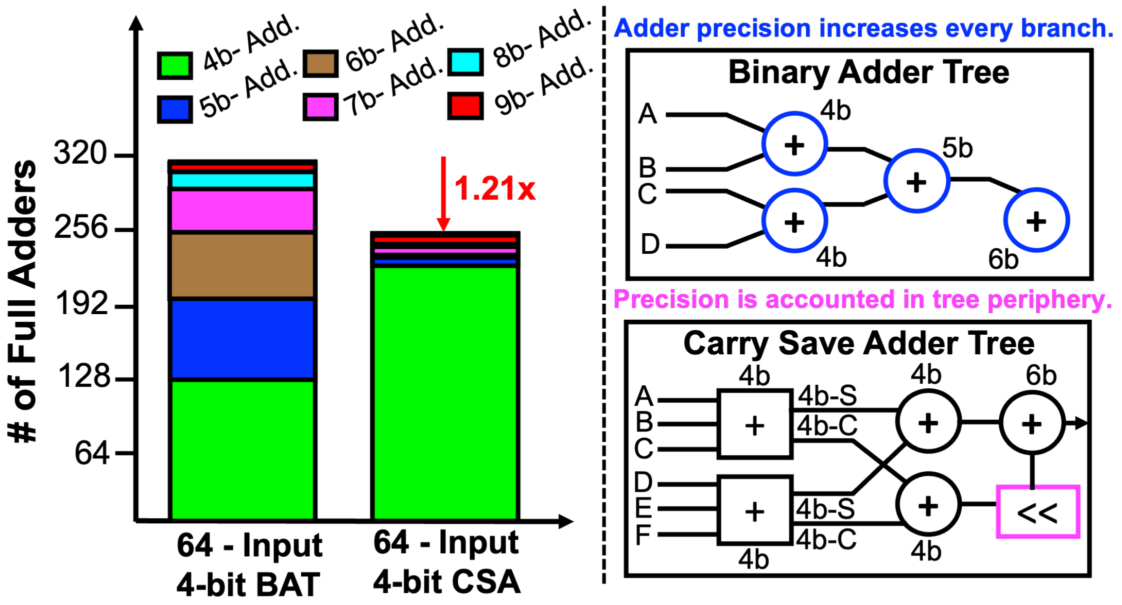


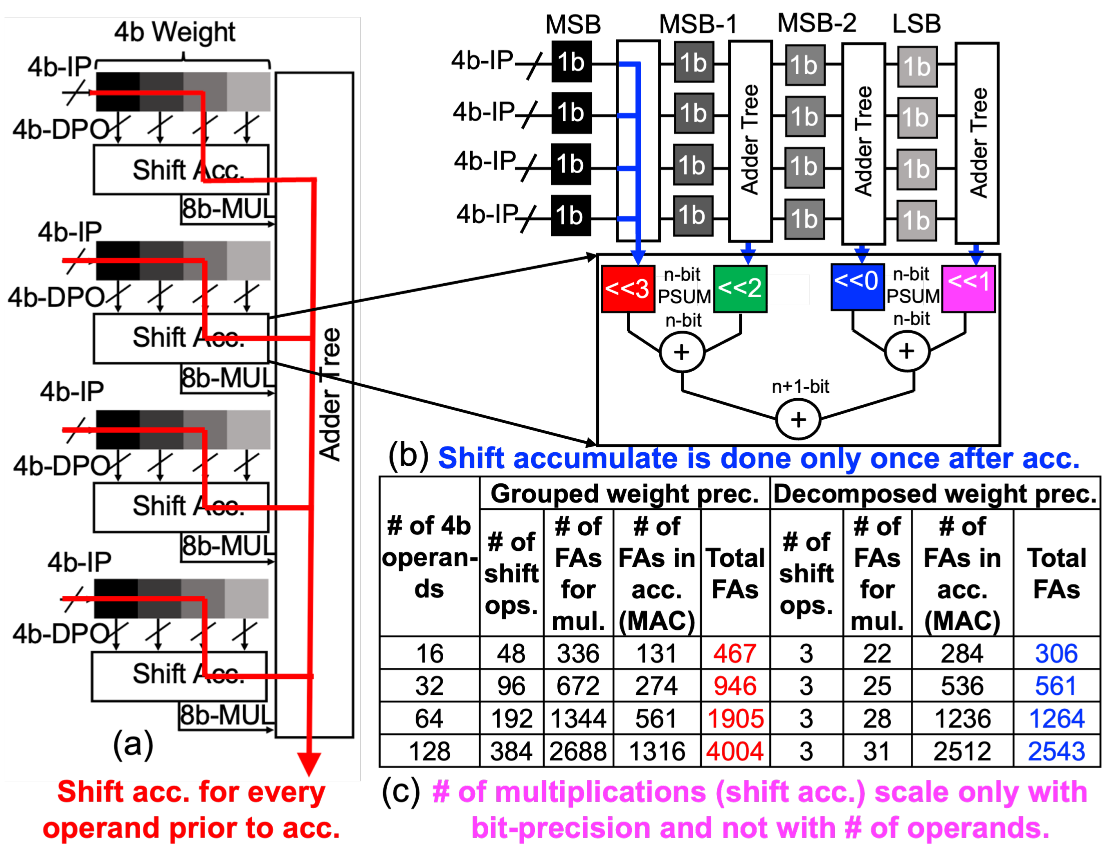
Figure 12. Precision Breakdown and Tree Structures of CSA and BAT

parallel and the bit-precision is accounted for in the tree periphery. This utilizes fewer high-precision adders and thereby results in an overall reduction in the number of full adders, also resulting in a shorter critical path delay. For a 64-operand 4-bit configuration, the CSA requires 21% fewer full adders when compared to the BAT counterpart.

3.2.3 Decomposed Weight Precision Data-flow

Prior IMC designs Fujiwara *et al.* (2022b); Chih *et al.* (2021); Mori *et al.* (2023b) spatially encode multi-bit weight precision by grouping memory columns together, limiting weight precision flexibility. Sharma *et al.* (2018) proposes a hierarchical approach to achieve this by decomposing large adder and multiplier precision into smaller blocks that can be selectively tiled together. In this implementation, MACs are performed in the traditional sense by completing multi-bit multiplication for each operand prior to accumulation. Given that different shift-add hardware is required to support flexible precision, Bit-Fusion trade-offs larger hardware overhead to support flexible precision. Ryu *et al.* (2022) overcomes this hardware cost by allocating a fixed bit-position to an entire PE (capable of small fixed-precision MACs across several operands). The support for flexible precision comes when fusing the PEs together, the partial sum from each PE is subject to a shift operation based on the allocated bit-position. However, this reduction in multiplication hardware comes at the cost of increased accumulation hardware, but given that multiplications are significantly more expensive to perform (a 4-bit multiplication requires 4 ANDs, 3 shifts, 2 4-bit additions and 1 5-b addition), the increase in accumulation hardware is easily off-set by reduced multiplication hardware. The lowest granularity of fixed precision MACs in each PE is 2-bit hence even Ryu *et al.* (2022) incurs a multiplication cost that scales with the number of operands.

With this as motivation, we design PS-IMC using a completely weight decomposed data-flow that un-groups all memory columns. Each memory column is allocated a weight bit-significance (for an n-b weight, the first column stores the MSB and the nth column stores the LSB). The 1bW:4bI partial product generated by each bit-cell across all rows are accumulated first. After accumulation, each column is subjected to a shift operation depending on the bit-position. For example, considering a 4b weight, PPs from the MSB column will go through left-shift by 3 (<<3) and as we move down the column the shift value decreases by 1 (Fig. 13(B)). By handling weight precision separately, the number of shift-accumulate operations remain constant regardless of the number of operands (Fig. 13(C)). Through this approach, considering a 4-bit precision and 128 operands, the total full-adder (FA) cost (multiplication +


4b Weight
4b-IP
4b-DPO
Shift Acc.
8b-MUL
Adder Tree
(a)
Shift acc. for every operand prior to acc.
MSB
MSB-1
MSB-2
LSB
4b-IP
1b
Adder Tree
<<3
<<2
<<0
<<1
n-bit PSUM
n-bit
n+1-bit
+
(b) Shift accumulate is done only once after acc.
(c) # of multiplications (shift acc.) scale only with bit-precision and not with # of operands.

# of 4b operan-ds	Grouped weight prec.				Decomposed weight prec.			
	# of shift ops.	# of FAs for mul.	# of FAs in acc. (MAC)	Total FAs	# of shift ops.	# of FAs for mul.	# of FAs in acc. (MAC)	Total FAs
16	48	336	131	467	3	22	284	306
32	96	672	274	946	3	25	536	561
64	192	1344	561	1905	3	28	1236	1264
128	384	2688	1316	4004	3	31	2512	2543

Figure 13. (A) Grouped Weight Precision, (B) PS-IMC's Decomposed Weight Precision, (C) Multiply and Accumulate Hardware Cost for 4-bit Operands

accumulation) is reduced by 1.57× and the number of shifts performed can be reduced by 128×.

3.2.4 Variable Shift Accumulator (VSA) for Precision Handling

PS-IMC supports a wide variety of MAC configurations (1b/4b/8b Weight and 1-8b Input). Fig. 14(b) shows the micro-architecture of the VSA. A VSA is padded to every 8 columns to support a maximum of 8b weight precision. Config[1:0] signals in the VSA control the bit-precision of the weights and inputs. The 1bW:4bI MACs

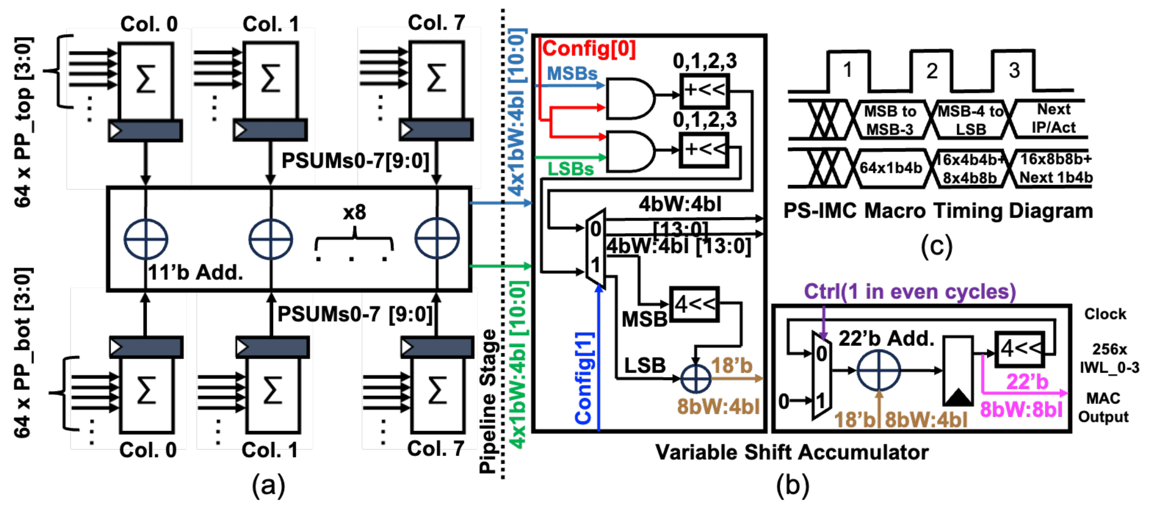


Figure 14. (A) Psum Reduction from Top and Bottom Sub-arrays, (B) Variable Shift Accumulator Micro-architecture, (C) Timing Diagram with the Total Number of Macs Performed in a Single PS-IMC Macro

are collected in the pipeline stage before VSA in the same cycle as the input stream in. These MACs are then gated by Config[0], which enables column-specific shifts (<<3 for the 1st column and no shift for the last column) to obtain 4bW:4bI MACs. Every 8 columns generate two 4bW:4bI MACs of 14-bit precision and these MAC outputs are further gated by Config[1] to selectively shift accumulate one of the 4bW:4bI MAC output to obtain a larger 8bW:4bI MAC of 18-bit precision. An alternative approach to implement a large weight precision (>4b) would be to subject each column to a variable shift (7b through 4b), but in this case, the un-selected shifters remain idle and each IMC column incurs a multiplexer overhead as opposed to only one de-multiplexer for every 4 IMC columns. Due to the bit-parallel nature of the 10T bit-cell, input precision of up to 4b can be handled in a single cycle without any time-multiplexing or special control. Only for input precision above 4b, multiplications are time-shared through shift-accumulators (Fig. 14(b)). The VSA is tailored to support input precision ranging from 1-nb for a n-bit W. As a result, the a

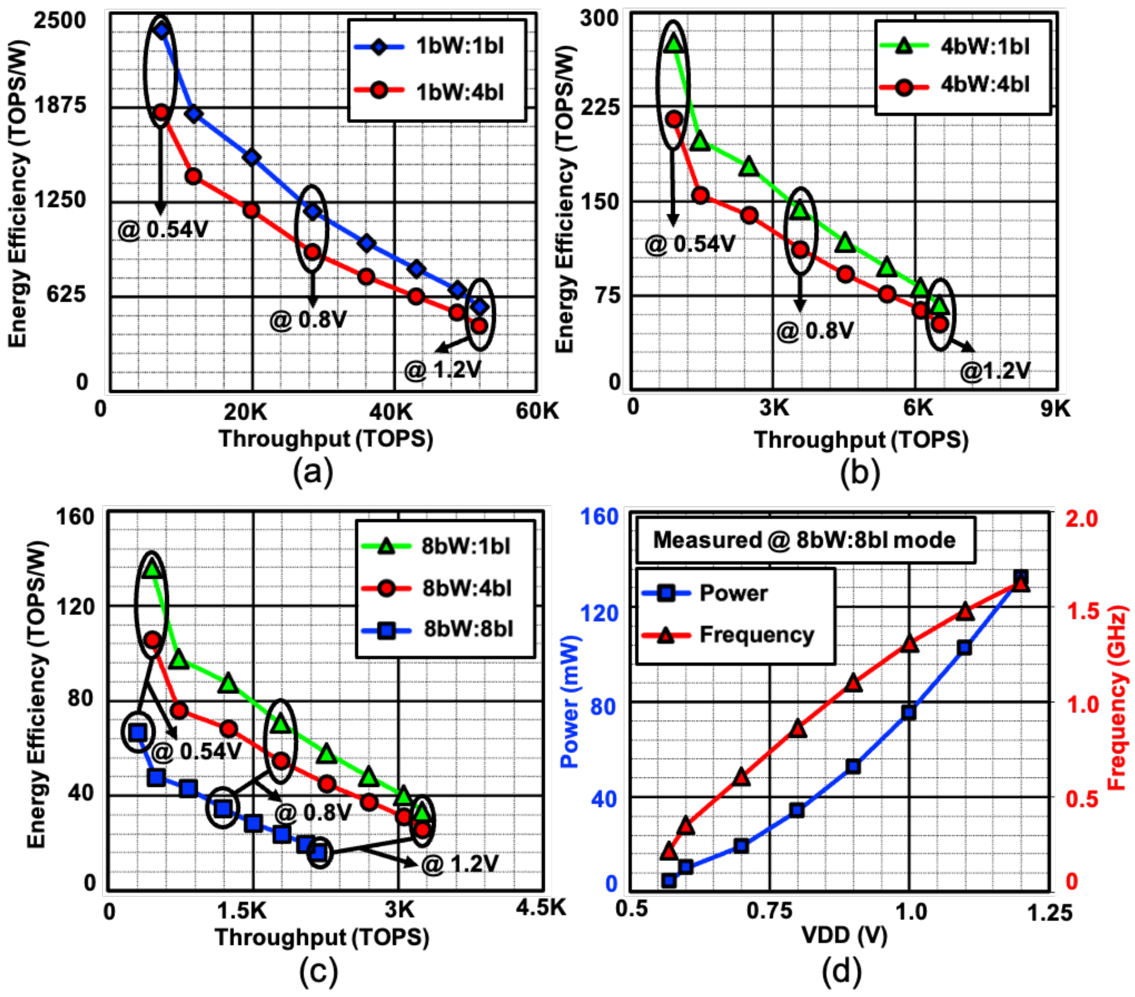


Figure 15. Throughput (TOPS) vs. Energy-efficiency (TOPS/W) for (a) 1bW:1/4bI, (B) 4bW:1/4bI, and (C) 8bW:1/4/8bI Modes. (D) Power and Frequency Scaling

time-shared shift-accumulator is only necessary once every n columns. This reduces the shift-accumulators/column since input precision is accounted with little (4b-8b) to no (1b-4b) time-multiplexing. PS-IMC macro has three pipeline stages. The first, second and third pipeline stages generate 64 4bI:1bW MACs, 16 4bI:4bW MACs and 8 8bI:8bW MACs respectively, as illustrated in Fig. 14(c) and (d).

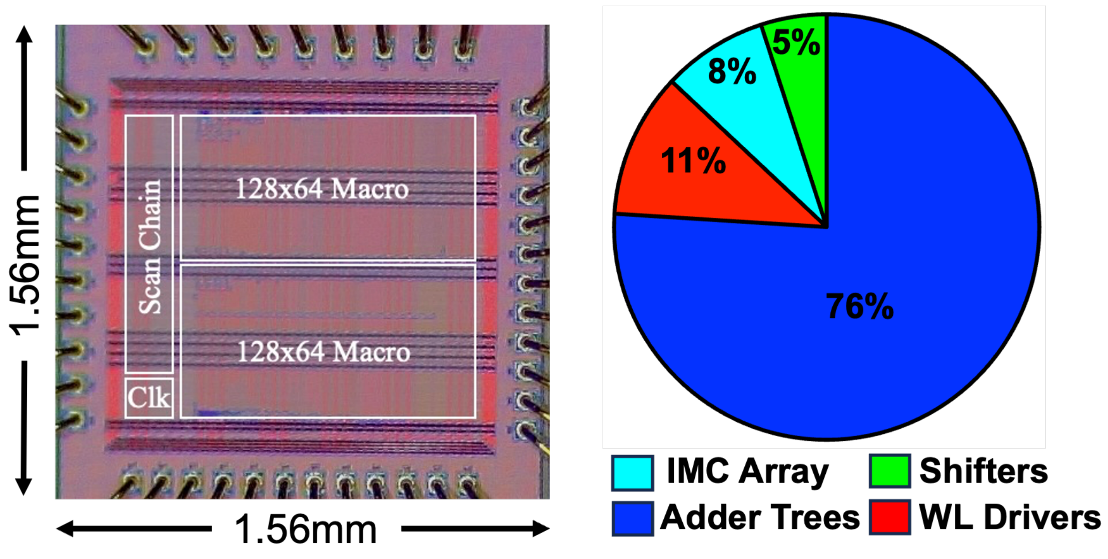
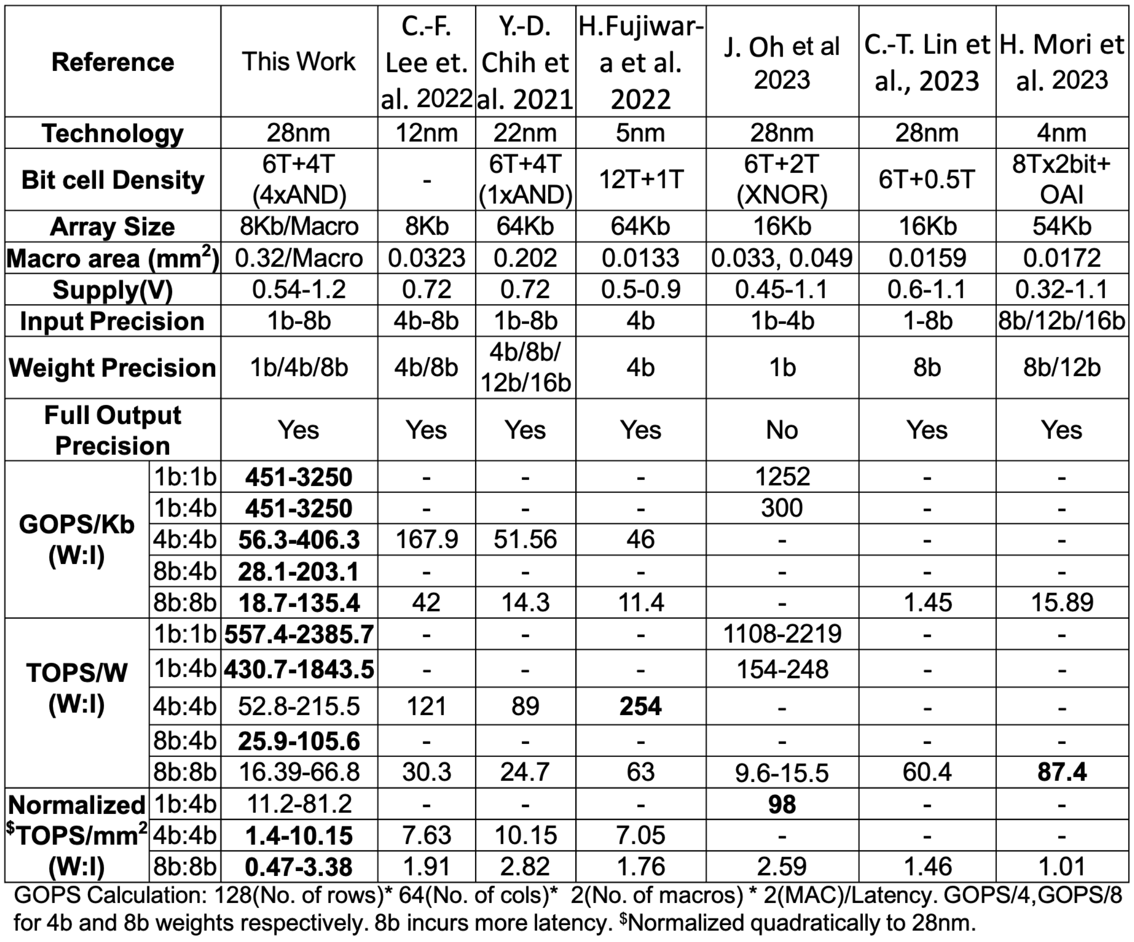


Figure 16. Die Micrograph (Left), and Area Breakdown (Right)

3.3 Chip Measurement Results

PS-IMC is prototyped in TSMC 28nm CMOS. We implement two 128×64 macros on the prototype chip, where each macro occupies $0.32mm^2$. The PS-IMC macro achieves the highest peak throughput/kb compared to all prior digital IMC works for 1bW:1-4bI (3.25 TOPS/Kb @ 1.2V), 4bW:1-4bI (406.3 GOPS/Kb @ 1.2V) and 8bW:4-8bI (135.4 GOPS/Kb @ 1.2V). It also achieves the highest energy efficiency for 1bW:4bI (1843 TOPS/W @ 0.56V) and 1bW:1bI (2385.7 TOPS/W @ 0.56V). The measurement condition is with 50% bit-wise weight sparsity and an average of 25% toggle rate for the inputs at 27^oC. Input toggle rate has a linear dependence on power, where 25% decrease in toggling rate will yield about 8.7-13% increase in energy efficiency. For a lower input precision (<4b), we disable the IWLs depending on the precision (a 2bI would mean we disable IWL_3 and IWL_2). A 1bW:1bI MAC in PS-IMC achieves the same throughput as a 1bW:4bI, but due to the reduction in the number of active IWLs 1bW:1bI achieves a higher energy efficiency. Fig. 15(a), Fig.

Table 5. Comparison with Prior Digital IMC Works



Reference		This Work	C.-F. Lee et. al. 2022	Y.-D. Chih et al. 2021	H.Fujiwar-a et al. 2022	J. Oh et al 2023	C.-T. Lin et al., 2023	H. Mori et al. 2023
Technology		28nm	12nm	22nm	5nm	28nm	28nm	4nm
Bit cell Density		6T+4T (4xAND)	-	6T+4T (1xAND)	12T+1T	6T+2T (XNOR)	6T+0.5T	8Tx2bit+ OAI
Array Size		8Kb/Macro	8Kb	64Kb	64Kb	16Kb	16Kb	54Kb
Macro area (mm^2)		0.32/Macro	0.0323	0.202	0.0133	0.033, 0.049	0.0159	0.0172
Supply(V)		0.54-1.2	0.72	0.72	0.5-0.9	0.45-1.1	0.6-1.1	0.32-1.1
Input Precision		1b-8b	4b-8b	1b-8b	4b	1b-4b	1-8b	8b/12b/16b
Weight Precision		1b/4b/8b	4b/8b	4b/8b/ 12b/16b	4b	1b	8b	8b/12b
Full Output Precision		Yes	Yes	Yes	Yes	No	Yes	Yes
GOPS/Kb (W:I)	1b:1b	**451-3250**	-	-	-	1252	-	-
	1b:4b	**451-3250**	-	-	-	300	-	-
	4b:4b	**56.3-406.3**	167.9	51.56	46	-	-	-
	8b:4b	**28.1-203.1**	-	-	-	-	-	-
	8b:8b	**18.7-135.4**	42	14.3	11.4	-	1.45	15.89
TOPS/W (W:I)	1b:1b	**557.4-2385.7**	-	-	-	1108-2219	-	-
	1b:4b	**430.7-1843.5**	-	-	-	154-248	-	-
	4b:4b	52.8-215.5	121	89	**254**	-	-	-
	8b:4b	**25.9-105.6**	-	-	-	-	-	-
	8b:8b	16.39-66.8	30.3	24.7	63	9.6-15.5	60.4	**87.4**
Normalized $TOPS/mm^2 (W:I)	1b:4b	11.2-81.2	-	-	-	**98**	-	-
	4b:4b	**1.4-10.15**	7.63	10.15	7.05	-	-	-
	8b:8b	**0.47-3.38**	1.91	2.82	1.76	2.59	1.46	1.01

GOPS Calculation: 128(No. of rows)* 64(No. of cols)* 2(No. of macros) * 2(MAC)/Latency. GOPS/4,GOPS/8 for 4b and 8b weights respectively. 8b incurs more latency. $Normalized quadratically to 28nm.

15(b) and Fig. 15(c) illustrate the difference in energy efficiency as input precision is scaled with fixed weight precision. In addition, Fig. 15(c) shows the 33% latency cost when the input precision is scaled above 4b.

Table 12 compares PS-IMC against state-of-the-art digital SRAM IMC designs. It achieves throughput improvements of 2.6×, 10.8×, 2.42×, 3.22× in 1bW:1bI, 1bW:4bI, 4bW:4bI and 8bW:8bI MACs respectively. By reducing multiplication and adder hardware (decomposed weight precision and CSA trees) and by increasing input bit-parallelism, PS-IMC achieves 1.1× and 7.4× improvements in TOPS/W for 1bW:1bI and 1bW:4bI MACs respectively. It also achieves 1.2× improvement in normalized compute density for 8bW:8bI MACs. Fig. 16 shows the PS-IMC prototype chip and area breakdown.

3.4 Conclusion

In this work, we present a throughput oriented IMC macro that has a unique decomposed weight precision data-flow for flexible precision bit-parallel MACs. PS-IMC maintains 100% utilization without weight replication with low hardware overhead. Measurement results show that PS-IMC achieves the highest throughput, energy efficiency and compute density for various MAC workloads compared to prior SoTA IMC works.

Chapter 4

A FULLY DIGITAL SPARSE IN-MEMORY MATRIX VECTOR MULTIPLIER FOR COMMUNICATION APPLICATIONS

Channel decoders are key computing modules in wired/wireless communication systems. Recently DNN based decoders have shown their promising error-correcting performance because of their end-to-end learning capability. However, compared with the traditional approaches, the emerging neural belief propagation (NBP) solution suffers higher storage and computational complexity, limiting its hardware performance. This stems from the fact that NBP has sparse matrix multiplications at its core. The previous chapter demonstrates that IMCs are very good at performing matrix multiplications, this is because IMCs have a matrix-like physical structure and is easy to map matrix compute onto it. But sparse matrix multiplications break the matrix structure if the zero compute needs to be skipped and makes it difficult to carry them out in IMCs. To address this challenge, we develop compute methodologies that enable sparse compute in IMCs. We do this by first analyzing the unique sparsity patterns in the NBP algorithm, and then propose new IMC friendly compression algorithm that enables zero skipping and also propose two new IMC designs that adapt this algorithm and perform fast and efficient structured and unstructured sparse matrix multiplications required by NBP. To our knowledge this is one of the first works to propose IMC methodology for NBP compute. Post implementation, we perform extensive experiments to demonstrate that our proposed design achieves significantly higher energy efficiency and throughput compared to sate-of-the-art counterparts both from a standalone IMC macro and a complete system level NBP implementation.

4.1 Introduction

Thanks to their powerful error-correcting capabilities, modern channel codes, such as low-density parity check (LDPC) Gallager (1962), polar Arikan (2009), and Turbo Berrou *et al.* (1993) codes, have been widely used in numerous real-world wired and wireless communication systems, including but not limited to 5G, Wi-Fi, StarLink, Ethernet, etc. In general, given a fixed channel code, its error-correcting performance is mainly determined by the *decoder*. Recently, neural belief propagation (NBP), as a neural network (NN)-based approach, has shown very promising decoding performance across different types of channel codes Nachmani *et al.* (2016); Cammerer *et al.* (2017); Gruber *et al.* (2017); Lugosch and Gross (2017). By unfolding the original iterative belief propagation procedure to form a sparse feedforward neural network, NBP makes the key scaling parameters, which were previously set in a heuristic way, can now be directly learned from the data, significantly improving the error-correcting capability of channel codes.

Hardware Challenge of NBP Decoder. Despite its attractive algorithmic advantage, NBP decoder is facing a severe challenge in hardware performance. The integration of NN into decoding process, though improving error-correcting performance, brings much higher storage and computation overhead. Because channel decoders are typically deployed in the real-time and/or low-power communication systems, the significantly increasing complexity, if not properly addressed, may hinder the widespread adoption of this promising technique.

IMC-NBP: A Double-Win Solution. Fortunately, we discover that the emerging hardware challenge for NBP decoder can be effectively addressed via in-memory computing (IMC), a technique that has been well-studied to develop low-power general NN hardware Biswas *et al.* (2018); Yue *et al.* (2020); Wang *et al.* (2022); Sridharan *et al.* (2022a). Considering NBP is essentially a type of specialized sparse feedforward neural network model, applying IMC to its hardware design, is naturally a very promising strategy towards achieving high hardware performance while preserving high decoding performance.

Which Type of IMC for NBP? Motivated by such promising benefits, in this paper, we propose to develop energy-efficient high-performance in-memory computing-based neural BP decoder. Since there exist numerous types of IMC techniques in the market, e.g., SRAM, RRAM, MRAM, etc., the very first design knob we need to consider is the most suitable IMC approach for NBP decoder. Our in-depth analysis concludes that digital in-SRAM computing is the best candidate for building the desired NBP decoder. This is because compared to AI applications, wired/wireless communication have very stringent requirement on error rate (at least 10^{-4} and above) and data rate, calling for a noiseless compute environment and high throughput. To that end, digital in-SRAM computing is an ideal candidate for NBP hardware because of its accurate computation, low read/write latency, and high flexibility.

Questions to be Answered. Considering NBP decoder is a sparse neural network with a unique sparsity pattern and activation function, a customized solution, instead of the existing general in-SRAM hardware, is desired to fully deliver its algorithmic promise. More specifically, several technical questions need to be answered. For instance, how should we properly leverage the unique structured and unstructured sparse patterns, which currently cannot be supported by the existing digital SRAM-based IMC implementations? What is the efficient way to map new computing flow and operation on the IMC circuits?

Technical Preview and Contributions. In this work, we perform systematic investigations to answer these questions, and then develop the corresponding hardware solutions. The main contributions are summarized as follows:

1. We, for the first time, design and develop an end-to-end, energy-efficient high-speed SRAM-based in-memory computing system for neural BP channel decoding, namely *DSPIMM*.
2. We propose an efficient and digital bit-serial in-memory matrix-vector multiplication (MVM) module using a novel 8T compute SRAM bit-cell circuit design, fully supporting the unique sparsity pattern in NBP decoding.
3. We propose a greedy weight compression and localization (GWCL) algorithm, which properly leverages the structured and unstructured sparsity pattern, to realize efficient data mapping and sparse computing.
4. We conduct extensive experiments showing the great energy efficiency and power improvement of our DSPIMM platform. We also systematically benchmark with other state-of-the-art counterparts.

4.2 Neural BP Algorithm

According to coding theory, a (N, K) channel code is uniquely defined by a $(N-K)$-by-N binary parity check matrix ($\boldsymbol{H}$), which can also be interpreted as a bipartite graph consisting of N variable nodes and $(N-K)$ check nodes. Suppose we use v to denote the v-th variable node in the node set V and c to denote the c-th check node in the node set C, respectively. Also, we use $E = \{e_{(c,v)} = (c, v) : H(c, v) = 1, v \in V, c \in C\}$ to denote the set of edges connecting the two types of nodes. Here the $e_{(c,v)}$ connecting

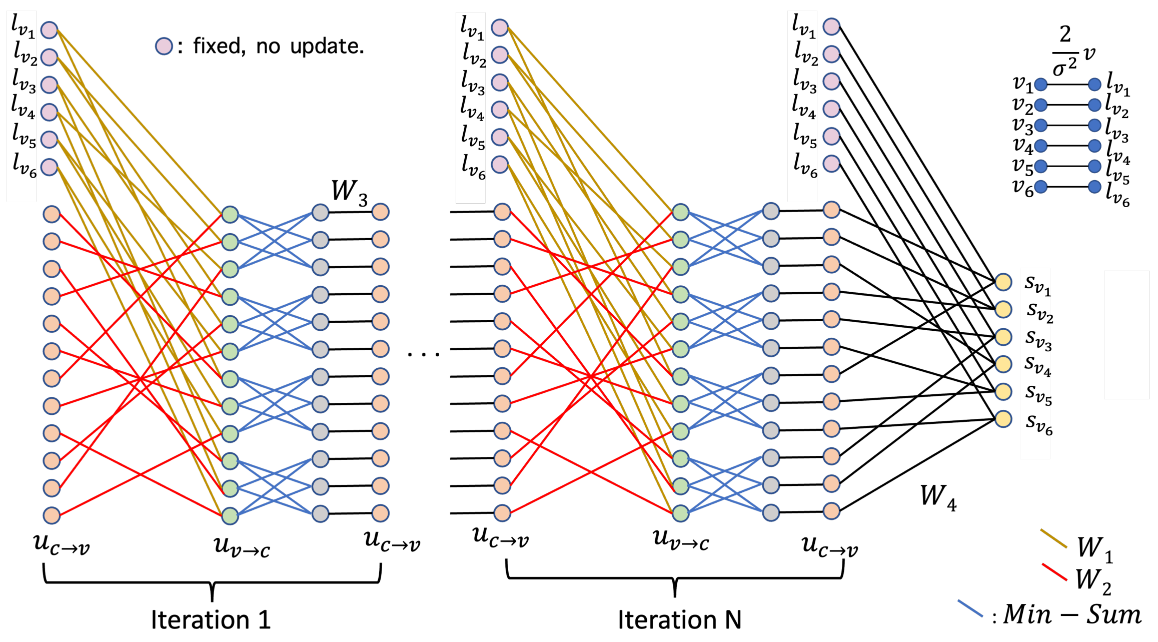


Figure 17. Example of Neural-BP Decoding Procedure

the c-th check node and the v-th variable node corresponds to one 1-valued entry ("$H(c, v) = 1$") of $\boldsymbol{H}$.

The key idea of NBP decoding Nachmani *et al.* (2018) is to perform message update in the unfolded bipartite graph. As illustrated in Fig. 17, the neurons denoted as orange and green circles represent $u_{c\to v}$ and $u_{v\to c}$, which are the messages (i.e., "belief") transmitted from the v-th variable node to the c-th check node and from the v-th variable node to the c-th check node at the t-th iteration through edges E, respectively. Different from traditional belief propagation, NBP treats the connections between $u_{c\to v}$ and $u_{v\to c}$ as trainable weights instead of the pre-set heuristics. Next, we summarize the overall dataflow and compute steps of Neural BP decoding. Initially, an NBP decoder receives the log-likelihood ratios (LLRs) $\boldsymbol{l} \in \mathbb{R}^n$ of the received codeword $\boldsymbol{r}$ as:

$$l_v = \log \frac{Pr(x_v = 1|r_v)}{Pr(x_v = 0|r_v)}. \tag{4.1}$$

Then the variable nodes and check nodes iteratively update the LLR messages during the entire Neural BP decoding process. The specific update principle of the LLR message in each iteration go through the following five steps:

Step 1: Structured Sparse Matrix-Vector Multiplication (SSP-MVM). At the t-th iteration, $u^t_{v\to c}$ can be calculated as:

$$\boldsymbol{u^t_{v\to c}} = \boldsymbol{W_1 l_v} + \boldsymbol{W_2 u^{t-1}_{c\to v}}, \tag{4.2}$$

where we define the first term as $\boldsymbol{k^t_{v\to c}} = \boldsymbol{W_1 l_v}$ and second term $\boldsymbol{q^t_{v\to c}} = \boldsymbol{W_2 u^{t-1}_{c\to v}}$. For the first term $\boldsymbol{k^t_{v\to c}} = \boldsymbol{W_1 l_v}$, the matrix format can be formulated as:

$$\begin{bmatrix} (k^t_{v\to c})_1 \\ \vdots \\ (k^t_{v\to c})_D \end{bmatrix} = \begin{bmatrix} 0,0,0,w,\dots \\ 0,0,w,0,\dots \\ \vdots \end{bmatrix} \begin{bmatrix} (l_v)_1 \\ \vdots \\ (l_v)_N \end{bmatrix} \tag{4.3}$$

with input vector $\boldsymbol{l_v} \in \mathbb{R}^N$ and weight matrix $\boldsymbol{W}_1 \in \mathbb{R}^{D\times N}$ that has one non-zero entry in each row (corresponding to v), denoted by golden connections between l_v (pink circles) and $u_{v\to c}$ (green circles).

Step 2: Unstructured Sparse Matrix Vector Multiplication (USP-MVM) and Accumulation. It requires another matrix multiplication followed by an addition with the results from the previous step. For the second term $\boldsymbol{q^t_{v\to c}} = \boldsymbol{W_2 u^{t-1}_{c\to v}}$, the matrix format can be formulated as:

$$\begin{bmatrix} (q^t_{v\to c})_1 \\ \vdots \\ (q^t_{v\to c})_D \end{bmatrix} = \begin{bmatrix} 0,0,0,0,\dots \\ 0,0,0,w,\dots \\ \vdots \end{bmatrix} \begin{bmatrix} (u^{t-1}_{c\to v})_1 \\ \vdots \\ (u^{t-1}_{c\to v})_D \end{bmatrix} \tag{4.4}$$

with input vector $\boldsymbol{u}^{t-1}_{\boldsymbol{c\to v}} \in \mathbb{R}^D$ and weight matrix $\boldsymbol{W}_2 \in \mathbb{R}^{D\times D}$ that has non-zero entries at the positions corresponding to $N(v)\backslash c$, where $N(v) = \{c \in C : e_{(c,v)} \in E\}$ and $M(c) = \{v \in V : e_{(c,v)} \in E\}$ are the neighbors of variable node v and check node c, respectively. $\boldsymbol{W}_2$ is denoted by the red connections between $u_{c\to v}$ (orange circles) and $u_{v\to c}$ (green circles). The $\boldsymbol{u}^t_{\boldsymbol{v\to c}}$ is calculated as the summation of $\boldsymbol{k}$ and $\boldsymbol{q}$ from Eq. 4.3 and 4.4 as:

$$\begin{bmatrix} (u^t_{v\to c})_1 \\ \vdots \\ (u^t_{v\to c})_D \end{bmatrix} = \begin{bmatrix} (k^t_{v\to c})_1 \\ \vdots \\ (k^t_{v\to c})_D \end{bmatrix} + \begin{bmatrix} (q^t_{v\to c})_1 \\ \vdots \\ (q^t_{v\to c})_D \end{bmatrix}. \tag{4.5}$$

Step 3&4: Min-Sum and Dot-Product Computation. The $u^t_{c\to v}$ is calculated by min-sum operation Lugosch and Gross (2017) as follows:

$$u^t_{c\to v} = w_{3_{c\to v}} \min_{v'\in M(c)\backslash v} |u^t_{v'\to c}| \prod_{v'\in M(c)\backslash v} \operatorname{sign}(u^t_{v'\to c}), \tag{4.6}$$

where $\boldsymbol{w_3} \in \mathbb{R}^D$ is learnable. This step is denoted by connections between $u_{v\to c}$ (green circles) and $u_{c\to v}$ (orange circles) including blue connections (min-sum operations) and black connections (weight dot products).

Step 5: S_V Calculation. Then, the final soft output after the t-th iteration can be calculated as:

$$\boldsymbol{s}^t_{\boldsymbol{v}} = \boldsymbol{l_v} + \boldsymbol{W}_4 \times \boldsymbol{u}^t_{\boldsymbol{c'\to v}} \tag{4.7}$$

$$\begin{bmatrix} (s^t_v)_1 \\ \vdots \\ (s^t_v)_N \end{bmatrix} = \begin{bmatrix} (l_v)_1 \\ \vdots \\ (l_v)_N \end{bmatrix} + \begin{bmatrix} 0,0,0,0,\dots \\ 0,w,0,0,\dots \\ \vdots \end{bmatrix} \begin{bmatrix} (u^t_{v'\to c})_1 \\ \vdots \\ (u^t_{v'\to c})_D \end{bmatrix}, \tag{4.8}$$

where $\boldsymbol{W}_4 \in \mathbb{R}^{N\times D}$ with non-zero elements correspond to $N(v)$, denoted by golden connections between $u_{c\to v}$ (orange circles) and s_v (yellow circles).

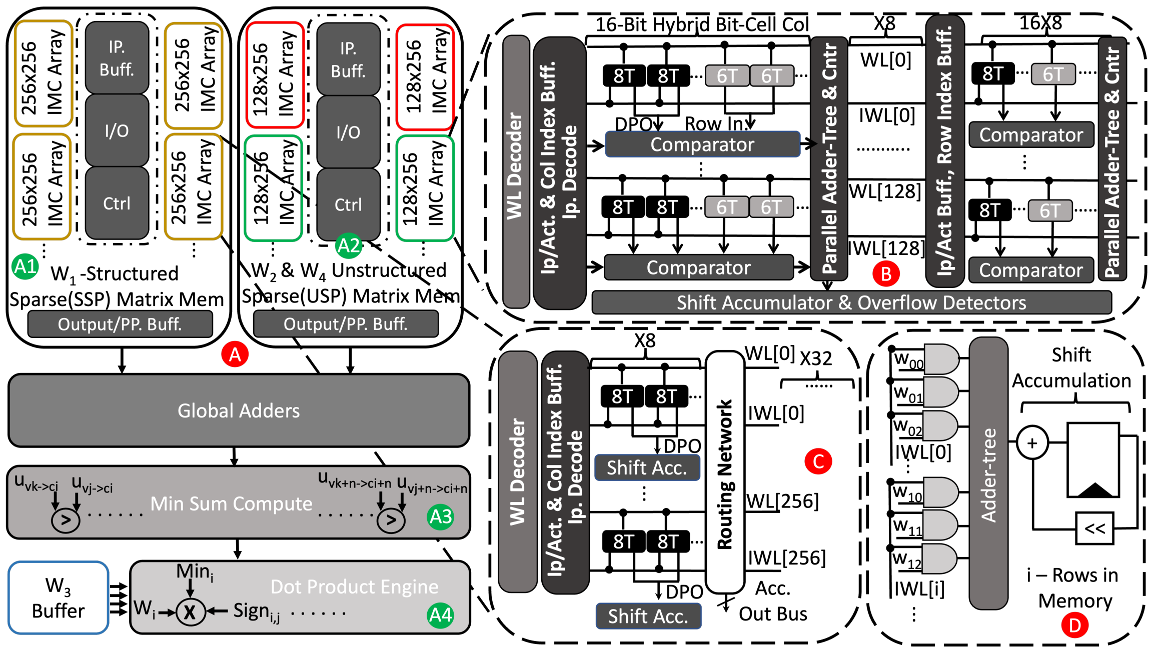


Figure 18. (A) DSPIMM Architecture (B) Unstructured Sparse Weight In-memory Compute (C) Structured Sparse Weight In-memory Compute (D) Data Flow of Bit-serial MVM. (Adapted from Sridharan *et al.* (2023) with Permission)

Remark. As shown in the neural BP computation flow (Fig. 17), the majority of learn-able weights $\boldsymbol{W_1}, \boldsymbol{W_2}, \boldsymbol{W_4}$ hold **high sparsity** with a special pattern under the matrix format. For instance, because $\boldsymbol{W}_1 \in \mathbb{R}^{D\times N}$ has only one non-zero weight in each row, the sparsity of $\boldsymbol{W_1}$ is $\frac{N-1}{N}\%$, which can easily achieve 90% when $N \geq 10$ and 99% when $N \geq 100$.

4.3 Proposed *DSPIMM* Platform

4.3.1 Architecture and Data Flow

Fig. 18(A) demonstrates the overall architecture of our DSPIMM for NBP. It supports all the required five algorithm steps as shown in the corresponding circuit model in Fig. 18(A1), (A2), (A3), and (A4). Note that, steps 1 and 2 are mainly MVMs and are implemented using our IMC modules, where the corresponding circuits are shown in Fig. 18(B) and (C).

4.3.1.1 W_1, Structured Sparse Matrix Memory

denoted by Fig. 18(A1). This corresponds to the NBP step 1. The IMC array with golden halos denotes the memory of W_1 weight matrices (i.e., golden connections in Fig. 17). Each IMC array size is 256x256 which translates to 8KB per block. The Control, I/O logic, and Input Buffers are shared among 4 IMC sub-arrays. The control and I/O logic help in scattering the stored weights in the Input Buffer (compiler-generated SRAMs) to the IMC sub-Arrays. They also store the Inputs, i.e., l_v vectors, and scatter them during compute mode. Then, the compute/partial product outputs are collected in the output/partial product buffers (Using SRAM compiler generated Register Files).

4.3.1.2 W_2, Unstructured Sparse (USP) Matrix Memory

Fig. 18(A2) shows the USP-Matrix Memory. The memory organization is similar to SSP Matrix memory, with 128x256 - 4KB IMC arrays. They carry W_2 (red halo) as well as W_4 (green halo) since both follow a similar computing pattern. This USP Matrix Memory is responsible for the sparse MVMs in Step 2 and Step 5 of the NBP algorithm.

4.3.1.3 Global Addition

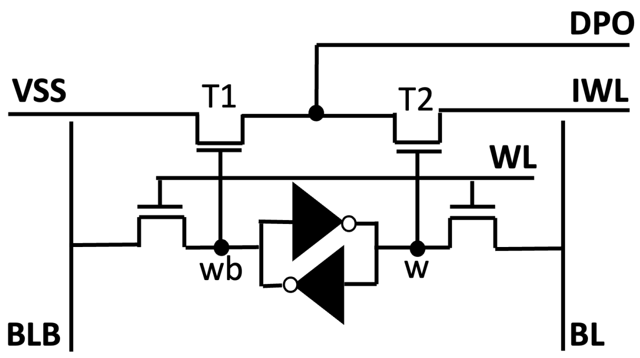
After SSP and USP-MVM, the stored partial products in the output buffers are streamed into parallel global adders to perform the addition operations on the two MVM outputs. This completes the compute of the green dots in Fig. 17 denoting the end of Step 2 of NBP algorithm.

4.3.1.4 MinSum Compute and Dot Products

These modules perform steps 3 and 4. The $Minsum$ and dot-product instructions do not have common operands, hence are not suitable for IMC. Thus, we leverage the digital comparators in parallel to compute the $MinSum$. The $Minsum$ outs along with the sign bits from the output buffers of USP-MM and W_3 weights are sent to the dot-product engine (DPE) to compute Step 4. The W_3 weight matrix is usually large and uncompressed. So a Register file is used to buffer a portion of W_3 weights to be streamed onto the DPE. when finished, the remaining data will be fetched from off-chip.

4.3.1.5 S_v Calculation

Step 5 needs to be performed only once after several iterations of Steps 1 through 4. It is performed by loading l_v onto the output buffers of the SSP-MVM memory and using the computational sub-array of the USM-MVM (green halos) to perform MVM between W_4 and the u_vc of the previous iteration. Now, the global adders are used to



IWL	w	wb	DPO
VSS	VSS	VDD	VSS
VSS	VDD	VSS	VSS
VDD	VSS	VDD	VSS
VDD	VDD	VSS	VDD-Vth

Figure 19. (A) 6+2T(8T) Compute Bit-cell (B) Truth Table of 2T And/Dot-product

sum the output buffers of USP Matrix Memory and SSP Matrix Memory containing the MVM outs and l_v respectively.

4.3.2 6+2T (8T) SRAM Bitcell Design for In-memory-computing

To implement in-memory computing (IMC), specifically for matrix multiplication in this work, we propose a 6+2T SRAM compute bitcell (CBC) as shown in Fig. 19(a) to implement 1 bit partial product and then the peripheral shift & accumulator circuits implement the rest for multi-bit matrix multiplication. For memory function, a traditional 6T SRAM bitcell is used. For compute, the bit-cell is augmented with two additional transistors - T1 and T2. Together, they perform the 'AND' function or a 1'b dot-product within the memory cell, between operand-1 (weight bit - w/wb) and operand-2 (external input bit - IWL). The weight bit (w and its complementary - wb) is stored in the cross-coupled inverters of bitcell which are connected with the gate terminals of T2 and T1, respectively. The other operand-2 is from the input world line (i.e., IWL), which goes to the source terminal of T2. Note that, as the name suggests, IWL is broadcasted to the entire worldline, providing inputs to all 8T CBCs in that row that store multi-bits of the weight parameters. The last signal is a VSS/GND connecting to the source terminal of T1. Finally, the AND or 1'b Dot product out (DPO) is obtained from the common drain terminals of T1 and T2, where the truth table is given in Fig. 19(b)

4.3.3 Bit-Serial Matrix Vector Multiplication (MVM) In-memory

For multi-bit MVM, the multi-bit weight operand is stored in the memory and the other input operand is streamed through the IWLs. The matrix operand stored in the memory is transposed before storage, this will put a single row of the matrix elements into a single column of the memory array. It is done to (1) perform dot products between all elements in one column of the second operand (through IWL) and all elements in a single row of the first operand. (2) Since IWLs are shared amongst rows, the same second operand column can be used to multiply with all the first operand rows, performing a parallel NXM (Op.1) * $MX1$ (Op.2) *vector dot-product*. Then, the accumulation of dot-products of every column in memory (corresponding to the row of the first operand) is implemented using adder trees to complete MVM. Due to bit-serial design, the IWLs can stream only 1 bit at a time, a shift accumulation is designed to respect the bit-position of the multi-bit operands streamed through the IWLs, where data flow is shown in Fig. 18(D).

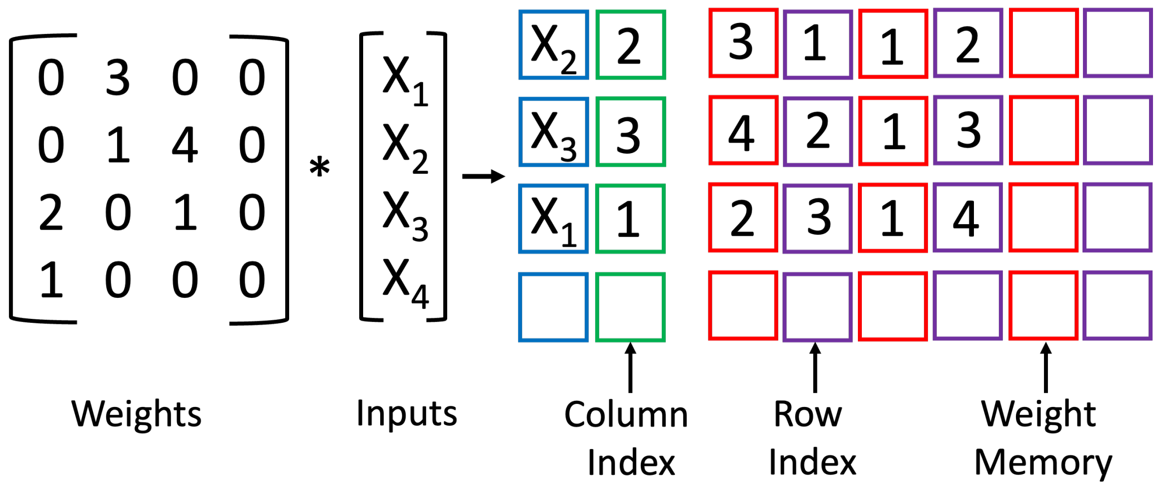


Figure 20. EGWCL Algorithm Example

4.3.4 Structured Sparse Matrix Vector Multiplication (SSP-MVM)

MVM-in-memory is dense and intensive, meaning all bit-cells are active and used for compute. But for NBP, the W_1 is extremely sparse with the special pattern as described in section II. So to leverage such property, we develop a hardware-friendly compression/encoding algorithm that localizes the weights and eliminates all zeros from being stored, thereby ignored for compute.

4.3.4.1 Greedy Weight Compression and Localization (GWCL) Algorithm for SSP-MVM

Fig.20 shows an example. It parses through the weight matrix and only stores the non-zero weights in memory. But, such an operation scatters the weights across memory, breaking the structure of matrix multiplication. To solve this, during the skip of zero weights, our circuit needs to be aware of (1) which input needs to be multiplied with which weight, and (2) which DPOs (post multiplication) need to be added together.

For (1), whenever the algorithm faces a non-zero weight, it stores not only the weight value but also the column index of this weight next to the input buffers. Through this, when a new input is streamed in, the column indices can dictate which row of input buffer should be streamed onto the IWLs for compute. Since the memory leverages the sharing of IWLs over several bit-cells to achieve high parallelism, during compression, all weights belonging to the same column are stored in a single row. For (2), the sparsity of W_1 matrix is structured and only one element per row is a non-zero weight. Since all elements in a row are added up for MVM(after dot product), the resulting accumulation in this case will be the dot-product of the input and weight alone. Therefore, due to the nature of this sparsity pattern, no accumulation is necessary. Hence, no circuitry is required for accumulation or decoding the accumulation of the scattered weights. In summary, the GWCL algorithm works in two stages:

Stage 1: Ignores zero-weights and greedily stores non-zero weights, it also stores the corresponding column index alongside the input buffer.

Stage 2: If encountering a weight belonging to a column previously stored, it stores the weight in the same row as that of the previously encountered column index to enable the parallel multiplication for a shared IWL.

4.3.4.2 SSP-MVM In-Memory Compute Circuit and Architecture

The one-time sparse weight compression discussed above is done off-line and mapped to our IMC arrays. During inference, a new set of inputs is fetched every iteration, so a decoding circuitry is designed to map the newly fetched inputs using column indices. It consists of a set of comparators that compare the indices of the new inputs against the stored column indices (next to the input buffers) and map the inputs to the input buffers of the respective rows. Then, these inputs will be streamed onto the IWLs in a bit-serial fashion for performing partial product. Since no accumulation is required, the DPOs are directly sent to the shift accumulators which completes the 8b8b dot-product. In summary, the implemented SSP-MVM IMC architecture has 32 8-bit columns, each column has 256 rows and each row of the 8-bit column consists of a shift accumulator and each 8-bit column have a routing network to route all the accumulated outputs.

4.3.5 Unstructured Sparse Matrix Vector Mult. (USP-MVM)

The above SSP-MVM has a fixed sparse pattern with one-hot element in a row, enabling us to skip accumulation. But, the unstructured sparse W_2 and W_4 matrices do not follow this pattern, with multiple non-zero elements in a row.

4.3.5.1 Enhanced Greedy Algorithm for USP-MVM

To adapt our hardware for USP-MVM, we enhance our GWCL algorithm to also support USP weights. The main difference here is that weights need to be accumulated and are scattered all over the memory array. To complete MVM, it needs a way to identify which DPOs (post input stream-in) require accumulation, hence an additional operation is performed alongside Stage 1, which is, the row indices of the weights are also stored alongside the weight memory. The reason is that, in an MVM between matrix A and B, the column of operand B is multiplied by the row of operand A, after which the dot-products accumulate together. Mapping such a process to our IMC memory array means the accumulation only happens to the dot-products of weights in the same row. So, by storing the weight indices during accumulation, we only need to accumulate the dot-products resulting from weights having the same row-indices. In summary, the Enhanced GWCL (EGWCL) is:

Stage 1: Store column indices of all non-zero weights next to the input buffers; store the row-indices next to the non-zero weights, ignoring the zero weights.

Stage 2: If the newly encountered weight has a column index that is previously stored next to the input buffers, it stores the weight in the subsequent column of the same memory row corresponding to the column index.

4.3.5.2 USP-MVM IMC Circuit and Architecture

As per our EGWCL algorithm, both the weights and row indices are stored in the weight memory. Since only the weights are used for compute and the row-indices are used for decoding the compressed weights for accumulation, a traditional 6T SRAM bitcell is used for storing the row indices and 8T CBC is used for storing weights. For a given memory size $m \times n$, $log(m)$ bits are required to represent the row indices. We use 128-bit rows memory with 256-bit columns. The 8-bit CBC and the 8-bit 6T SRAM together form a 16-Bit Hybrid Bit-Cell Column. So, we have 16 columns of the 16-Bit Hybrid Bit-Cells in total. As for the inputs, the column indices are stored alongside the input buffers in flops. When a new input is fetched, the comparator-based decoder is designed to parse through the column indices of the input matrix and store the corresponding inputs onto the input buffers. Then, they are streamed onto IWLs for dot product computing. The CBCs in every 16-bit column will hold 8-bit DPOs that will be accumulated next. The EGWCL algorithm scatters the weights across the memory, so multiple rows of the weight matrix can be present in a single column of the memory array. Circuitry is required to (1) identify which rows are present and (2) parse through and accumulate all the weights in the column. For (1), we attach comparators to every word, which enables reading of the weight indices directly from 6T bitcells. These comparators take in the row index as well as the 8'DPO and compare the row-index against a generated index. If it compares it outputs the 8'DPO, else it outputs a 0. For (2), we need to identify all the row-indices present in a single column. So a Mod-Counter is placed in every column. Every counter is given the first and last index present in the corresponding column to parse through all the row indices of that column. The output of these counters is sent to the comparators, providing indices to compare against and identify the weights needs to be accumulated.

4.3.5.3 Overflows

The row indices of the accumulated DPOs are also stored alongside the flops in the shift accumulators (Propagated from counters in the adder tree). This is to tackle an inherent drawback called overflow that arises due to the nature of the EGWCL algorithm. When the weights are compressed, there is a chance that weights from a single row (W_2 matrix) can span multiple columns (memory). In this case, these values need to be accumulated. So, to keep track of which row the partial product is being computed, the row index is also stored in flops for every column of the memory array. After the counters parse through all the row-indices, the overflow detectors

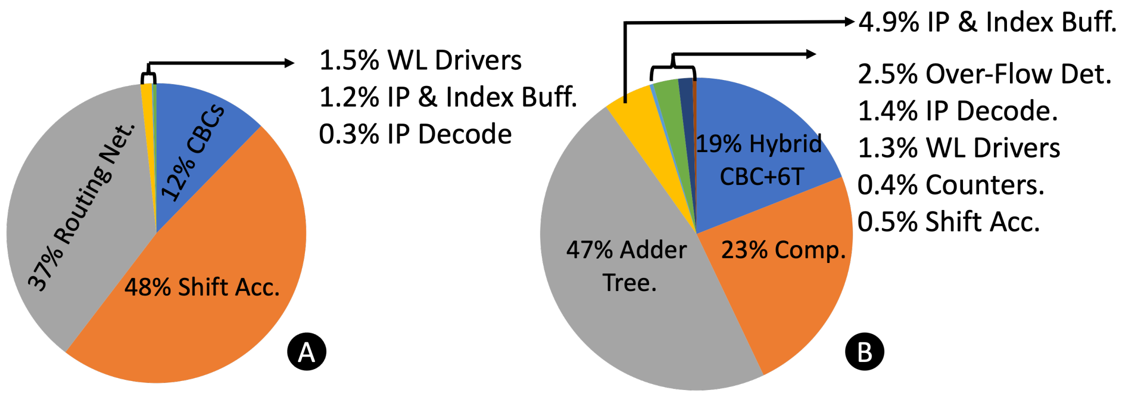


Figure 21. Area Breakdown of (A) SSP Matrix Memory (B) USP Matrix Memory

present alongside the shift accumulators accumulate all the weights that belong to the same row. This completes the USP-MVM.

4.4 Evaluation and Results

4.4.1 Experiment Setup

Cadence Spectre is used for all custom circuitry, designed using TSMC 28nm to verify functionality and to check for latency and power consumption. The area evaluation of custom circuitry is done by making layouts in Cadence Virtuoso. For an SSP-MVM Memory, we simulate a 128x256 memory array. For USP-MVM, we simulate a 256x256 array. For all digital components, we use Synopsys Design Compiler to synthesize the gate netlist. For all reported code lengths, VCD files are generated using SDF annotated post-synthesized RTL simulations. These VCD files are used in Synopsys PrimePower for reporting the power numbers. The Post-synthesized netlist is used in Synopsys PrimeTime to obtain latency numbers.

Table 6. Power Breakdown

SSP-Matrix Mem(256x256)		USP-Matrix Mem(128x256)	
Hardware	Power (mW)	Hardware	Power(mW)
Bit-Cell array(8T)	11.6mW	Bit-cell array(6T+8T)	4.2mW
Shift Accumulator	46.73mW	Comparator	21.3mW
Routing Network	R+C Parasitics	Adder Tree	18.7mW
IP Index+IP Buff.	6.3mW	Ip Index + Ip Buff.	4.22mW
Decoder	0.88mW	Shift Accumulator	6.74mW
Ip Decode	1.3mW	Overflow + Counters	4.63mW
Total	66.81mW	Total	59.79mW

4.4.2 Experiment Results

Since this is the first work to demonstrate NBP in an SRAM-based IMC, we compare our work with other popular LDPC channel decoding hardware implementations in Table 9, even using different algorithms. The reason is that LDPC is the most commonly used channel code in real-world applications, and its hardware decoder design receives the most attention as compared to other channel codes. Ours achieves the best energy efficiency and lowest power. We also evaluate the efficacy of our compression algorithm in Table 7. It clearly achieves memory savings that match the sparsity ratio.

For IMC performance, we draw comparisons with state-of-art IMC designs that have MAC operation as their core in Table 8. Compared to existing SRAM-based IMC platforms, the USP-MVM module and the SSP-MVM module achieve the best TOP/s metric. The USP-MVM achieves a throughput almost equal to Chih *et al.* (2021) even though it is only 1/4th in size and our SSP-MVM IMC module can complete an 8b8b MAC one cycle faster from skipping accumulations. The complete area and power breakdown for all sub-modules in USP-MVM and SSP-MVM are shown in Fig. 21

Table 7. GWCL Algorithm Memory Benefits(Excludes Index Memory)

Code Length/	121		672		1056	
Weight Memory	Uncompressed	GWCL algorithm	Uncompressed	GWCL algorithm	Uncompressed	GWCL algorithm
W_1	73.2KB	0.6KB	1.5MB	2.2KB	3.7MB	3.52KB
W_2	366KB	2.4KB	5MB	5.5KB	12.3MB	8.7KB
W_3	366KB	N/A	4.9MB	N/A	12.3MB	N/A
W_4	73.2KB	0.6KB	1.5MB	2.2KB	3.7MB	3.52KB

Table 8. Comparison with State-of-the-art SRAM Based IMC Accelerators

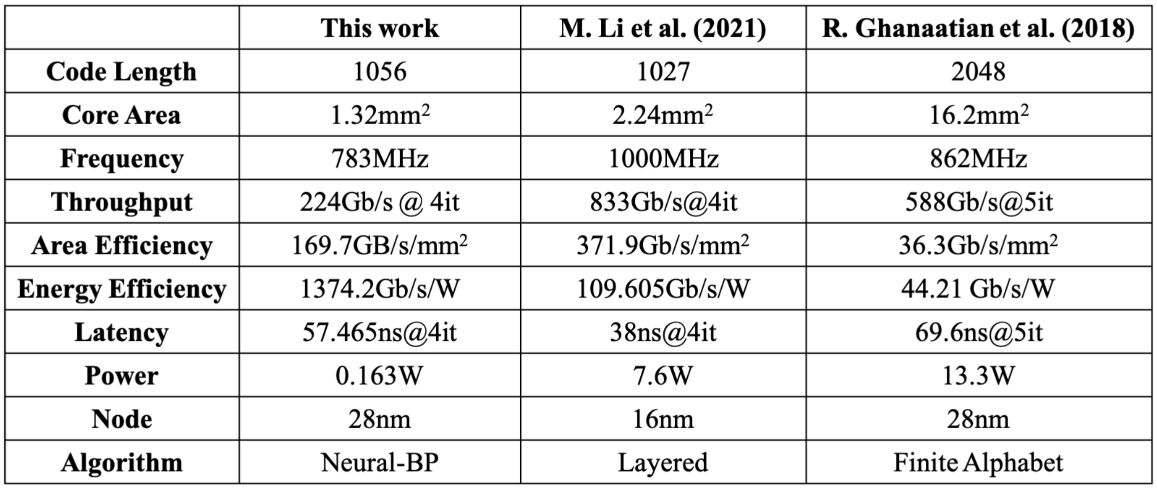
Reference	**USP-MVM (This work)**	**SSP-MVM (This work)**	**Y.-D. Chih et al (2021)**	**H. Kim et al. (2019)**	**D. Wang et al. (2022)**
Technology	28nm	28nm	22nm	65nm	28nm
Array Size	2+2 KB (W+I)	8KB	8KB	0.8KB	2KB
Bit-cell overhead	1T per bit-cell	2T per bit-cell	4T per bit-cell	XOR+MUX+FA	2T per bit-cell
Sparsity Level	50%	>99% fixed	50%	50%	50%
Macro Size	0.187mm^2	0.7673mm^2	0.202mm^2	0.242mm^2	0.049mm^2
Performance (GOPs) (8b8b)	786.18	1927.3	917	N/A	2035(4b1b)
Efficiency (TOPS/w) (8b8b)	12.92	29.56	24.7	2.06	154
Latency(8b8b)	10.42ns (9 cycles)	8.501(8 cycles)	18ns	NA	~20ns
Implementation	Synthesis	Synthesis	Post-Silicon	Post-Silicon	Post-Silicon

and Table 6. All above detailed hardware evaluation and bench-marking show great performance improvement and hopefully, our design could serve as a benchmark for future neural decoder implementations.

4.5 Conclusion

In this work, we propose a novel SRAM-based IMC circuit and architecture to implement the Neural BP channel decoding algorithm. We utilize the sparse nature of the algorithm by proposing IMC algorithm-hardware co-design to perform sparse MVMs whose operands have fixed (algorithm specific) or generic unstructured sparse patterns. Our proposed IMCs achieve the best throughput out of state-of-the-art IMC

Table 9. Comparison with Prior LDPC Implementations



	This work	**M. Li et al. (2021)**	**R. Ghanaatian et al. (2018)**
Code Length	1056	1027	2048
Core Area	1.32mm^2	2.24mm^2	16.2mm^2
Frequency	783MHz	1000MHz	862MHz
Throughput	224Gb/s @ 4it	833Gb/s@4it	588Gb/s@5it
Area Efficiency	169.7GB/s/mm^2	371.9Gb/s/mm^2	36.3Gb/s/mm^2
Energy Efficiency	1374.2Gb/s/W	109.605Gb/s/W	44.21 Gb/s/W
Latency	57.465ns@4it	38ns@4it	69.6ns@5it
Power	0.163W	7.6W	13.3W
Node	28nm	16nm	28nm
Algorithm	Neural-BP	Layered	Finite Alphabet

MAC implementations and significantly higher energy efficiency than state-of-the-art LDPC decoder hardware.

Chapter 5

A SPARSITY AWARE IN-MEMORY-COMPUTING MACRO WITH CONFIGURABLE SPARSE REPRESENTATIONS

Deep neural networks (DNNs) have achieved remarkable success across a wide range of cognitive tasks, prompting efforts to deploy them on edge devices. However, edge platforms are inherently constrained in compute power and memory capacity, making it challenging to accommodate large DNN models—which are both compute- and memory-intensive due to their reliance on multiply-and-accumulate (MAC) operations. To address this, extensive research has focused on improving model efficiency through various compression techniques such as quantization and sparsification. Model sparsification is achieved via weight pruning, where less significant weights are set to zero to reduce model complexity—often with an impact on accuracy. Numerous studies have explored the tradeoff between hardware support for sparsity and maintaining acceptable model accuracy Meng *et al.* (2021); Zhou *et al.* (2021).

These studies conclude that unstructured sparsity typically offers the highest model accuracy, but poses significant challenges for hardware implementation due to its irregular and non-deterministic pruning patterns. In contrast, structured sparsity—while more hardware-friendly—often results in a loss of accuracy. Furthermore, the optimal sparsity ratio varies not only across different DNN models but also between layers within the same model, making uniform sparsity support impractical for achieving ideal performance.

So, from a hardware perspective, it is essential to support a wide range of sparsity formats to meet the evolving demands of modern DNN models. Building on the previous chapter, this chapter explores how hardware can be generalized to accommodate diverse sparsity formats and ratios. We introduce a Sparsity-Aware In-Memory Computing (SP-IMC) macro that, for the first time, supports multiple compression schemes—including run-length encoding and coordinate (COO) format—as well as varying sparsity precisions such as $N{:}M$ formats, enabling real-time adjustment of the sparsity ratio. Additionally, SP-IMC incorporates scalable integer precision as proposed in Sridharan *et al.* (2024a). Fabricated in a 28nm CMOS process, our SP-IMC prototype demonstrates a reduction in compute resources by 4–50× compared to state-of-the-art IMC designs.

5.1 Introduction

5.1.1 Background and Motivation

In-memory computing (IMC) methodologies have demonstrated substantial improvements in energy efficiency and throughput for DNN workloads by minimizing data movement and eliminating frequent memory accesses. However, unlike many ASIC-based accelerators hsin Chen *et al.* (2018); Zhang *et al.* (2021b); Han *et al.* (2016), few silicon-proven IMC designs have effectively leveraged sparsity through compressed storage and compute skipping to realize similar benefits.

A recent work Liu *et al.* (2023) explored sparsity by employing a binary mask and a custom compression format to encode weights. However, this approach significantly restricts compatibility with a broad range of sparse DNN models. Moreover, the hardware implementation relies on a complex butterfly routing mechanism and incurs additional compute overhead for decoding the compressed weights. Another effort Kim *et al.* (2021) enables sparse computation via compute skipping but suffers from two key limitations: (1) it supports only block-wise sparsity, which limits achievable model accuracy, and (2) it does not operate on truly compressed weights—zeros are still physically stored in the IMC array, thus preserving the compute overhead associated with dense workloads.

Another work Yue *et al.* (2023) delegates sparse computation to a separate ASIC, using the IMC array solely for dense operations. This separation limits the energy and area benefits of sparsity within the IMC fabric. In contrast, SP-IMC directly operates on sparsely compressed weights within the IMC array itself, eliminating the need for external compute and fully leveraging the advantages of in-memory sparsity. Fig. 22 illustrates prior sparse and dense IMC approaches, highlighting their inherent limitations.

5.1.2 Challenges and Benefits of Sparse Compute

The IMC paradigm is naturally well-suited for matrix multiplications due to the structured and parallel nature of memory arrays. However, this advantage diminishes when the matrix structure is disrupted—as is often the case with sparsity, especially after compression. Compressed sparse matrices break the regularity required for efficient in-memory computation, making it challenging to directly apply the IMC model. This challenge is further compounded in the case of unstructured sparsity, where the number and location of non-zero elements vary unpredictably across tensors. As a result, non-uniform data distribution leads to imbalance across IMC compute blocks, causing significant underutilization and degraded performance.

To address these challenges, we propose a design methodology that time-multiplexes

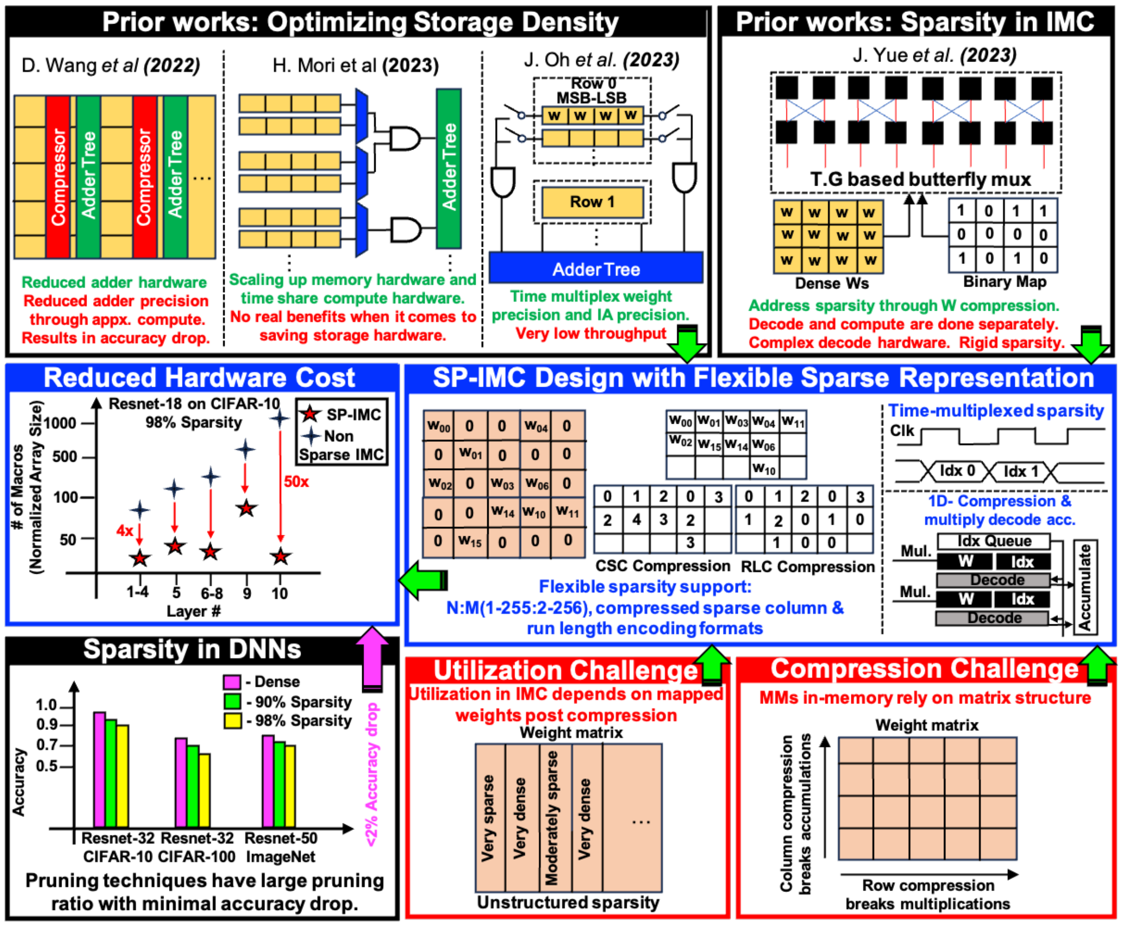

Prior works: Optimizing Storage Density
D. Wang et al (2022)
H. Mori et al (2023)
J. Oh et al. (2023)
Compressor
Adder Tree
Row 0 MSB-LSB
Row 1
Reduced adder hardware
Reduced adder precision through appx. compute.
Results in accuracy drop.
Scaling up memory hardware and time share compute hardware.
No real benefits when it comes to saving storage hardware.
Time multiplex weight precision and IA precision.
Very low throughput
Prior works: Sparsity in IMC
J. Yue et al. (2023)
T.G based butterfly mux
Dense Ws
Binary Map
Address sparsity through W compression.
Decode and compute are done separately.
Complex decode hardware. Rigid sparsity.
Reduced Hardware Cost
Resnet-18 on CIFAR-10 98% Sparsity
SP-IMC
Non Sparse IMC
# of Macros (Normalized Array Size)
4x
50x
Layer #
SP-IMC Design with Flexible Sparse Representation
CSC Compression
RLC Compression
Flexible sparsity support: N:M(1-255:2-256), compressed sparse column & run length encoding formats
Time-multiplexed sparsity
Clk
Idx 0
Idx 1
1D- Compression & multiply decode acc.
Mul.
Idx Queue
W
Idx
Decode
Accumulate
Sparsity in DNNs
- Dense
- 90% Sparsity
- 98% Sparsity
Accuracy
Resnet-32 CIFAR-10
Resnet-32 CIFAR-100
Resnet-50 ImageNet
<2% Accuracy drop
Pruning techniques have large pruning ratio with minimal accuracy drop.
Utilization Challenge
Utilization in IMC depends on mapped weights post compression
Weight matrix
Very sparse
Very dense
Moderately sparse
Very dense
Unstructured sparsity
Compression Challenge
MMs in-memory rely on matrix structure
Weight matrix
Column compression breaks accumulations
Row compression breaks multiplications

Figure 22. Current Sparse IMC Implementation Drawbacks, Benefits of Sparse Encoding, Challenges of Sparse Encoded Weights in IMC

the sparse decode operation. By doing so, we decouple the variability of unstructured sparsity from the hardware footprint and instead map it to the time domain. This approach allows the compute resources to remain fixed, while latency becomes a function of the sparsity level. As a result, we maintain consistent throughput across all sparsity ratios, while still reaping the benefits of compressed storage. Fig. 22 highlights both the challenges posed by sparsity and the advantages of our proposed design.

5.2 Circuit, Architecture and Operation

5.2.1 SP-IMC Macro Architecture

Fig. 23 shows the circuit and architecture of the proposed SP-IMC macro which comprises of 64×128 bit-cells. The IMC macro consists of 16 column groups (CG), where each CG consists of 32 row groups (RG) and one accumulation logic (AL) block. Each RG has 16 bit-cells split into two 8b-rows by a multiply decode and compare (MDC) block. The two 8-bit-rows have four bits of 10T bit-cells for weight storage and another four bits of traditional 6T bit-cells to store the indices of the compressed sparse weights. The bit-cell takes in the activations and performs AND operations with the stored weight to generate partial multiply results. These partial-multiply outs serve as inputs to the MDC block which has shift accumulators/partial multipliers to complete the 4bW2bI/Act multiplications. The 4-bit indices at the top and bottom of the MDC block serve as the input to the RL/COO decode block which calculates the RLC indices based on the index from the previous column or directly pass the indices from the bit-cell to the comparator blocks. Now the comparators compare it with indices generated either by a local counter or the spillover counter. If the comparison is successful, the partial products (PP) are sent to the adder tree and then to a shift accumulator to complete MACs. The adder trees are split into two 32-input trees, and the outputs of adder trees are shift-accumulated. The accumulator precision is chosen by the weight precision control (WPC) signal between 14-bit (for 8b-W) and 11-bit (for 4-bit-W). Finally, a spillover accumulator is present to support edge cases of compressed weights, e.g. uneven sparsity across matrix columns.

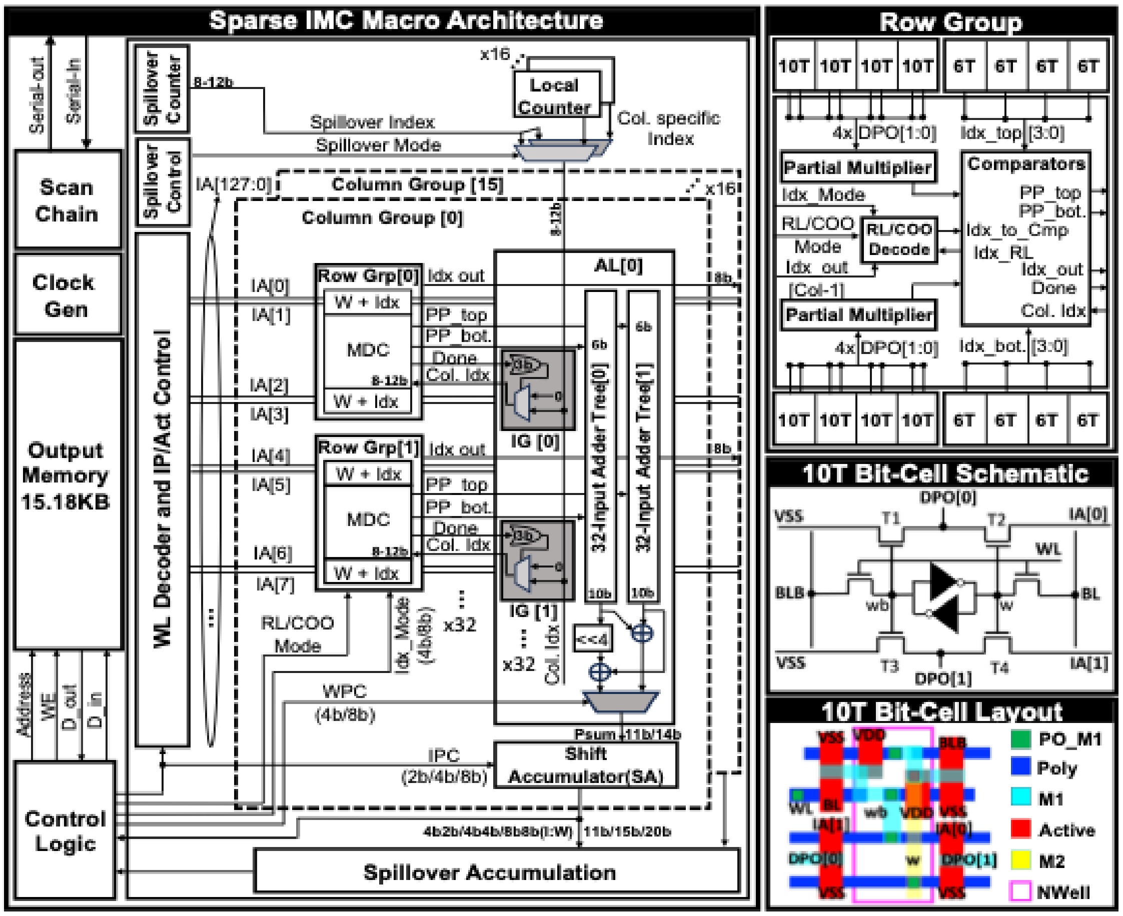


Figure 23. Overall Architecture of SP-IMC Macro, Bit-cell Schematic, Layout, and Micro-architecture of In-memory Decode Hardware. (Adapted from Sridharan *et al.* (2024b) with Permission)

5.2.2 Bitcell Design

The 10T bit-cell extends the conventional 6T cell by incorporating four additional transistors T1–T4. Transistor pairs "T1", "T2" and "T3, T4" function as parallel AND gates, enabling simultaneous bitwise operations between the streamed input activation (IA) and the stored weight. This parallelism mitigates the IA stream-in bottleneck commonly encountered in prior designs Wang *et al.* (2022); Mori *et al.* (2023a); Oh *et al.* (2023a); Yue *et al.* (2023); Fujiwara *et al.* (2022a), which rely on purely bit-serial methods to support high input precision. Additionally, this 2T-AND scheme resolves the issue of floating partial products observed in the PSIMC bit-cell design Sridharan *et al.* (2024a) when weights are zero. Each 10T bit-cell performs a 1-bit weight by 2-bit IA multiplication, and the four 10T cells positioned at the top and bottom collectively compute two 4b-IA $\times$ 2b-W partial products. Fig. 23 shows the schematic and layout of the proposed bit-cell.

5.2.3 Sparse Compression Formats and Variable Sparsity Ratio

The SP-IMC architecture is designed to support multiple compression formats to accommodate a wide range of sparse DNN workloads. Specifically, it implements two representative sparse encoding schemes: the coordinate (COO) format and run-length encoding (RL). In the COO format, each non-zero value is stored alongside its explicit index, providing full flexibility in representing unstructured sparsity. In contrast, RL encoding captures the number of zeros between successive non-zero elements using a compact zero-count value, offering a more storage-efficient representation for sparsity patterns with longer contiguous runs of zeros.

In addition to these formats, SP-IMC also supports N:M sparsity, a semi-structured compression scheme wherein only one out of every M elements is retained, and the remaining $N-1$ elements are pruned. This format, initially developed and commercialized by NVIDIA (supporting only 2:4 sparsity in hardware), strikes a balance between unstructured flexibility and structured hardware efficiency. Unlike NVIDIA's fixed-ratio approach, SP-IMC offers generalized support for a wide range of N:M formats—from 1 (to) 255:16 (t0) 256 without requiring any hardware modifications, thanks to its fully unstructured sparsity support and flexible indexing logic. This capability makes SP-IMC highly adaptable to diverse model requirements, including models where each layer may benefit from a different sparsity ratio. By supporting flexible sparsity formats and adaptive index precision, SP-IMC maximizes hardware utilization and energy efficiency across a wide spectrum of DNN sparsity patterns. Additional benefits of deploying N:M sparsity in SP-IMC, including compute regularity and hardware reuse opportunities, will be discussed in detail in Section 5.2.5. From a hardware perspective, the maximum sparsity ratio that can be supported is governed by the bit-width of the index or zero-count field, which determines how many unique positions or gaps can be represented. To accommodate this, SP-IMC introduces variable index precision, controlled by an Idx_Mode signal. This signal selects between 4-bit and 8-bit index modes, trading off between metadata overhead and sparsity granularity. In COO format, a 4-bit index can represent up to 16 positions, while in RL format, it can encode up to 15 zeros between two non-zero values. The 8-bit mode extends this capability, allowing up to 256 distinct indices in COO or 255 zeros in RL, thus enabling significantly higher sparsity ratios when needed

5.2.3.1 Compression Direction

Matrices are inherently two-dimensional, and compression can be applied along either the row or column direction. Compressing along rows preserves the accumulation structure but disrupts the multiplication pattern. In contrast, compressing along columns retains the multiplication structure while breaking the accumulation flow. In IMC architectures, activations are typically streamed in through horizontal word-lines, which aligns with the multiplication dimension of the matrix. As a result, preserving the multiplication structure is critical for efficient in-memory computation. Therefore, SP-IMC adopts column-wise compression, commonly referred to as Compressed Sparse Column (CSC) format, as it aligns naturally with the compute flow in IMC and maintains compatibility with the activation streaming mechanism.

5.2.4 Dataflow and Pipeline

RL and COO formats have different dataflows to support the decode of their respective indices, as shown in Fig. 24 N:M sparse encoding follows the dataflow of COO. In COO mode, each column of the memory array generates an index (through local counters) every cycle that pertains to the index of stored weights in their respective columns and these indices are used to gate accumulations of PP generated in each RG using the comparators. The accumulated PPs are then sent to the shift accumulator block for IA precision compensation semi-bit serially (2-bits/cycle). The dataflow is similar in RL mode as well, but the index generator (counters) now generates the zero count (ZC) between two non-zero weights, and RL compression incurs additional decode hardware in the row direction to specify the non-zero weight position. The index stored in the neighboring CG[n-1] is streamed and is added with the indices in the current CG[n]. The spillover dataflow exists to support corner cases, for e.g., all elements of a matrix row are not always mapped to the same CG, (Fig. 25 COO-CSC mapping) and can "spillover" to neighboring CGs. This arises out of uneven sparsity across matrix columns and is the case for RL, COO and N:M sparsity. It is greatly reduced for N:M sparsity due to fixed M. Fig. 24 shows the pipeline diagram of 4b-IA:4b-W MACs from a CG in the SP-IMC macro. It also shows the priority queue for index handling and the parallelism achieved in a SP-IMC to process sparse compressed MACs.

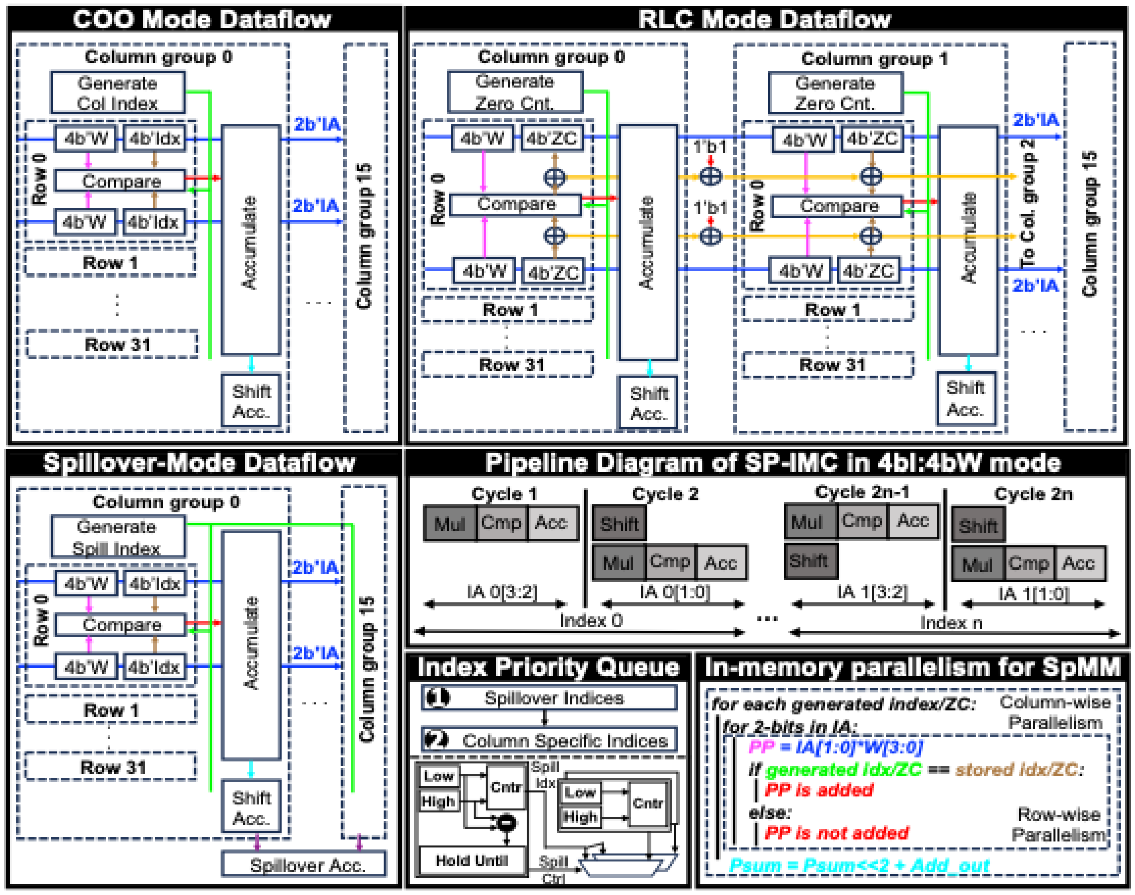


Figure 24. Dataflow of Various Modes in SP-IMC, Pipeline Diagram, Index Priority Queue, SpMM Parallelism in Memory

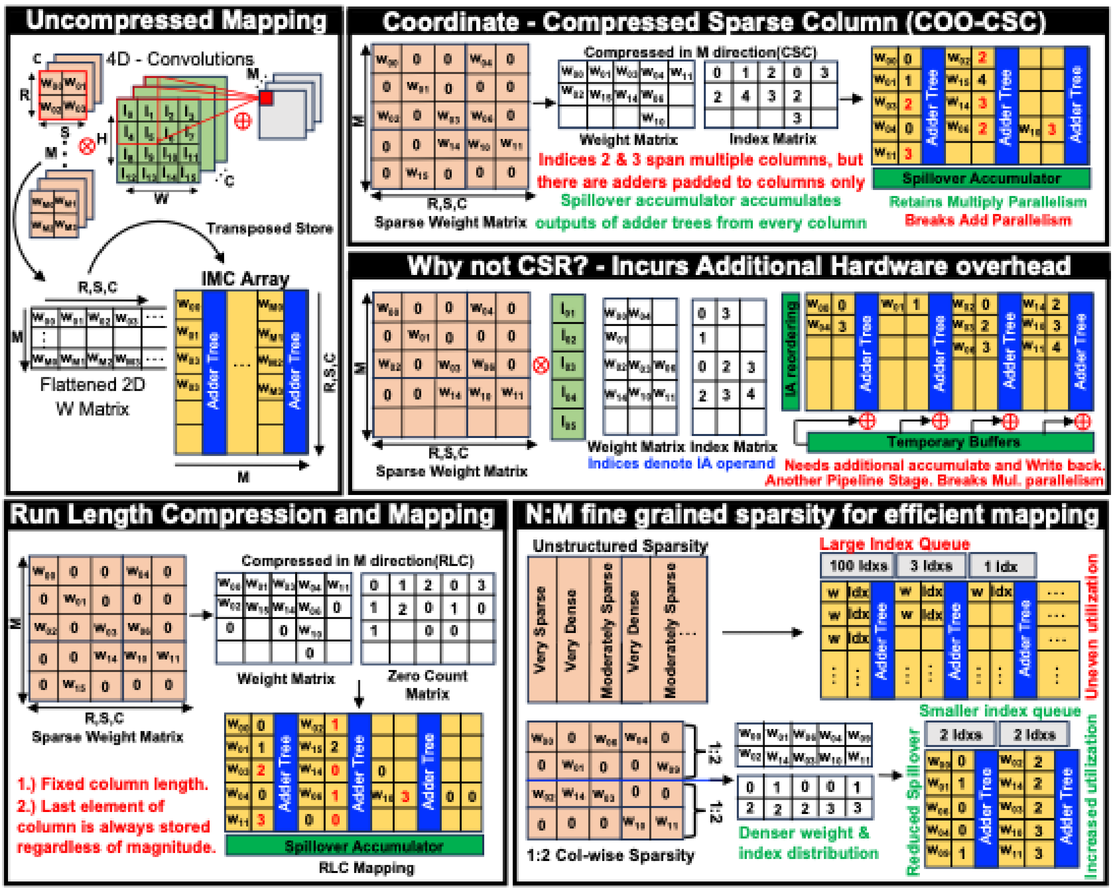
5.2.5 Mapping Compressed kernels

The compression and IMC mapping methodology is elaborated in Fig. 25. Uncompressed mapping is done for convolutions by first flattening the 4D kernels to a 2D weight matrix and is transposed and stored onto the IMC array such that the kernel dimensions and input channel (R, S, C) fall into columns with adder trees and the output channel is mapped in the row direction to support parallel multiplications. We employ a similar approach when it comes to mapping compressed weights. Encoding in column direction is more IMC friendly because it retains column structure while breaking row structures, i.e., breaks accumulations and retains multiplications. Compressed sparse row (CSR) is not very IMC friendly and incurs additional hardware overhead i.e., IA reordering, additional accumulate and WB operations, hence not implemented in this macro. RL mapping is similar to COO, the indices are replaced with ZC. In RL to denote the end of each matrix column, its length is fixed, and the last element of all matrix columns are stored regardless of magnitude. Mapping matrices that have unequal non-zero weight distribution in every column will lead to utilization issues in the IMC. This can be alleviated during training by employing a fine-grained N:M sparsity structure. Through this method the SP-IMC macro can also achieve a significant speedup and better utilization by fixing the indices/column.

5.3 Experiment Results and Analysis

5.3.1 Chip Measurements

The SP-IMC chip is fabricated in a 28nm CMOS process and evaluated across a supply voltage range of 0.57V to 1.2V at a nominal temperature of 25°C. Under a 25% input toggle rate (TR), SP-IMC achieves energy efficiencies ranging from 8.4 to 36.6 TOPS/W for dense 4b-IA $\times$ 4b-W MAC operations—defined as the fully non-sparse case where each column activates all bit-cells and adder tree nodes via a single index. For a 1:16 pruning ratio (i.e., high sparsity), the energy efficiency significantly improves, reaching 7.5 to 115.3 TOPS/W under the same TR. Furthermore, a 25% reduction in TR results in an average 10% gain in energy efficiency, highlighting the benefits of low activity factor designs. The chip supports a peak operating frequency of 1.16 GHz at 1.18V across all operational modes.

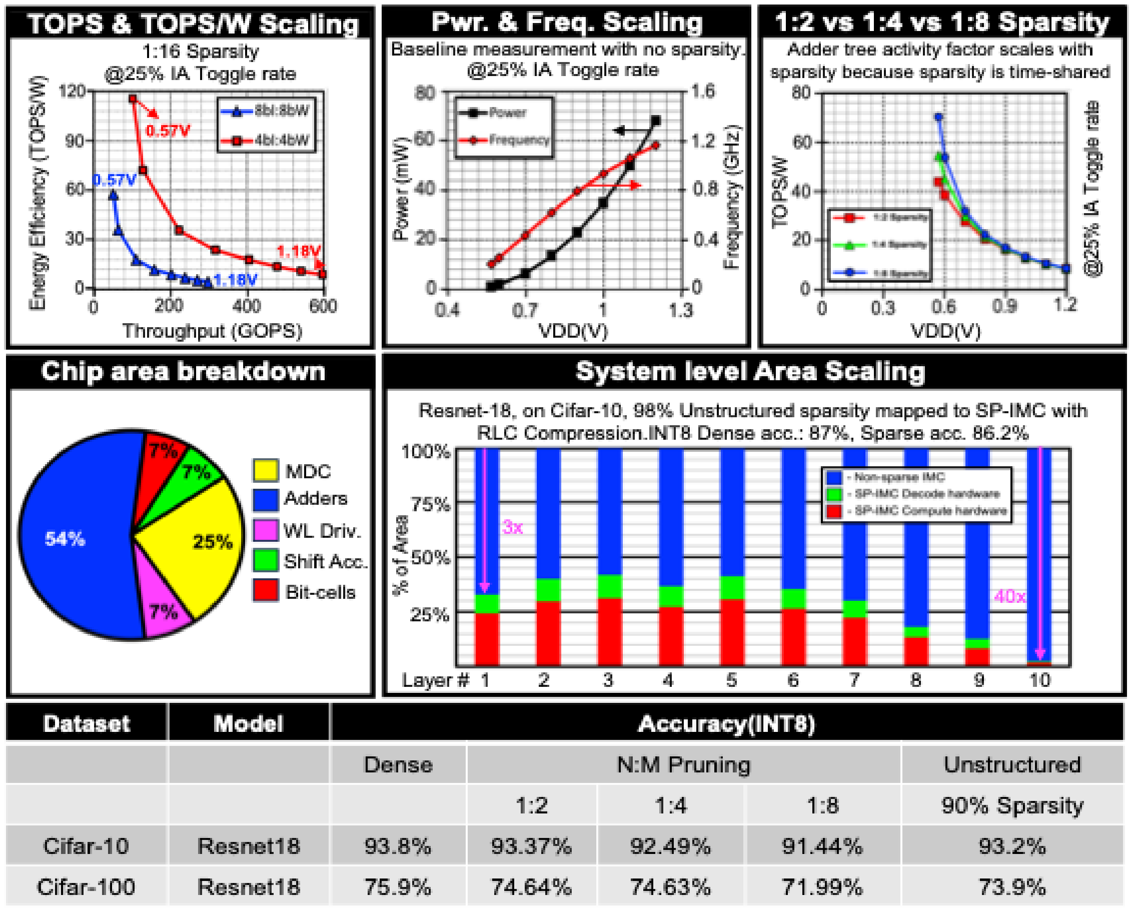


Uncompressed Mapping
4D - Convolutions
Transposed Store
IMC Array
Flattened 2D W Matrix
Adder Tree
Coordinate - Compressed Sparse Column (COO-CSC)
Compressed in M direction(CSC)
Weight Matrix
Index Matrix
Sparse Weight Matrix
Indices 2 & 3 span multiple columns, but there are adders padded to columns only
Spillover accumulator accumulates outputs of adder trees from every column
Spillover Accumulator
Retains Multiply Parallelism
Breaks Add Parallelism
Why not CSR? - Incurs Additional Hardware overhead
IA reordering
Indices denote IA operand
Temporary Buffers
Needs additional accumulate and Write back. Another Pipeline Stage. Breaks Mul. parallelism
Run Length Compression and Mapping
Compressed in M direction(RLC)
Zero Count Matrix
1.) Fixed column length.
2.) Last element of column is always stored regardless of magnitude.
RLC Mapping
N:M fine grained sparsity for efficient mapping
Unstructured Sparsity
Very Sparse
Very Dense
Moderately Sparse
Large Index Queue
100 Idxs
3 Idxs
1 Idx
Uneven utilization
Smaller index queue
2 Idxs
1:2 Col-wise Sparsity
Denser weight & index distribution
Reduced Spillover
Increased utilization

Figure 25. Mapping Methodologies for COO-CSC, RL, Why CSR Is Not IMC Friendly and Benefits of N:M Sparse Encoding

SP-IMC leverages a time-multiplexed sparsity mechanism that maps unstructured sparsity to the time domain. As the number of active indices per column decreases (i.e., increased sparsity), the adder tree activity factor is reduced since fewer adders are engaged per cycle. Fig. 26 presents the detailed measurement results, including the macro-level area breakdown. The adder trees dominate the macro area, followed by the multipliers and comparators.

To evaluate the benefits of sparse compressed storage in reducing the number of required macros, we scaled up the SP-IMC design and mapped a ResNet-18 model (trained on CIFAR-10 with 98% unstructured sparsity) onto the architecture.



Dataset	Model	Accuracy(INT8)				
		Dense	N:M Pruning			Unstructured
			1:2	1:4	1:8	90% Sparsity
Cifar-10	Resnet18	93.8%	93.37%	92.49%	91.44%	93.2%
Cifar-100	Resnet18	75.9%	74.64%	74.63%	71.99%	73.9%

Figure 26. Chip Measurement Results, Accuracy Results of Pruned DNNs, Area Breakdown in Macro and System Level

Compared to a dense baseline, we observe a 3× to 40× reduction in area, depending on the layer and sparsity configuration.

We also conducted extensive software-level validation to assess the accuracy of sparse DNN models when deployed on SP-IMC. As shown in Fig. 26, both unstructured and N:M sparsity schemes—particularly at high sparsity ratios—achieve competitive accuracy on CNN models such as ResNet-18, demonstrating the practicality and effectiveness of sparsity support in SP-IMC.

5.3.2 Comparison and Evaluation

Fig. 27 presents a comparison between SP-IMC and prior state-of-the-art digital IMC designs. Even without leveraging sparsity, SP-IMC achieves the highest throughput, primarily due to its dual activation word-line parallelism. A key focus of this work is on sparse compressed storage, enabling SP-IMC to maintain consistent hardware throughput across various sparsity formats.

To evaluate the benefits of compressed storage, we compare the number of write operations required to perform a large volume of MAC operations. SP-IMC significantly reduces the number of writes compared to prior works. Most existing IMC designs adopt a weight-stationary compute model, where weights must be written into the array prior to computation hsin Chen *et al.* (2018). This step introduces considerable latency at the system level, especially for large models. In contrast, SP-IMC dramatically reduces this overhead by minimizing the number of weight parameters through sparsity-aware compression.

This improvement is illustrated in the bottom two plots of Fig. 27. The left plot shows the total cycle latency as a function of increasing MAC operations, while the right plot depicts the number of write operations required relative to MAC count. SP-IMC consistently achieves the lowest cycle latency and the fewest write operations, even at scale, demonstrating its efficiency in both compute and memory access.

To quantitatively capture the benefits of sparsity and compressed storage, we define a new Figure of Merit (FoM): TOPS/W $\times$ TOPS/mm2 $\times$ # of weights stored per kb. This metric reflects not only energy and area efficiency, but also storage density enabled by compression. Under this proposed FoM, SP-IMC achieves up

Work	D. Wang *et al (2022)*	J. Yue *et al. (2023)*	H. Fujiwara et al. (2022)	H. Mori et al **(2023)**	J. Oh *et al.* ***(2023)***	**This Work**
Technology	28nm	28nm	5nm	4nm	28nm	28nm
MAC Implementation	Digital	Digital	Digital	Digital	Digital	Digital
IMC Sparsity Support	Approx. Addition	X	X	X	X	RLC/CSC/N:M
Supply Voltage (V)	0.45-1.10	0.64-1.03	0.5-0.9	0.32-1.1	0.9-1.1	0.57-1.18
Macro Area (mm^2)	0.049	NA	0.0133	0.0172	0.0159	0.24
Clock Frequency (MHz)	250	20-320	360-1440	1490	30-360	201-1160
Bitcell Transistors	8T	8T(55%) 10T(45%)	12T	8T x 2bit +OAI	6T+0.5T	6T+4T(50%) 6T(50%)
Array Size(b)	16K	1.15M	64K	54K	16K	4K(Weights) + 4K(Index)
Bit Precision	IA:1-4b W:1b	INT8	IA: 1-8b W:4b	IA: 8/12/16 W: 8/12	IA: 1-8 W: 8	IP:2b/4b/8b W:4b/8b
Full output precision	No	Yes	Yes	Yes	Yes	Yes
Performance(GOPS)^{1,2}	62.5*	22.9*	104.735	127.15	0.95-11.6	**41.29-238.86^{B}**
Energy Efficiency^{2} (TOPS/W)	9.6-15.5	15.6^{4}/70.37^{5}	17.5-63	87.4	22.4-60.4	4.38-57.67^{6,7}
Compute Density^{2,3} TOPS/mm^2	2.59	0.85	0.44-1.76	49.9**	0.12-1.46	0.21-1.2^{7}

(1) Normalized to 8Kb. (2) One operation is either 8b multiplication or addition. (3) Normalized quadratically to 28nm.
*Estimated from previous works. (4) 75% Sparsity, (5) 92% Sparsity. (6) 93.75% Sparsity (15:16 Pruning).
GOPS Calculation: 32(Rows) x 16(Columns)/Latency(5xClk period). (7)Excludes write energy/latency otherwise incurred by other works for a scaled-up matrix that fits in SP-IMC and not in other works. (8) Includes write latency - 1 cycle/word size.

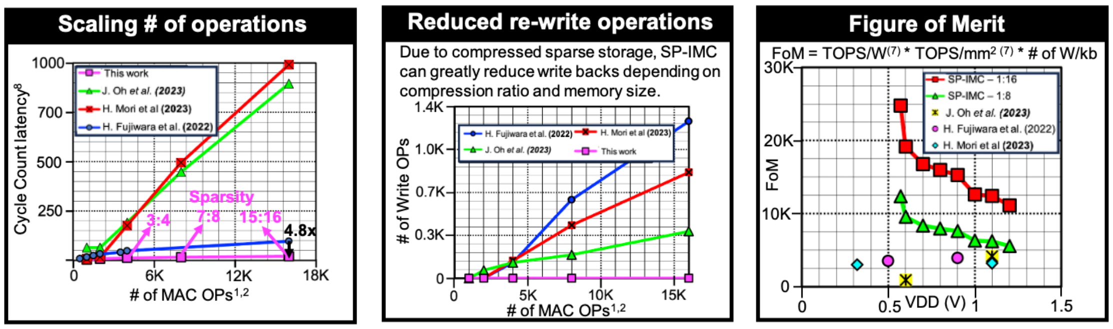


Figure 27. System Latency, Write Operations, Figure of Merit (FOM), and Comparison to Prior Digital IMCs

to 5.9× improvement over the best prior work, highlighting its superior balance of compute throughput, area efficiency, and sparsity-aware storage capability.

5.4 Conclusion

In summary, this work presents SP-IMC, a fully digital sparsity-integrated in-memory computing (IMC) macro that pushes the boundaries of flexibility, efficiency, and scalability in sparse DNN acceleration. Unlike prior designs that either lack support for compressed representations or require significant architectural modifications to handle different sparsity types, SP-IMC natively supports a wide range of sparse encoding formats—including Coordinate (COO), Run-Length (RL), and N:M structured sparsity—directly within the IMC array. SP-IMC also accommodates multiple data precisions to suit diverse DNN workload requirements. Specifically, it supports input activations (IA) at 2-bit, 4-bit, and 8-bit precision, and weights at 4-bit and 8-bit precision. To maximize sparsity coverage, the design features scalable sparsity encoding, with support for both 4-bit and 8-bit index or zero-count fields, enabling real-time configurability for different sparsity ratios and storage efficiencies. This flexible architecture allows SP-IMC to efficiently execute a wide range of modern DNN models—such as CNNs and Transformers—under various sparsity and quantization regimes, while maintaining high throughput, minimal energy overhead, and reduced area footprint. By integrating compressed storage and time-multiplexed compute, SP-IMC not only reduces memory and compute resources but also ensures sustained performance across diverse sparsity patterns. As a result, SP-IMC represents a significant step forward in building general-purpose, sparsity-aware IMC accelerators that can adapt to the growing demands of edge and datacenter-scale AI workloads.

Chapter 6

SPARSITY INTEGRATED COMPUTE-IN-MEMORY ACCELERATOR WITH A FUSED DOT-PRODUCT ENGINE AND A RISC-V CPU

Previous chapters explored the implementation of unstructured sparsity and precision scaling in single macro silicon designs. However, these implementations exhibit inefficiencies, such as redundant multiplications and additions, and there is no visible reduction in compute resources other than reduced storage. Moreover, prior designs are confined to single macro implementations, which are inadequate for evaluating how large-scale LLM and CNN models can be efficiently mapped and scaled-up hardware. And previous chapters targeted scalable fixed precision workloads, which combined with high sparsity levels cannot support several AI models. This chapter addresses all the above shortcomings by presenting a sparsity-aware in-SRAM multiply-and-accumulate (MAC) accelerator with a fused dot-product engine (SAFE) and a RISC-V CPU (SAFER). For the first time, we implement a unified dot-product compute methodology in Compute-in-memory (CIM) circuits vastly reducing the hardware footprint for simultaneously supporting both floating point (FP) and integer (INT) MACs. Additionally, we integrate various N:M sparsity formats allowing the CIM macro to store and operate exclusively on compressed non-zero weights. We also tightly integrate a 32-bit RISC-V CPU to SAFE for efficient data-movement across chip. The CPU orchestrates data-movement across 4 implemented macros to enhances matrix scaling for large AI workloads. The 28nm SAFER prototype achieves a peak energy efficiency of 105.7 TOPS/W (78.9 TOPS/W) and 79.9 TOPS/W (63 TOPS/W) in the macro (chip) level for FP8 and INT8 workloads respectively. SAFER also achieves

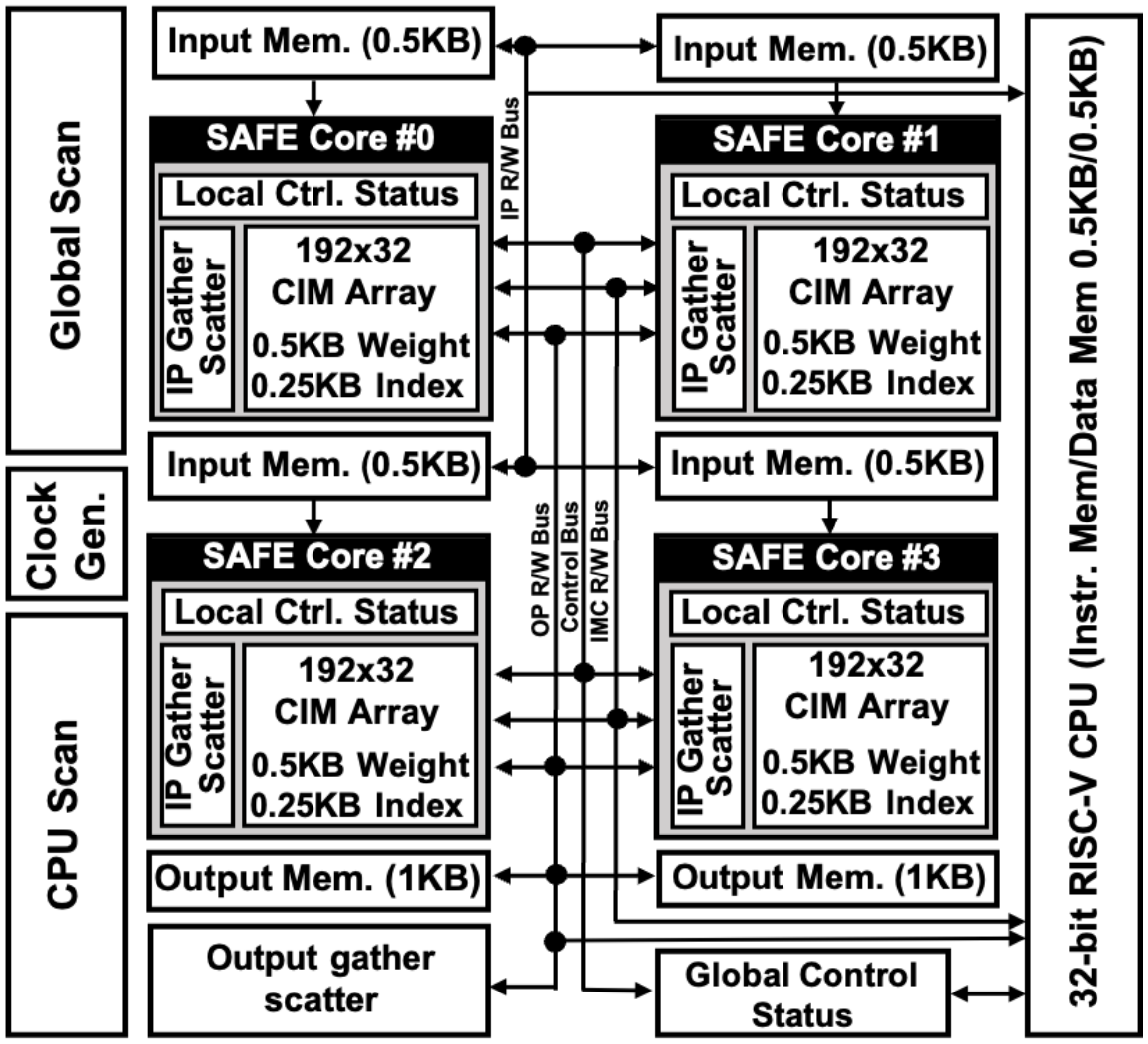


Figure 28. SAFER Chip Architecture

a memory footprint reduction proportional to sparsity through compressed storage, vastly reducing the macro count required for large AI models. For our proposed figure of merit, which accounts for memory footprint, SAFER improves current SoTA CIMs by 1.4× and 14× for INT8 and FP8 workloads.

6.1 Introduction

There have been a plethora of CIM designs targeting various hardware and software features to enable efficient processing of deep neural networks (DNNs).

Sparsity is one such feature and has been widely adopted in various DNNs. DNN models are getting larger and is becoming more challenging to fit them on chip. This is further exacerbated by the fact that current CIMs have poor storage density due to integration of compute logic circuits. Weight sparsity achieved through pruning in DNNs, offers significant memory footprint reduction when paired with compression. Compressed storage can also reduce memory accesses to CIMs when mapping large DNN workloads. Prior CIM works have explored this: Liu *et al.* (2023) uses bitmaps for compression but requires external compute and complex dense-format conversion via butterfly multiplexers. SP-IMC Sridharan *et al.* (2024b) adopts compressed sparse column (CSC) with simplified decode hardware but suffers from low MAC utilization. SAFE aims to address these drawbacks by offering a simple decode mechanism for various sparsity ratios and full utilization of the compute circuits.

Data-type re-configurability is another feature that enables support for various DNNs. Supporting different number formats such as integer (INT) and floating-point (FP) precision in CIM hardware is expensive and needs careful hardware re-use. Recent CIM works have explored this, but schemes in Yuan *et al.* (2025); Yue *et al.* (2025); Ali *et al.* (2025) trade off compute accuracy in FP arithmetic for hardware complexity, whereas Saikia *et al.* (2023) has no accuracy drop but incurs large hardware overhead. SAFE, for the first time, explores a fused-dot product (FSD) approach in CIM, which was previously only employed in ASICs Hickmann *et al.* (2020); Park *et al.* (2023). This method can reduce the cost of rounding and normalization for large vector-vector

MACs Hickmann *et al.* (2020). Each column in CIM arrays typically performs large vector-vector MACs, therefore FSD can naturally fit well with CIM array design. Additionally, FSD scheme uses fixed-point adders for accumulations. Adder trees occupy a large footprint in digital CIMs, which can be amortized by using the same adder trees in FSD for both INT and FP.

MAC configuration flexibility is another key feature that requires attention as there are a variety of AI workloads all requiring different MAC structures. Self-attention layer in large language models (LLMs) differs from a convolution layer and even convolutions vary from layer to layer in different DNNs. SAFER integrates a custom RISC-V CPU to cast weights and activations across all memories on chip to help with MAC reconfigurability. We also augment the RISC-V CPU with vector additions to help with post accumulation of partial sums for large matrices.

6.2 Architecture and Operation

6.2.1 SAFER Chip Architecture

Fig. 28 shows the SAFER chip architecture, consisting of four SAFE cores, input/output (IP/OP) buffers, control logic, and OP gather-scatter unit. To enable parallel processing, each SAFE core is allocated a dedicated 0.5KB IP buffer for feeding unique IPs. The OPs/partial-sums from all SAFE cores are collected and transferred to two 1KB OP buffers via the OP gather-scatter unit. We incorporate a custom 32-bit single-cycle RISC-V CPU to aid with the address calculations for data movement between IP/OP buffers and weights (Ws) in the CIM array. This

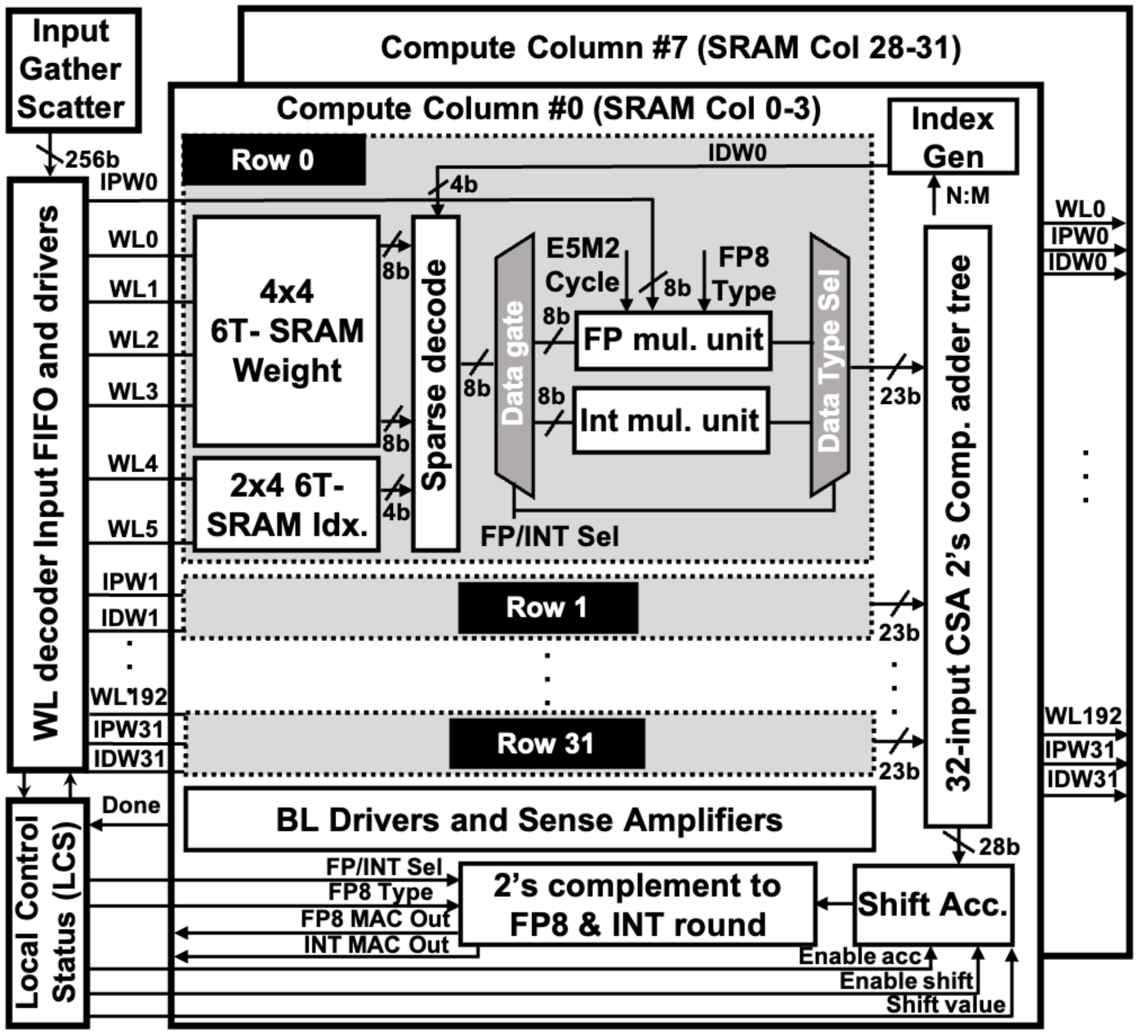


Figure 29. SAFE Core and Macro Architecture

is done by augmenting the base RV32-IM instruction set with additional load/store instructions. These additional instructions provide support for a variety of MAC configurations through uni/multi-casting Ws/OPs/IPs to any/all of the SAFE cores. The CPU also monitors the status of all SAFE cores through global control status registers. Additionally, we include FP8 (E4M3 and E5M2) and INT32 vector addition to the CPU to enable partial-sum aggregations from SAFE macros for large matrices. Table 10 provides the list of all supported instructions.

Table 10. RISC-V Instruction Set Extensions

Instruction Type	Function
RV32IM Base	**All ratified instructions**
Load store instructions *New opcode for each instruction, honors the same funct field for byte,1/2 byte, word. R-Type: Loads to CPU reg, stores to CPU D-mem.	**Load/Store IMC**
	Load/Store IP mem
	Load/Store OP mem
	Load/Store global CSR
Floating point Extensions *(R-type and I-type extension)	**FP8 E4M3/E5M2 8-vector addition**
	FP8 E4M3/E5M2 multiply
	FP8 E4M3/E5M2 Single add
	32-bit INT 8-vector addition
Copy instructions *Moves data from non CPU memory to desired location *(R-type and I-type extension)	**Copy desired IP mem location to OP mem location and vice versa**

6.2.2 SAFE Macro Architecture

Fig. 29 illustrates a single SAFE core. Each core has a 192×32 CIM array which is broken into eight FSD “compute columns" (CCs) for compressed W storage and sparse FP8/INT8 MAC operations. Each core also has an input gather-scatter (IGS) which retrieves IPs and distributes them to the CCs through the IP FIFOs. The CIM array also has WL-decoders, BL-drivers, and sense amplifiers to facilitate row-by-row read and writes. The control logic manages compute modes and sparsity ratios. Status registers track MAC count and manage IP requests via the IGS.

6.2.2.1 Compute Column (CC)

Each CC in a SAFE macro contains 192×4 6T-SRAM bits, out of which 128×4 is used for W storage and 64×4 for Idx storage. The CC also includes a 32-vector FSD MAC unit, supported by a backup SRAM with 64 storage locations for 8-bit weights. This backup SRAM allows a new set of weights to be written to the CIM array while the current set is actively used for computation. CC is divided into 32 rows, where each row has 4×4 bits for W storage and 2×4-bits for Idx storage. These Ws and Idxs are sent to a sparse decode unit which filters two 8-bit Ws using the two 4-bit Idxs into one weight (or 0) and this weight is sent to the multipliers. There is an individual FP8 and INT8 multiplier to handle both data-types. The outputs from the multiplier is sent to a 32-input 2's complement adder tree. The multipliers are data-gated to save power between different data types. Both data types share the same adder tree as the accumulation data format is normalized between the two. The outputs from the adder tree is then fed into a shift accumulator (SA). The shift portion of the SA can be enabled by the control logic depending on the data type. Now the 2's complement partial-sum from the SA can optionally converted back to FP.

6.2.2.2 Sparse Decode Unit (SD)

As shown in Fig. 30, SAFE's sparse decode method supports various N:M sparsity formats. It adopts a compressed sparse row (CSR) compression scheme, storing an index for each non-zero weight. The index bit-width is determined by the maximum supported M in N:M. All supported N:M ratios and their corresponding bit-width

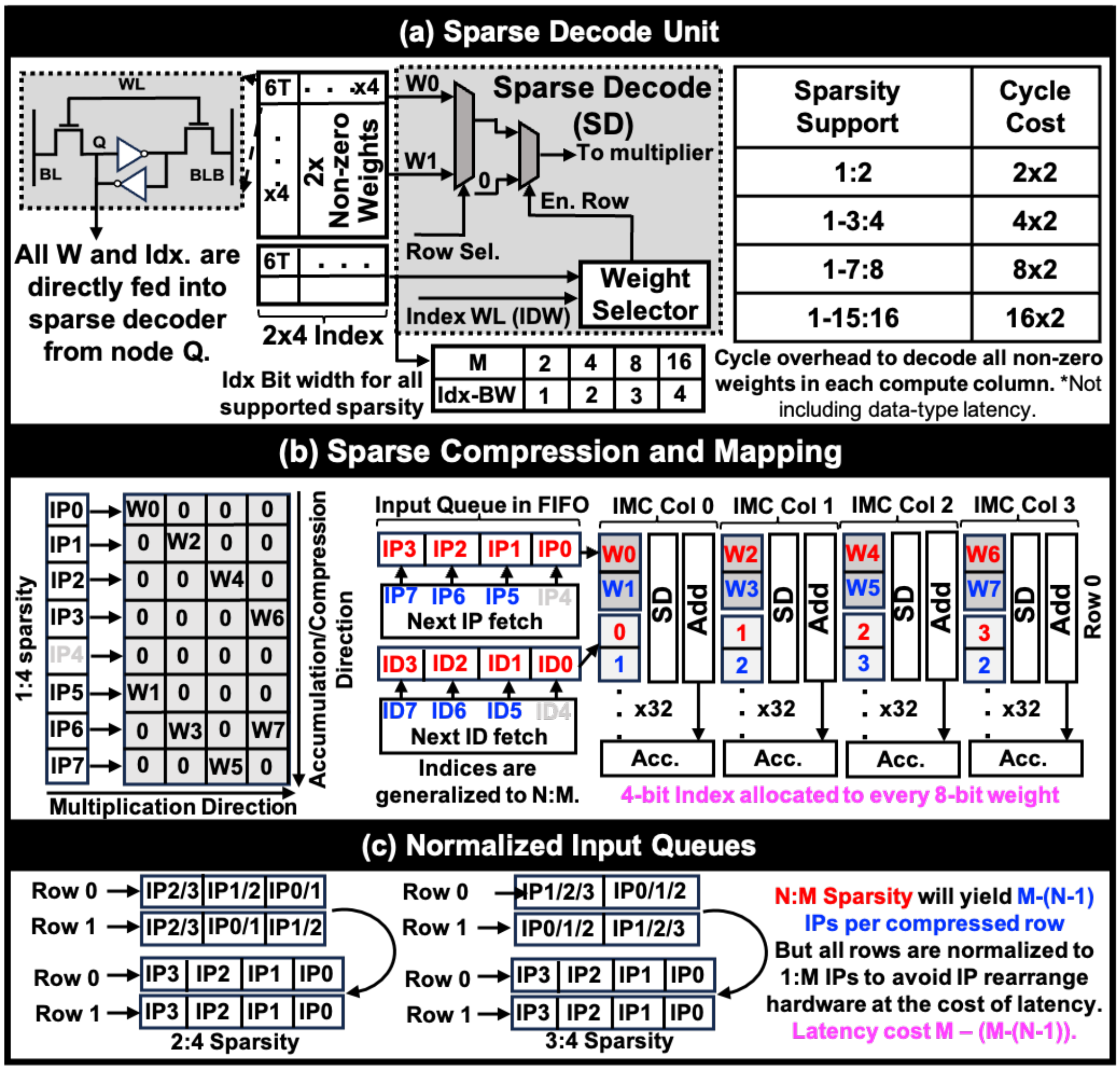

(a) Sparse Decode Unit
WL
Q
BL
BLB
6T
x4
2x Non-zero Weights
All W and Idx. are directly fed into sparse decoder from node Q.
2x4 Index
W0
W1
Sparse Decode (SD)
To multiplier
0
En. Row
Row Sel.
Weight Selector
Index WL (IDW)
Idx Bit width for all supported sparsity
M 2 4 8 16
Idx-BW 1 2 3 4
Sparsity Support
Cycle Cost
1:2 2x2
1-3:4 4x2
1-7:8 8x2
1-15:16 16x2
Cycle overhead to decode all non-zero weights in each compute column. *Not including data-type latency.
(b) Sparse Compression and Mapping
1:4 sparsity
IP0 IP1 IP2 IP3 IP4 IP5 IP6 IP7
W0 W1 W2 W3 W4 W5 W6 W7
Multiplication Direction
Accumulation/Compression Direction
Input Queue in FIFO
IP3 IP2 IP1 IP0
IP7 IP6 IP5 IP4
Next IP fetch
ID3 ID2 ID1 ID0
ID7 ID6 ID5 ID4
Next ID fetch
Indices are generalized to N:M.
IMC Col 0
IMC Col 1
IMC Col 2
IMC Col 3
SD
Add
x32
Acc.
Row 0
4-bit Index allocated to every 8-bit weight
(c) Normalized Input Queues
Row 0
Row 1
IP2/3 IP1/2 IP0/1
IP2/3 IP0/1 IP1/2
IP3 IP2 IP1 IP0
2:4 Sparsity
IP1/2/3 IP0/1/2
IP0/1/2 IP1/2/3
3:4 Sparsity
N:M Sparsity will yield M-(N-1) IPs per compressed row
But all rows are normalized to 1:M IPs to avoid IP rearrange hardware at the cost of latency.
Latency cost M – (M-(N-1)).

Figure 30. (A) Sparse Decode Unit, (B) Sparse Compression and Mapping, (C) Reducing Hardware Complexity by Normalizing Input Queue Length

is shown in Fig. 30(a). The stored indices specify which weights the IPs must be multiplied with before accumulation. Each row in a CC has an Idx word line (IDWL); as IPs are streamed into the macro via the IPWs, the corresponding indices are simultaneously streamed through the IDWLs and the W selector uses these indices to determine whether the corresponding weight needs multiplication from the streamed-in IPs. This is required because when a matrix is compressed along the accumulation direction, it breaks the multiplication structure. Not all Ws need to be multiplied with the streamed-in IPs. We also time-multiplex sparsity, this is because there is only a single IPW for every W, if a 1:4 sparsity is implemented then for each CIM row, four IPs need to be streamed-in along with 4 Idxs and will vastly increase the routing resources. The sparse compression and mapping mechanism is shown in Fig. 30(b). For N:M sparsity where $N{\neq}1$, each row would require $M{-}(N{-}1)$ IPs, and the IPs required in each row are not in-order. To avoid more hardware for input re-arrangement, we stream-in all M IPs regardless of N. This normalizes the cycle count to M, as described in Fig. 30(c).

6.2.2.3 FP8 fused dot-product (FSD) and INT8 MAC

The FSD method supports vector-vector MACs for FP data-types, as shown in Fig. 31(b). FP multiplications begin with exponent addition and mantissa multiplication; instead of rounding, the mantissa's precision is preserved and is shifted by the exponent and mapped to a 2's complement number line. For E4M3, the number line spans $\pm 2^9 - 2^{-12}$ (23 bits) and for E5M2 the number line spans $\pm 2^{16} - 2^{-18}$ (36 bits). Now that the vectors are in 2's complement format, they can be accumulated using a fixed-point adder. The adder-tree bit-width is set to 23 bits instead of 36 bits because of area constraints. To support E5M2, we time-multiplex the adder hardware over two cycles. The first cycle is used to handle all exponents below 18 and the next cycle is used for exponents above 18 and the shift accumulator performs shift and accumulate computation of the two partial sums. Fig. 31(a) shows the detailed FP multiply units in SAFE. To support INT8, an 8-bit integer multiplier is added to the pipeline after

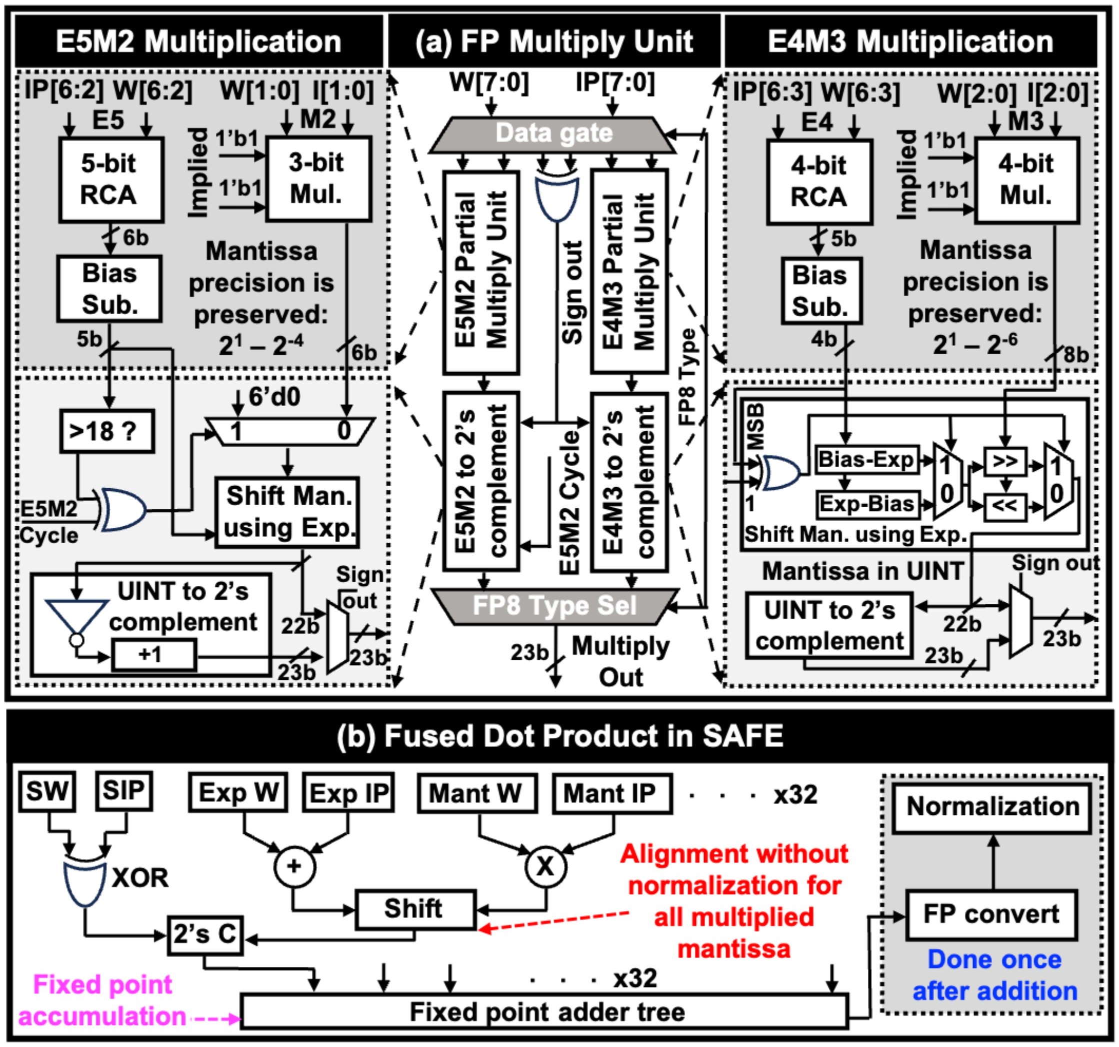


Figure 31. (A) Floating-point Multiply Unit and (B) Fused Dot-product in SAFE

the SD stage, while the adder hardware is reused; this is shown in Fig. 29. Fig. 32 illustrates SAFE's pipeline for all supported data types.

6.3 Chip Measurements and Results

SAFER is prototyped in 28nm CMOS. It occupies $0.95mm^2$, and each SAFE core occupies $0.15mm^2$. Fig. 33 shows the power/area breakdown for a SAFE core. FP8

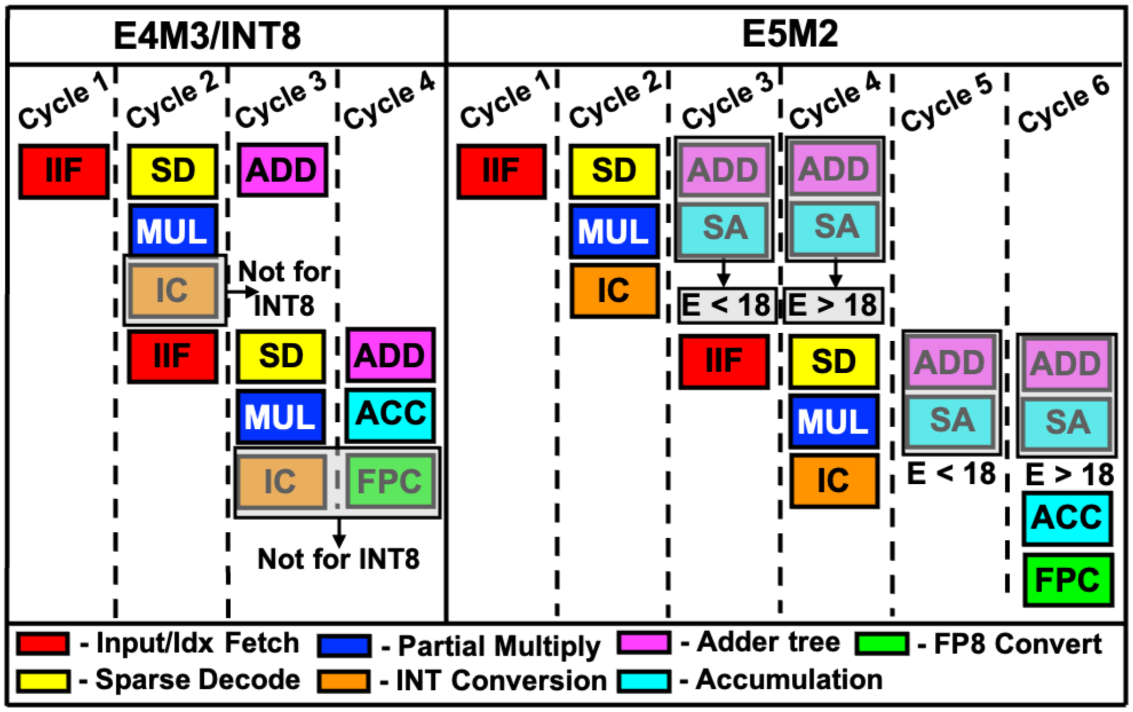


Figure 32. Pipeline Diagram for a 1:2 Sparse Workload

MACs consume less power and area, compared to INT8 MACs. This is due to the wide multiplier unit for INT8 and the 2's complement version of an FP8 number is very sparse and reduces the overall activity factor of the adder tree. In prior digital CIM works Sridharan *et al.* (2022a); Chih *et al.* (2021); Lin *et al.* (2023); Oh *et al.* (2023b), the adder tree typically dominated power and area overheads compared to multipliers. This is because bit-serial MAC operations reduce multiplications to simple per-cycle AND operations. But Sridharan *et al.* (2024a) mitigates multiplier overhead in bit-parallel designs by isolating weight precision and deferring precision handling, akin to the bit-serial approach which does this in the time domain. In contrast, SAFER implements full-precision 8b×8b multipliers for every selected weight, avoiding such optimizations and instead prioritizing raw compute throughput. SAFER can operate at 0.57-1.2V, reaching a F_{max} of 141 MHz@0.57V and 815 MHz@1.2V.

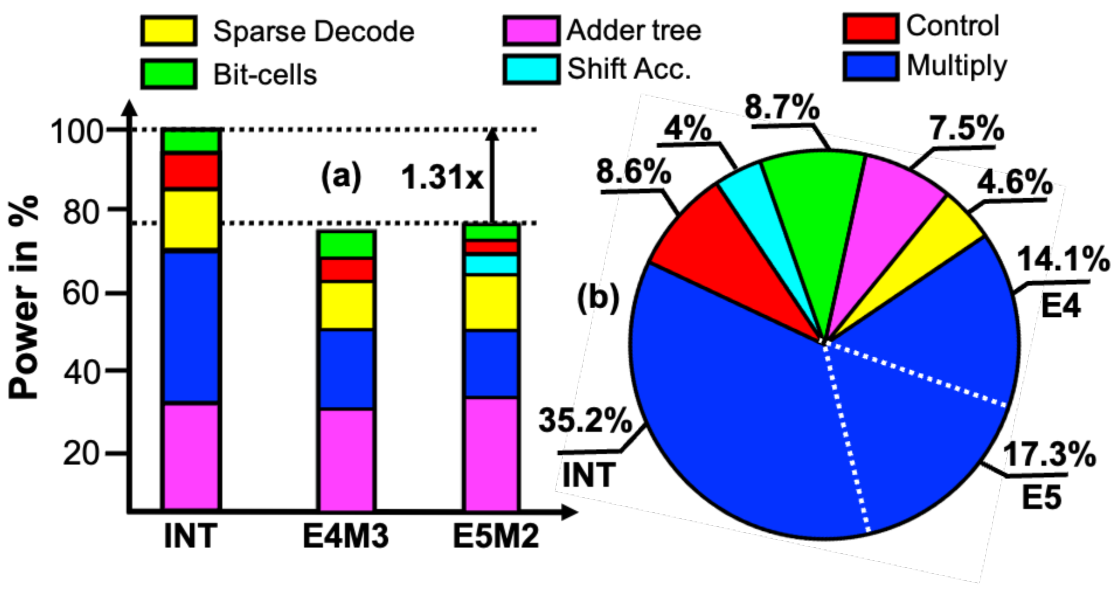


Figure 33. (A) SAFE Power, (B) Area Breakdown

Fig. 34(a) shows the voltage-frequency scaling measurements for both FP8 and INT MACs, where the CPU was set to perform NoP instruction. All measurements were done at 28^{o}C. For INT8 workloads, we use an IP toggle rate of 50% and bit-wise W sparsity of 50%. FP8 workloads use randomly generated numbers for both IPs and Ws within the representable range. The SAFE(R) achieves a peak energy efficiency of 105.7 (78.9), 78.8 (59) and 79.9 (63) TFLOPS/W for FP8 E4M3, FP8 E5M2, and INT8 MACs respectively.

To quantify how well various sparsity translates to memory footprint, we map a ResNet-18 model (∼11M parameters) trained for CIFAR-100 dataset for various 1:M sparsity. With 1:16 sparsity, only 91 SAFE macros are required for the entire model, which marks 15.5× savings in macro area, as shown in Fig. 34(b). We also validate the accuracy for ResNet-18 (DNN) and a Llama-2-7b (LLM) model for FP8 and INT8 under various sparsity to demonstrate the practical need for different N:M

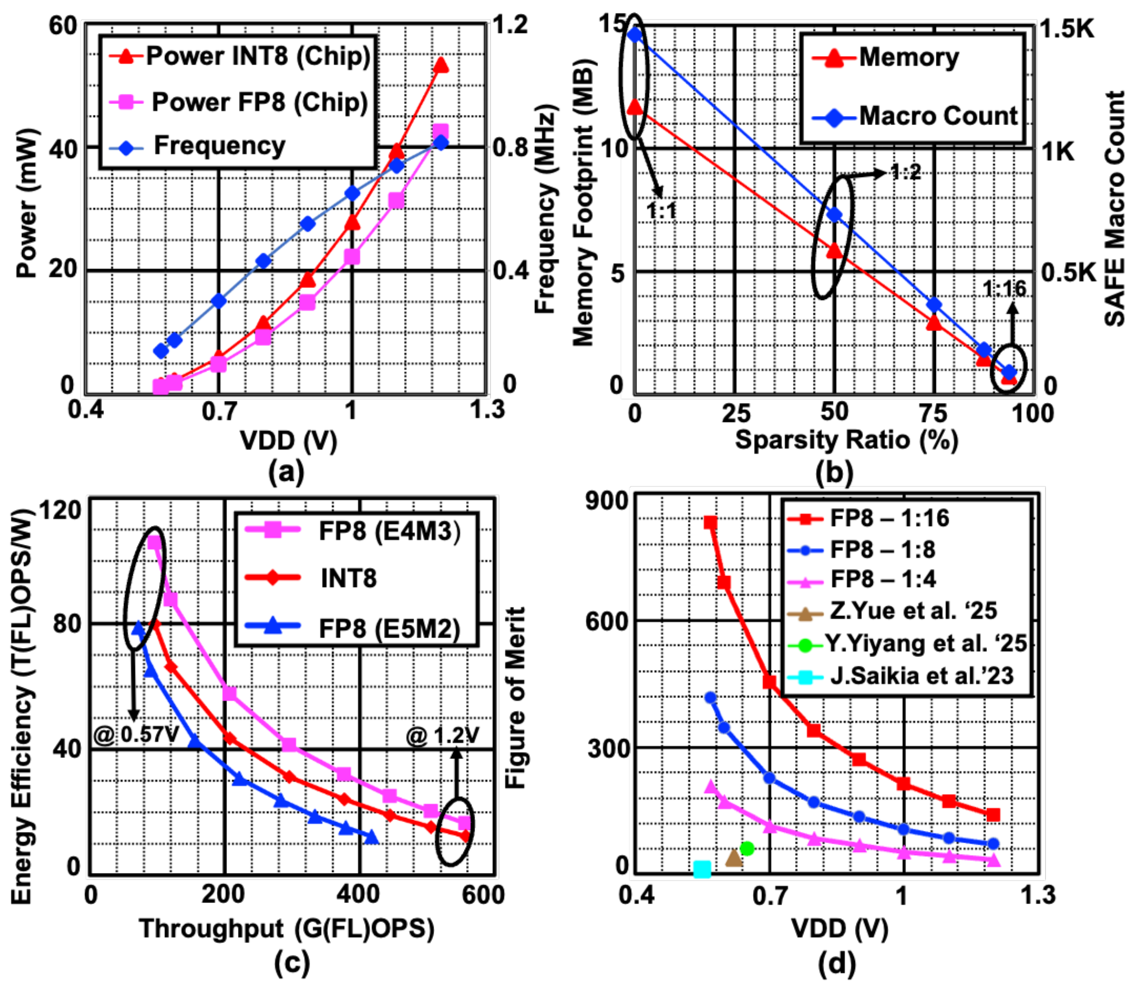


Figure 34. Power and Frequency Scaling (A), Sparsity Savings (B), Throughput and Energy Efficiency Scaling (C), Figure of Merit (D)

sparsity, as shown in Table 11. To account for reduced memory footprint due to compressed storage, we devise a figure of merit (FoM) of TOPS/W/mm^2/32K 8-bit Ws/Kb, which takes energy efficiency, chip area and memory size in Kb when 32K 8-bit weight parameters are mapped onto the CIM array. Table 12 and Fig. 34(d) show that SAFER achieves 1.4× and 14× improvements in this FoM for INT8 and FP8 workloads respectively, compared to prior SoTA digital CIMs.

Table 11. AI Model Accuracy for Various Sparsity Ratios

Model/Dataset	ResNet -18 on CIFAR-100							
Act/W Precision	INT8/INT8				FP8/FP8			
Sparsity ratio	1:1	1:4	1:8	1:16	1:1	1:4	1:8	1:16
Model accuracy (%)	76	75.7	75.3	73.9	76.5	76.5	76.2	74.8
Model	Llama-2-7b-hf (Post trained, not fine tuned)							
Act/W Precision	FP8/FP8							
Performance Score	Wikitext2 (Lower is better)				BoolQ 0-shot (Higher is better)			
Sparsity ratio	1:1	2:4	4:8	8:16	1:1	2:4	4:8	8:16
Model accuracy score	5.1	12	8.3	7.2	0.77	0.67	0.72	0.74

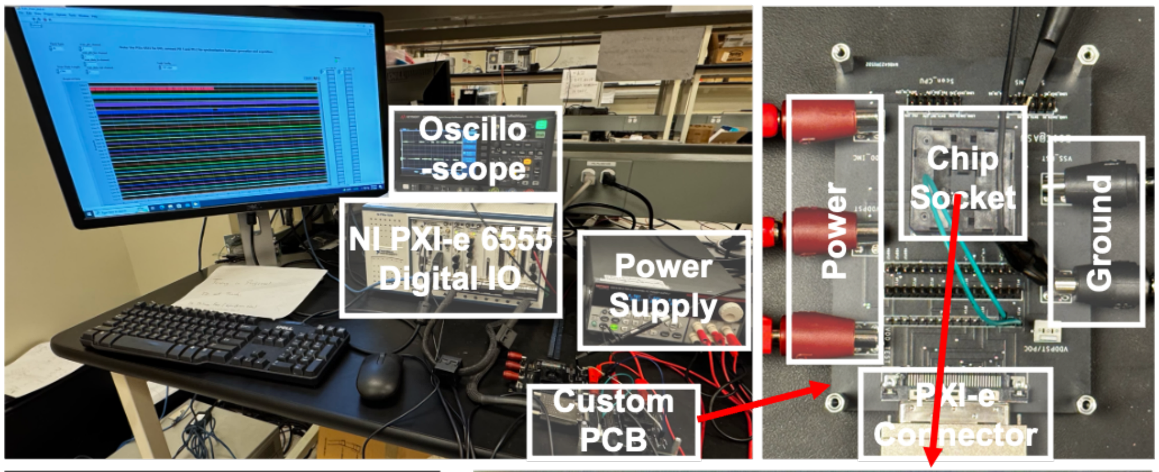
6.4 Conclusion

In this work, we prototype SAFER, an in-SRAM sparse FSD-based CIM processor, which integrates a custom RISC-V CPU enabling support for a variety of AI models from DNNs to LLMs. Through FSD, SAFE enables FP8 MACs with minimal overhead while also supporting hardware sharing between FP8 and INT8 formats. Measurement results show that SAFE achieves close to SoTA TOPS/W and TOPS/mm^2 while maintaining full accuracy for all MAC workloads. SAFE also implements sparsity in the form of compressed storage, achieving memory footprint reduction proportional to sparsity. For the proposed FoM, SAFER achieves 1.4-14× improvement compared to SoTA digital CIMs.

Table 12. Comparison with State-of-the-art Fully Digital CIM Works

		This work	Z. Yue et al., 2025	Y. Yiyang et al., 2025	A. Sridharan et al., 2024	J. Saikia et al., 2023
Technology		**28nm**	28nm	28nm	28nm	28nm
Voltage (V)		**0.57-1.2**	0.62-0.9	0.65-0.9	0.57-1.18	0.55-1.2
Frequency (MHz)		**141-815**	100-525	153-400	201-1160	650
SRAM Cell		**6T Logic**	10T	6T	10T	8T/14T
Macro size		**24Kb (16KbW)**	224Kb	32Kb	8Kb (4KbW)	4Kb
Weight/Input Precision		**INT8,FP8 (E4/5M3/2)**	FP16/8, INT8	INT4/8 FP8,BF16	INT4/8	FP8
Accuracy Loss		**No**	Yes	Yes	No	No
Sparsity Support		**CSR/N:M (50%-99%)**	Zero skip dynamic	None	CSC/N:M (1%-99.9%)	None
Energy Efficiency T(FL)OPS/W	FP8*	**16-105.7(PIM) 13-78.9(Chip)**	99.7	192.3	--	12.1
	INT8	**12.4-79.9(PIM) 10-63(Chip)**	115	71.4	57.7	--
Compute density T(FL)OPS/mm^2	FP8*	**0.3-1.79**[&]	3.3	3.23	--	0.94
	INT8	**0.24-1.3**[#]	3.81	2.34	1.2	--
Figure of Merit[$]: T(FL)OPS/W/ mm2* Ws/byte, where W is an 8-bit weight	FP8*	**8.6-51.9(1:1) 34.7-207(1:4) 138-831(1:16)**	37.42 For all sparsity	59.53 For all sparsity	--	29.2 For all sparsity
	INT8	**6.8-41.8(1:1) 27.4-167(1:4) 109-670(1:16)**	164.3 For all sparsity	84.1 For all sparsity	57.7(1:1) 230.68(1:4) 922.7(1:16)	--

TFLOPS Calculation: IP Channels x OP Channels/Latency . *FP8 metrics are shown for E4M3.
[&]INT and E5M2 Multiplier area subtracted for accurate comparison. [#]FP8 Multiplier area subtracted.
[$]FoM = Energy efficiency/die area * # of 8-bit weights/byte



Node	28nm
Macro area	0.15 mm^2
Die area	1.52 mm^2
Channels	32-IP & 8-OP per macro
Alignment	One-shot post multiplication
Buffer size	4KB
PIM size	3KB

Peak TOPS/W	16-105.7(E4M3)
	12-78.8 (E5M2)
	12.4-79.9(INT)

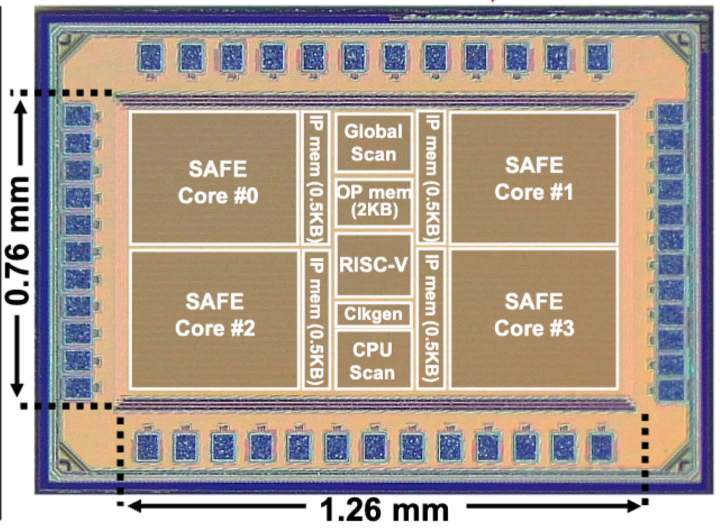


Figure 35. Testing Setup, Die-Micrograph and Chip Summary

Chapter 7

CONCLUSION

This dissertation presents a series of innovations in programmable and sparsity-aware in-memory computing (IMC) architectures, demonstrating a significant leap in flexibility, efficiency, and scalability for AI acceleration.

First this dissertation introduces a programmable PSRAM chip fabricated in TSMC 65nm CMOS technology, capable of executing a complete set of Boolean vector operations—including NOR, NAND, XOR (both 2- and 3-input), majority, and full-adder—in a single memory cycle. This design, taped out in silicon, is validated through three real-world case studies: parallel vector operations, neural networks, and data encryption.

Building on PSRAM, PS-IMC is proposed, a high-throughput IMC macro with a unique decomposed weight-precision dataflow that enables bit-parallel multiply-accumulate (MAC) operations without requiring weight replication. PS-IMC maintains 100% utilization with minimal hardware overhead and achieves state-of-the-art throughput, energy efficiency, and compute density across a wide range of MAC workloads.

To further extend IMC to algorithm-specific acceleration, a novel SRAM-based IMC architecture optimized for the Neural Belief Propagation (Neural BP) decoding algorithm is proposed. By leveraging sparsity through algorithm-hardware co-design, this IMC implementations efficiently perform sparse matrix-vector multiplications (MVMs) with either fixed or unstructured sparsity patterns. These implementations outperform state-of-the-art LDPC decoder hardware in both throughput and energy efficiency.

Expanding on sparse DNN acceleration, this dissertation introduces SP-IMC, a fully digital sparsity-integrated IMC macro that natively supports a wide range of sparse encoding formats—including Coordinate (COO), Run-Length (RL), and N:M structured sparsity—within the memory array. SP-IMC accommodates multiple data precisions for input activations (2b, 4b, 8b) and weights (4b, 8b), and supports scalable sparsity through 4-bit and 8-bit indexing, offering real-time configurability for various compression ratios. By combining compressed storage with time-multiplexed sparse decode, SP-IMC achieves high energy efficiency, and a compact area footprint across diverse CNN workloads.

Finally, this dissertation presents SAFER, an in-SRAM CIM processor that integrates a custom RISC-V CPU to support end-to-end execution of AI models ranging from DNNs to large language models (LLMs). Using Fused Multiply Accumulate, SAFER enables FP8 MACs with minimal overhead and allows compute hardware sharing across FP8 and INT8 formats. The architecture supports sparsity through compressed storage, reducing memory footprint in proportion to the degree of sparsity. Measured results show that SAFER achieves near state-of-the-art TOPS/W and TOPS/mm^2 while maintaining full accuracy, and delivers 1.4–13.8$\times$ improvement in the proposed figure of merit compared to prior digital CIM architectures. Collectively, these contributions advance the state of the art in SRAM based CIMs, offering a versatile and efficient compute substrate that is well-suited for the evolving demands of edge AI workloads.

REFERENCES

Ahn, J., S. Hong, S. Yoo, O. Mutlu and K. Choi, “A scalable processing-in-memory accelerator for parallel graph processing”, in “Proceedings of the 42nd Annual International Symposium on Computer Architecture”, ISCA ’15, p. 105–117 (Association for Computing Machinery, New York, NY, USA, 2015).

Ali, A. H., A. Sridharan, C. Guo, W. Hwang, W. Tsai, J. Zhang, Y. Chen, S. X. Wang and D. Fan, “Fp-smr: A fully digital floating-point processing-in-sas-mram for session-based recommender system”, in “Proceedings of the Great Lakes Symposium on VLSI 2025”, GLSVLSI ’25, p. 341–347 (Association for Computing Machinery, New York, NY, USA, 2025), URL https://doi.org/10.1145/3716368.3735206.

Arikan, E., “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels”, IEEE Transactions on information Theory **55**, 7, 3051–3073 (2009).

Berrou, C., A. Glavieux and P. Thitimajshima, “Near shannon limit error-correcting coding and decoding: Turbo-codes. 1”, in “Proceedings of ICC ’93 - IEEE International Conference on Communications”, vol. 2, pp. 1064–1070 vol.2 (1993).

Biswas, A. *et al.*, “Conv-sram: An energy-efficient sram with in-memory dot-product computation for low-power convolutional neural networks”, IEEE JSSC (2018).

Cammerer, S., T. Gruber, J. Hoydis and S. ten Brink, “Scaling deep learning-based decoding of polar codes via partitioning”, in “GLOBECOM 2017 - 2017 IEEE Global Communications Conference”, pp. 1–6 (2017).

Chih, Y.-D., P.-H. Lee, H. Fujiwara, Y.-C. Shih, C.-F. Lee, R. Naous, Y.-L. Chen, C.-P. Lo, C.-H. Lu, H. Mori, W.-C. Zhao, D. Sun, M. E. Sinangil, Y.-H. Chen, T.-L. Chou, K. Akarvardar, H.-J. Liao, Y. Wang, M.-F. Chang and T.-Y. J. Chang, “16.4 an 89tops/w and 16.3tops/mm2 all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications”, in “2021 IEEE International Solid- State Circuits Conference (ISSCC)”, vol. 64, pp. 252–254 (2021).

Dally, B., “ Hardware for Deep Learning ”, in “2023 IEEE Hot Chips 35 Symposium (HCS)”, pp. 1–58 (IEEE Computer Society, Los Alamitos, CA, USA, 2023), URL https://doi.ieeecomputersociety.org/10.1109/HCS59251.2023.10254716.

Das, S., “Itrs assessment and benchmarking of emerging logic devices”, in “Emerging Nanoelectronic Devices”, pp. 405–416 (2015).

Faraone, J., N. Fraser, M. Blott and P. H. Leong, “Syq: Learning symmetric quantization for efficient deep neural networks”, in “Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)”, (2018).

Fujiwara, H., H. Mori, W.-C. Zhao, M.-C. Chuang, R. Naous, C.-K. Chuang, T. Hashizume, D. Sun, C.-F. Lee, K. Akarvardar, S. Adham, T.-L. Chou, M. E. Sinangil, Y. Wang, Y.-D. Chih, Y.-H. Chen, H.-J. Liao and T.-Y. J. Chang, "A 5-nm 254-tops/w 221-tops/mm2 fully-digital computing-in-memory macro supporting wide-range dynamic-voltage-frequency scaling and simultaneous mac and write operations", in "2022 IEEE International Solid-State Circuits Conference (ISSCC)", vol. 65, pp. 1–3 (2022a).

Fujiwara, H. *et al.*, "A 5-nm 254-TOPS/W 221-TOPS/mm^2 Fully-Digital Computing-in-Memory Macro Supporting Wide-Range Dynamic-Voltage-Frequency Scaling and Simultaneous MAC and Write Operations", in "IEEE ISSCC", (2022b).

Gallager, R., "Low-density parity-check codes", IRE Transactions on information theory **8**, 1, 21–28 (1962).

Gruber, T., S. Cammerer, J. Hoydis and S. t. Brink, "On deep learning-based channel decoding", in "2017 51st Annual Conference on Information Sciences and Systems (CISS)", pp. 1–6 (2017).

Han, S., X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network", in "2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)", pp. 243–254 (2016).

Hickmann, B., J. Chen, M. Rotzin, A. Yang, M. Urbanski and S. Avancha, "Intel nervana neural network processor-t (nnp-t) fused floating point many-term dot product", in "2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)", pp. 133–136 (2020).

hsin Chen, Y., T.-J. Yang, J. S. Emer and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices", IEEE Journal on Emerging and Selected Topics in Circuits and Systems (2018).

Hu, X., H. Mun, J. Meng, Y. Liao, A. Sridharan and J.-s. Seo, "A 28nm 20.9-137.2 tops/w output-stationary sram compute-in-memory macro featuring dynamic look-ahead zero weight skipping and runtime partial sum quantization", in "2025 IEEE Custom Integrated Circuits Conference (CICC)", pp. 1–3 (2025).

Hutter, M. *et al.*, "A cryptographic processor for low-resource devices: Canning ecdsa and aes like sardines", in "IFIP", (2011).

Jiang, Z., S. Yin, J.-S. Seo and M. Seok, "C3sram: An in-memory-computing sram macro based on robust capacitive coupling computing mechanism", IEEE Journal of Solid-State Circuits **55**, 7, 1888–1897 (2020).

Kim, J.-H., J. Lee, J. Lee, J. Heo and J.-Y. Kim, “Z-pim: A sparsity-aware processing-in-memory architecture with fully variable weight bit-precision for energy-efficient deep neural networks”, IEEE Journal of Solid-State Circuits **56**, 4, 1093–1104 (2021).

Lee, C.-F. *et al.*, “A 12nm 121-TOPS/W 41.6-TOPS/mm^2 All Digital Full Precision SRAM-based Compute-in-Memory with Configurable Bit-width For AI Edge Applications”, in “IEEE Symp. VLSI Circuits”, (2022).

Li, S., D. Niu, K. T. Malladi, H. Zheng, B. Brennan and Y. Xie, “Drisa: a dram-based reconfigurable in-situ accelerator”, in “Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture”, MICRO-50 ’17, p. 288–301 (Association for Computing Machinery, New York, NY, USA, 2017).

Lin, C.-T. *et al.*, “DIMCA: An Area-Efficient Digital In-Memory Computing Macro Featuring Approximate Arithmetic Hardware in 28 nm”, IEEE JSSC (2023).

Liu, S., P. Li, J. Zhang, Y. Wang, H. Zhu, W. Jiang, S. Tang, C. Chen, Q. Liu and M. Liu, “16.2 a 28nm 53.8tops/w 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned nn and cim-based local-attention-reusable engine”, in “2023 IEEE International Solid-State Circuits Conference (ISSCC)”, pp. 250–252 (2023).

Lugosch, L. and W. J. Gross, “Neural offset min-sum decoding”, in “2017 IEEE International Symposium on Information Theory (ISIT)”, pp. 1361–1365 (2017).

Mathew, S., F. Sheikh, A. Agarwal, M. Kounavis, S. Hsu, H. Kaul, M. Anders and R. Krishnamurthy, “53gbps native gf(24)2 composite-field aes-encrypt/decrypt accelerator for content-protection in 45nm high-performance microprocessors”, in “2010 Symposium on VLSI Circuits”, pp. 169–170 (2010).

Meng, J., L. Yang, X. Peng, S. Yu, D. Fan and J.-S. Seo, “Structured pruning of rram crossbars for efficient in-memory computing acceleration of deep neural networks”, IEEE Transactions on Circuits and Systems II: Express Briefs **68**, 5, 1576–1580 (2021).

Mori, H., W.-C. Zhao, C.-E. Lee, C.-F. Lee, Y.-H. Hsu, C.-K. Chuang, T. Hashizume, H.-C. Tung, Y.-Y. Liu, S.-R. Wu, K. Akarvardar, T.-L. Chou, H. Fujiwara, Y. Wang, Y.-D. Chih, Y.-H. Chen, H.-J. Liao and T.-Y. J. Chang, “A 4nm 6163-tops/w/b $\mathbf{4790-TOPS/mm^2/b}$ sram based digital-computing-in-memory macro supporting bit-width flexibility and simultaneous mac and weight update”, in “2023 IEEE International Solid-State Circuits Conference (ISSCC)”, pp. 132–134 (2023a).

Mori, H. *et al.*, “A 4nm 6163-TOPS/W/b 4790-TOPS/mm^2/b SRAM Based Digital-Computing-in-Memory Macro Supporting Bit-Width Flexibility and Simultaneous MAC and Weight Update”, in “IEEE ISSCC”, (2023b).

Mutlu, O., "Processing data where it makes sense in modern computing systems: Enabling in-memory computation", in "2018 7th Mediterranean Conference on Embedded Computing (MECO)", pp. 8–9 (2018).

Nachmani, E., Y. Be'ery and D. Burshtein, "Learning to decode linear codes using deep learning", in "2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)", pp. 341–346 (2016).

Nachmani, E., E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein and Y. Be'ery, "Deep learning methods for improved decoding of linear codes", IEEE Journal of Selected Topics in Signal Processing **12**, 1, 119–131 (2018).

Oh, J., C.-T. Lin and M. Seok, "D6cim: 60.4-tops/w, 1.46-tops/mm2, 1005-kb/mm2 digital 6t-sram-based compute-in-memory macro supporting 1-to-8b fixed-point arithmetic in 28-nm cmos", in "ESSCIRC 2023- IEEE 49th European Solid State Circuits Conference (ESSCIRC)", pp. 413–416 (2023a).

Oh, J. *et al.*, "D6CIM: 60.4-TOPS/W, 1.46-TOPS/mm^2, 1005-Kb/mm^2 Digital 6T-SRAM-Based Compute-in-Memory Macro Supporting 1-to-8b Fixed-Point Arithmetic in 28-nm CMOS", in "IEEE ESSCIRC", (2023b).

Park, J.-S., C. Park, S. Kwon, T. Jeon, Y. Kang, H. Lee, D. Lee, J. Kim, H.-S. Kim, Y. Lee, S. Park, M. Kim, S. Ha, J. Bang, J. Park, S. Lim and I. Kang, "A multi-mode 8k-mac hw-utilization-aware neural processing unit with a unified multi-precision datapath in 4-nm flagship mobile soc", IEEE Journal of Solid-State Circuits **58**, 1, 189–202 (2023).

Ryu, S. *et al.*, "BitBlade: Energy-Efficient Variable Bit-Precision Hardware Accelerator for Quantized Neural Networks", IEEE JSSC (2022).

Saikia, J., A. Sridharan, I. Yeo, S. Venkataramanaiah, D. Fan and J.-S. Seo, "Fp-imc: A 28nm all-digital configurable floating-point in-memory computing macro", in "ESSCIRC 2023- IEEE 49th European Solid State Circuits Conference (ESSCIRC)", pp. 405–408 (2023).

Seo, J.-s., J. Saikia, J. Meng, W. He, H.-s. Suh, Anupreetham, Y. Liao, A. Hasssan and I. Yeo, "Digital versus analog artificial intelligence accelerators: Advances, trends, and emerging designs", IEEE Solid-State Circuits Magazine **14**, 3, 65–79 (2022).

Seshadri, V., D. Lee, T. Mullins, H. Hassan, A. Boroumand, M. A. Kim, Jeremieand Kozuch, O. Mutlu, P. B. Gibbons and T. C. Mowry, "Ambit: in-memory accelerator for bulk bitwise operations using commodity dram technology", in "Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture", MICRO-50 '17, p. 273–287 (Association for Computing Machinery, New York, NY, USA, 2017).

Sharma, H. *et al.*, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network", in "ACM/IEEE ISCA", (2018).

Sridharan, A., S. Angizi, S. K. Cherupally, F. Zhang, J.-S. Seo and D. Fan, "A 1.23-ghz 16-kb programmable and generic processing-in-sram accelerator in 65nm", in "ESSCIRC 2022- IEEE 48th European Solid State Circuits Conference (ESSCIRC)", pp. 153–156 (2022a).

Sridharan, A., J. Saikia, Anupreetham, F. Zhang, J.-S. Seo and D. Fan, "Ps-imc: A 2385.7-tops/w/b precision scalable in-memory computing macro with bit-parallel inputs and decomposable weights for dnns", IEEE Solid-State Circuits Letters **7**, 102–105 (2024a).

Sridharan, A., F. Zhang and D. Fan, "Mnm: A fast and efficient min/max searching in mram", in "Proceedings of the Great Lakes Symposium on VLSI 2022", GLSVLSI '22, p. 39–44 (Association for Computing Machinery, New York, NY, USA, 2022b).

Sridharan, A., F. Zhang, J.-S. Seo and D. Fan, "Sp-imc: A sparsity aware in-memory-computing macro in 28nm cmos with configurable sparse representation for highly sparse dnn workloads", in "2024 IEEE Custom Integrated Circuits Conference (CICC)", pp. 1–2 (2024b).

Sridharan, A., F. Zhang, Y. Sui, B. Yuan and D. Fan, "Dspimm: A fully digital sparse in-memory matrix vector multiplier for communication applications", in "2023 60th ACM/IEEE Design Automation Conference (DAC)", pp. 1–6 (2023).

Valavi, H. *et al.*, "A 64-tile 2.4-mb in-memory-computing cnn accelerator employing charge-domain compute", IEEE JSSC (2019).

Wang, D., C.-T. Lin, G. K. Chen, P. Knag, R. K. Krishnamurthy and M. Seok, "Dimc: 2219tops/w 2569f2/b digital in-memory computing macro in 28nm based on approximate arithmetic hardware", in "2022 ISSCC", vol. 65, pp. 266–268 (2022).

Wang, J. *et al.*, "A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing", IEEE JSSC (2020).

Wang, Z., P. S. Nalla, G. Krishnan, R. V. Joshi, N. C. Cady, D. Fan, J.-s. Seo and Y. Cao, "Digital-assisted analog in-memory computing with rram devices", in "2023 International VLSI Symposium on Technology, Systems and Applications (VLSI-TSA/VLSI-DAT)", pp. 1–4 (2023).

Yin, S., Z. Jiang, J.-S. Seo and M. Seok, "Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks", IEEE Journal of Solid-State Circuits **55**, 6 (2020).

Yuan, Y., B. Zhang, Y. Yang, Y. Luo, Q. Chen, S. Lv, H. Wu, C. Ma, M. Li, J. Yue, X. Wang, G. Xing, P.-I. Mak, X. Li and F. Zhang, "14.5 a 28nm 192.3tflops/w accurate/approximate dual-mode-transpose digital 6t-sram cim macro for floating-point edge training and inference", in "2025 IEEE International Solid-State Circuits Conference (ISSCC)", vol. 68, pp. 258–260 (2025).

Yue, J., C. He, Z. Wang, Z. Cong, Y. He, M. Zhou, W. Sun, X. Li, C. Dou, F. Zhang, H. Yang, Y. Liu and M. Liu, "A 28nm 16.9-300tops/w computing-in-memory processor supporting floating-point nn inference/training with intensive-cim sparse-digital architecture", in "2023 IEEE International Solid-State Circuits Conference (ISSCC)", pp. 1–3 (2023).

Yue, J. *et al.*, "14.3 a 65nm computing-in-memory-based cnn processor with 2.9-to-35.8 tops/w system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse", in "IEEE ISSCC", (2020).

Yue, Z., X. Xiang, Y. Wang, R. Guo, H. Han, S. Wei, Y. Hu and S. Yin, "14.4 a 51.6tflops/w full-datapath cim macro approaching sparsity bound and <2-30 loss for compound ai", in "2025 IEEE International Solid-State Circuits Conference (ISSCC)", vol. 68, pp. 1–3 (2025).

Zhang, F., S. Angizi and D. Fan, "Max-pim: Fast and efficient max/min searching in dram", in "2021 58th ACM/IEEE Design Automation Conference (DAC)", pp. 211–216 (2021a).

Zhang, F., A. Sridharan, W. He, I. Yeo, M. Liehr, W. Zhang, N. Cady, Y. Cao, J.-S. Seo and D. Fan, "A 65-nm rram compute-in-memory macro for genome processing", IEEE Journal of Solid-State Circuits **59**, 7, 2093–2104 (2024a).

Zhang, F., A. Sridharan, W. Tsai, Y. Chen, S. X. Wang and D. Fan, "Efficient memory integration: Mram-sram hybrid accelerator for sparse on-device learning", in "Proceedings of the 61st ACM/IEEE Design Automation Conference", DAC '24 (Association for Computing Machinery, New York, NY, USA, 2024b).

Zhang, J., Z. Wang and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6t sram array", IEEE Journal of Solid-State Circuits **52**, 4, 915–924 (2017).

Zhang, J.-F., C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler and Z. Zhang, "Snap: An efficient sparse neural acceleration processor for unstructured sparse deep neural network inference", IEEE Journal of Solid-State Circuits **56**, 2, 636–647 (2021b).

Zhang, Y., L. Xu, Q. Dong, J. Wang, D. Blaauw and D. Sylvester, "Recryptor: A reconfigurable cryptographic cortex-m0 processor with in-memory and near-memory computing for iot security", IEEE Journal of Solid-State Circuits **53** (2018).

Zhou, A., Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun and H. Li, "Learning n:m fine-grained structured sparse neural networks from scratch", URL https://arxiv.org/abs/2102.04010 (2021).

APPENDIX A

PREVIOUS PUBLISHED WORKS

Five chapters are based on the previously published works of the first author, listed below:

Chapter 2: A. Sridharan, S. Angizi, S. K. Cherupally, F. Zhang, J. -S. Seo and D. Fan, "A 1.23-GHz 16-kb Programmable and Generic Processing-in-SRAM Accelerator in 65nm," ESSCIRC 2022- IEEE 48th European Solid State Circuits Conference (ESSCIRC), Milan, Italy, 2022, pp. 153-156, doi: 10.1109/ESSCIRC55480.2022.9911440.

Chapter 3: A. Sridharan, J. Saikia, Anupreetham, F. Zhang, J. -S. Seo and D. Fan.,"PS-IMC: A 2385.7-TOPS/W/b Precision Scalable In-Memory Computing Macro With Bit-Parallel Inputs and Decomposable Weights for DNNs," in IEEE Solid-State Circuits Letters, vol. 7, pp. 102-105, 2024, doi: 10.1109/LSSC.2024.3369058

Chapter 4: A. Sridharan, F. Zhang, Y. Sui, B. Yuan and D. Fan., "DSPIMM: A Fully Digital SParse In-Memory Matrix Vector Multiplier for Communication Applications," 2023 60th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 2023, pp. 1-6, doi: 10.1109/DAC56929.2023.10247829

Chapter 5: A. Sridharan, F. Zhang, J. -S. Seo and D. Fan., "A Sparsity Aware In-Memory-Computing Macro in 28nm CMOS with Configurable Sparse Representation for Highly Sparse DNN Workloads," 2024 IEEE Custom Integrated Circuits Conference (CICC), Denver, CO, USA, 2024, pp. 1-2, doi: 10.1109/CICC60959.2024.10529009.

Chapter 6: A. Sridharan, A.H. Ali, Y. Lee, A. Anupreethem, Y. Liu, J. Zhang, J. -S. Seo, D. Fan., "SAFER: Sparsity Integrated Compute-in-Memory AI Accelerator with a Fused Dot-Product Engine and a RISC-V CPU" IEEE European Solid-State Electronics Research Conference (ESSERC), September 2025, accepted for publication.

APPENDIX B

PERMISSION STATEMENT FOR THE PREVIOUS PUBLISHED WORKS

The author of this dissertation thesis, Amitesh Sridharan, confirms that all included published works have been granted permission by all co-authors.